Universidad Autónoma de Madrid
Departamento de Ingeniería Informática

Aportes a la Reducción de Consumo en FPGAs

Tesis Doctoral

Autor: Gustavo Sutter
Director: Eduardo Boemo Scalvinoni

Escuela Politécnica Superior
Enero de 2005

Resumen:

En esta tesis se desarrolla una metodología para reducción de consumo en FPGAs a nivel topológico, arquitectural y algorítmico pues el consumo, al igual que el área o la velocidad, puede ser atacado en todos los niveles de la jerarquía de diseño.

En primer lugar se revisan las principales técnicas de reducción de consumo en tecnología VLSI y el caso particular de FPGAs. Se realiza una clasificación de las diferentes ideas y se llevan a cabo algunos experimentos generales para su comprobación. A continuación, se profundiza en algunas técnicas de aplicación universal tales como la segmentación (*pipelining*), o la inhabilitación de bloques. Estos conocimientos son aplicados posteriormente para reducir el consumo en máquinas de estados y en diferentes bloques aritméticos. Para las primeras, se estudian diferentes métodos de codificación y partición para diversos circuitos de prueba y número de estados. En lo referente a bloques aritméticos, los experimentos abarcan las operaciones de suma, multiplicación, multiplicación modular y división, teniendo en cuenta diferentes ideas algorítmicas y estrategias de implementación.

Las conclusiones expuestas en esta tesis se sustentan sobre un exhaustivo trabajo experimental. Se han construido y medido más de 600 circuitos en diversos modelos de FPGAs Xilinx que en conjunto abarcan más de una década de esta tecnología. Los resultados finales se resumen en una serie de recomendaciones y reglas de reducción de consumo de utilidad a nivel de diseñador.

Summary:

In this thesis, a methodology for power reduction in FPGAs at the topological, architectural, and algorithmic levels is developed. The power consumption, as the area or the speed, can be modified in all the levels of the design hierarchy.

In the first place, the main low-power design techniques in technology VLSI are revised, and the particular case of FPGAs is analyzed. A classification of the different ideas is done, and some general experiments are performed to verify them. Next, a deeper study of some techniques of universal application as the pipelining, or the disabling of blocks is done. Later, this knowledge is applied in order to reduce the consumption in state machines and different arithmetic blocks. For the first ones, different coding methods and partition schemes are studied for diverse benchmark circuits and number of states. For arithmetic blocks, the experiments embrace the operations of addition, multiplication, module multiplication, and finally division, taking into account different algorithmic ideas and implementation strategies.

The conclusions exposed in this thesis are sustained by an exhaustive experimental work. More than 600 circuits have been built and measured using diverse models of Xilinx FPGAs that comprise more than a decade of this technology. As final result, a summary of recommendations and low-power design rules at designer's level are established.

Agradecimientos

Indudablemente Eduardo ha sido mucho más que un director de tesis. Su amistad y compañerismo, exceden su ya invaluable aporte técnico y académico. En aquellos primeros y lejanos primeros tiempos en Madrid, Eduardo hizo mucho más de lo que cualquiera pueda imaginar. Mi gratitud y reconocimiento son difíciles de plasmar en unas cuantas líneas.

Desde aquellos primeros momentos, Sergio ha sido un amigo de hierro y su inmensa capacidad técnica y científica no solo han resuelto innumerables problemas cotidianos y concretos sino que han sido una guía para mí.

A Elías compañero de peripecias y avatares y con quien hicimos nuestros primeros experimentos y siempre discutimos resultados. A los compañeros de laboratorio por su paciencia y amistad: Paco, Javier, Guillermo, Juan, Ivan, Miguel, Alberto(s), Tani, y tantos otros que han pasado por aquí.

No quiero dejar de agradecer a Juana, Angel, Eugenio, Conrado y Pablo por su siempre buena predisposición a resolver los problemas diarios con la burocracia y la docencia.

Índice

Resumen: I

Summary: III

Índice VII

Lista de Figuras XIX

Lista de Tablas XXIX

Lista de Acrónimos XXXIII

Anglicismos, barbarismos, neologismos y otros defectos XXXVII

Capítulo 1:
Introducción y Motivación 1

1.1 Reducción y estimación del consumo 1

1.2 Marco tecnológico 2

1.3 Selección de técnicas de reducción de consumo 4

1.4 Objetivos de ésta tesis 5

1.5 Organización y lectura de ésta tesis 6

1.6 Referencias del capítulo 8

Capítulo 2:
Consumo en circuitos CMOS y su efecto en FPGAs 11

2.1. Consumo en circuitos CMOS 11

2.1.1 Consumo estático 12

2.1.2 Consumo dinámico 13
2.2. Espacio de diseño para bajo consumo 15
2.2.1 Tensión de alimentación. 15
2.2.2 Capacidad 16
2.2.3 Actividad del circuito 17
2.3 Temas destacados en el diseño para bajo consumo 19
2.4. Nivel de proceso y tecnología 21
2.4.1 Encapsulado (*package*) 21
2.4.2 Proceso de fabricación 22
2.4.2.1 Optimización de la tensión umbral (*Threshold Voltaje-Vt*) 22
2.4.2.2 Reducción en la tecnología (*Technological scaling*) 22
2.4.2.3 Trazado (*Layout*) 23
2.4.2.4 Dimensiones de los transistores (*Transisitor sizing*) 23
2.5. Nivel de implementación 24
2.5.1 Descomposición tecnológica y mapeo 24
2.5.2 Reordenar las entradas 24
2.5.3 Reducción de *glitches* 25
2.5.4 Concurrencia y redundancia 26
2.6. Nivel de arquitectura y sistema 27
2.6.1 Procesamiento concurrente 27
2.6.1.1 Paralelismo 27
2.6.1.2 Segmentación (*pipeline*) 29
2.6.2 Manejo de potencia (*power managemet*) 30
2.6.3 Particionado 31
2.6.4 Representación de los datos 32
2.7. Nivel algorítmico 34
2.7.1. Algoritmos para bajo consumo. 34

Complejidad 35
Precisión 35
Regularidad 35
2.7.2 Algoritmos para arquitecturas de bajo consumo 36
Concurrencia 36
Modularidad y localidad 37
2.8. Nivel sistema 37
2.8.1 Optimización de memoria 37
2.8.2 Particionado hardware-software 37
2.8.3 Optimización a nivel de instrucciones 38
2.8.4 Manejo dinámico del consumo 38
2.8.5 Minimización del consumo en las interfaces 38
2.9. Otros conceptos en el tratamiento del consumo 39
2.9.1. Energía vs. potencia 39
2.9.2. Proyecciones para la tecnología de semiconductores 40
2.10 Desglose del consumo en FPGAs 41
2.11 Resumen del capítulo 43
2.12 Referencias del capítulo 44

Capítulo 3:
Experimentos sobre Bajo Consumo 49

3.1 Experimentos generales 49
3.1.1 Relación de la frecuencia y tensión en el consumo 50
Dispositivo XC4K 50
Dispositivo Virtex 51
Dispositivo Virtex II 51

3.1.2 Corriente Estática 52
3.1.3 Corriente de Sincronización 54
3.1.4 Conclusiones sobre experimentos generales 59
3.2 Relación entre velocidad y consumo en FPGAs 60
3.2.1 Circuitos de prueba 62
3.2.2 Resultados experimentales de la relación velocidad - consumo 63
3.2.2.1 Correlación velocidad-consumo. Efecto de las diferentes topologías 63
3.2.2.2 Correlación velocidad-consumo para implementaciones dentro de la misma topología 65
3.2.2.3 Importancia de los *glitches* 67
3.2.2.4 Correlación área-velocidad-consumo (*Area-Time-Power*) 70
3.2.3 Conclusiones de la relación velocidad - consumo. 71
3.3 Conmutación de los datos de entrada (propiedad conmutativa y diseño de bajo consumo) 72
3.3.1 Circuitos de pruebas familia 4K 74
3.3.2 Resultados experimentales familia 4K 74
3.3.2.1 Esquema de medición 75
3.3.2.2 Resultados 75
3.3.3 Circuitos de pruebas familia Virtex 80
3.3.4 Resultados experimentales familia Virtex 80
3.3.4.1 Relación retardo - consumo 82
3.3.4.2 Uso de herramientas de estimación de consumo 82
3.3.5 Conclusiones de la conmutación de datos 86
3.4 Efecto de la segmentación en el consumo 87
3.4.1 Medidas sobre XC4K 87
3.4.2 Mediciones sobre Virtex 90
3.4.2.1 Actividad espuria (*glitches*) 92

3.4.2.2 Uso de herramienta de estimación de consumo 92
3.4.3 Mediciones sobre Virtex II 94
3.4.4 Conclusiones sobre la segmentación 96
3.5 Observaciones en la disminución del consumo registrando las entradas y salidas. 97
3.5.1 Experiencias sobre la familia XC4K 97
3.5.2 Experiencias sobre la familia Virtex 100
3.5.3 Experiencias sobre la familia Virtex II 101
3.5.4 Conclusiones del registro de entradas y salidas 102
3.6 Reducción del consumo por deshabilitación de partes inactivas del circuito 103
3.6.1 Introducción 103
3.6.2 Técnicas de inhabilitación en FPGAs 104
3.6.2.1 Bloqueo por inhabilitación del reloj 105
3.6.2.2 Bloqueo de datos mediante la señal de habilitación 105
3.6.2.3 Bloqueo de datos con latches 106
3.6.2.4 Bloqueo de datos con puertas lógicas 107
3.6.2.5 Uso de *buffers* de tercer estado 107
3.6.3 Resultados de las técnicas de deshabilitación 108
3.6.4 Conclusiones de las técnicas de deshabilitación 110
3.7 Conclusiones y recomendaciones a nivel diseñador 111
3.8 Referencias del Capítulo 116

Capítulo 4:
Reducción de Consumo en Máquinas de Estado 121

4.1 Estrategias para reducción de consumo en máquinas de estados 121
4.1.1 Minimización de estados 122
4.1.2 Asignación de estados (*state assignment / encoding*) 122

4.1.2.1 Enfoques tradicionales para la codificación de estados 122
4.1.2.2 Aproximaciones para asignación de estado de bajo consumo 122
4.1.2.3 Asignación de estados en FPGAs 123
4.1.3 Descomposición de maquinas de estado (*FSM partitioning o descomposition*) 123
4.2 Experimentos sobre codificación de máquinas de estados en FPGAs 124
4.2.1. Experimentos 124
4.2.2. Resultados Experimentales 129
4.2.3 Conclusiones sobre codificación de máquinas de estados en FPGAs 134
4.3 Partición de máquinas de estado en FPGAs 135
4.3.1 Alternativas de bajo consumo en FSMs 135
4.3.2 Decomposición o partcionado de FSMs 136
4.3.2.1 Modelo general para la partición de máquinas de estado 137
4.3.2.2 Partición ortogonal de máquinas de estado 138
4.3.3 Técnicas de descomposición para bajo consumo 139
4.3.4 Una arquitectura de descomposición de FSMs para FPGAs 141
4.3.5. Cálculo de probabilidades de transición en un STG 142
4.3.6. Particionando la FSM en submáquinas 144
4.3.7 Métodos para bloquear los datos 146
4.3.8 Síntesis de la máquina de estados 146
4.3.9 Experimentos con partición de máquinas de estado 146
4.3.10 Resultados experimentales de la partición de máquinas de estado 148
4.3.11 Conclusiones de la partición de máquina de estado en FPGAs 154
4.4 Recomendaciones para la reducción de consumo en máquinas de estado 155
4.5 Referencias del Capítulo 156

Capítulo 5:
Experimentos sobre Bloques Aritméticos 161

5.1 Introducción 161

5.2 Multiplicación modular 162

5.2.1 Introducción a la multiplicación modular 162

5.2.2 Algoritmos para la multiplicación modular 162

5.2.2.1 Multiplicación y reducción (*Multiply and Reduce*) 163

5.2.2.1 Suma y desplazamiento (*shift and add*) 164

5.2.2.3 Multiplicación de Montgomery 166

5.2.3 Detalle de la síntesis 168

5.2.3.1 Multiplicación y reducción 168

5.2.3.2 Sumas y desplazamientos 170

5.2.3.3 Multiplicación de Montgomery 171

5.2.3.4 Comparación en área y velocidad 171

5.2.4 Realización secuencial 173

5.2.5 Consumo de potencia en la multiplicación modular 175

5.2.5.1 Consumo en implementaciones combinacionales 175

5.2.5.2 Consumo en implementaciones secuenciales 176

5.2.6 Sugerencias para la multiplicación modular 178

5.3. Sumadores de alta velocidad 179

5.3.1 Sumador *ripple-carry* 179

5.3.2 Sumador *carry-skip* 180

5.3.3 Resultados experimentales del sumador *carry-skip* 184

5.3.3.1 Resultados en área - velocidad 184

5.3.3.2 Resultados en consumo 187

5.3.3.3 Justificación de los resultados en consumo 188

5.3.4 Conclusiones sobre algoritmos sumadores 189

5.4 Algoritmos y arquitecturas para la división entera 191

5.4.1 Algoritmos de división para números naturales y enteros 191

5.4.1.1 Algoritmo con restauración (*restoring division algorithm*) 192

5.4.1.2 Algoritmo sin restauración (*non-restoring division algorithm*) 194

5.4.1.3 Otros algoritmos para números enteros 196

5.4.2 Implementaciones en FPGA para números naturales y enteros 198

5.4.2.1 Algoritmo con restauración en base dos 198

5.4.2.2 Algoritmo sin restauración en base dos 199

5.4.3 Implementaciones para divisores de números enteros 201

5.4.3.1 Resultados del consumo para divisores enteros 201

5.4.3.2 Resultados en la actividad interna en los divisores enteros 203

5.4.4 Conclusiones para la división de números enteros 206

5.5 Algoritmos y arquitecturas para la división de números fraccionarios 207

5.5.1 Algoritmos de división fraccionaria 207

5.5.2 Algoritmos con y sin-restauración (*restoring and non-restoring algorithm*) 208

5.5.3 Algoritmos SRT 208

5.5.4 Implementaciones combinacionales en FPGAs para números fraccionarios 211

5.5.4.1 Algoritmo con restauración y sin restauración 211

5.5.4.2 Algoritmo SRT base 2 resto complemento a dos 211

5.5.4.3 Algoritmo SRT base 4 resto complemento a dos 215

5.5.4.4 Algoritmo SRT base 8 resto complemento a dos 219

5.5.4.5 Algoritmo SRT base 16 resto complemento a dos 220

5.5.4.6 Algoritmo SRT base 2 resto *carry-save* 222

5.5.5 Arquitecturas segmentadas 226

5.5.6 Circuitos Secuenciales 228

5.5.7 Resultados en área y velocidad para los divisores para números fraccionarios 229

5.5.7.1 Resultados en implementaciones combinacionales 230
5.5.7.2 Resultados en implementaciones segmentadas 233
5.5.7.3 Resultados en implementaciones secuenciales 235
5.5.7.4 Comparación de resultados en área y velocidad 237
5.5.8 Resultados del consumo para divisores de números fraccionarios 240
5.5.8.1 Resultados en circuitos combinacionales 240
5.5.8.2 Resultados en circuitos segmentados 242
5.5.8.3 Resultados en circuitos iterativos 244
5.5.8.4 Comparaciones arquitecturales en el consumo 245
5.5.9 Conclusiones de la división de números fraccionarios 246
5.6 Conclusiones del Capítulo 247
5.7 Referencias de Capítulo 251

Capítulo 6:
Conclusiones y Futuros Trabajos 255

6.1 Conclusiones y Aportaciones 255
6.1.1 Recomendaciones para la reducción de consumo en FSMs 256
6.1.2 Observaciones y consejos a nivel topológico 256
6.1.3 Resultados a nivel algorítmico 260
6.1.4 Bloques aritméticos 261
6.1.5 Conclusiones sobre la herramientas de estimación de consumo 263
6.2 Trabajos futuros 264
6.3 Publicaciones relacionadas con éste trabajo 265
6.4 Reglas empíricas para el diseño de bajo consumo en FPGAs 268

Apéndice A:
Placa de prueba familia XC4K 271

A.1 Introducción 271
A.2 Características de la placa de prueba 271
A.3 Dispositivos utilizados 273
A.4 Conexión de la placa de pruebas 274
A.5 Método de medición de consumo 276
A.6 Generador de vectores 277
A.7 Referencias del apéndice 279

Apéndice B:
Placa de prueba AFX (Virtex) 281

B.1 Introducción 281
B.2 Características de la placa de prueba AFX 282
B.3 Dispositivos Virtex utilizados 283
B.4 Conexión de la placa de pruebas 283
B.5 Metodología de medición de consumo 285
B.6 Referencias del apéndice 287

Apéndice C:
Placa de prueba para VIRTEX II 289

C.1 Introducción 289
C.2 Características de la placa de prueba para Virtex II 290
C.3 Dispositivos Virtex II utilizados 292

C.4 Conexión de la placa de pruebas 292
C.5 Metodología de medición de consumo 293
C.6 Referencias del apéndice 295

Apéndice D:
Traductor Kiss2VHDL 297

D.1 Introducción 297
D.2 El formato KISS 297
D.3 Bancos de pruebas (*Benchmarks*) 299
D.3 Traductor Kiss2VHDL 300
D.4 Referencias del Apéndice 303

Apéndice E:
Particionador de máquinas de estado: *Part_FSM* 305

E.1. Introducción 305
E.2 Estructuras de datos 305
E.3 Cálculo de probabilidad estática 306
E.4 Partición de la máquina de estados 307
E.5 Generación del código VHDL 309

Apéndice F:
Utilización de la herramienta de estimación de consumo Xpower 317

F.1. Introducción 317

F.2. Uso de Xpower con FPGAs 317
F.2.1 Generación del fichero VCD. 318
F.2.2 Uso de la interfaz gráfica Xpower 320
F.2.3 Uso de la línea de comando Xpwr 320
F.3 *Scripts* para el uso del Xpower 321
F.3.1 *Script* para la implementación del diseño 321
F.3.2 *Script* para la simulación y generación del fichero VCD 322
F.4 Cometarios y conclusiones 323
F.5 Referencias del apéndice 325

Lista de Figuras

Figura 2.1. Inversor CMOS 12

Figura 2.2. Consumo estático en un inversor CMOS 13

Figura 2.3. Consumo dinámico en un cambio de estado del inversor 14

Figura 2.4. Relación energía-tensión y retardo-tensión respectivamente 15

Figura 2.5. Rutado simplificado en dispositivos tipo FPGA 16

Figura 2.6. Interpretación de la actividad en circuitos síncronos: Glitches 18

Figura 2.7. Transiciones espureas (*glitches*). a. Producidos por el desbalance en el arribo de las señales. b. Efecto avalancha debido al aumento de la profundidad lógica 19

Figura 2.8. Jerarquía en el espacio de diseño 20

Figura 2.9. Dependencia del orden de entrada para la reducción de actividad 25

Figura 2.10. Estructuras de cascada y balanceadas. 26

Figura 2.11. Desbalanceo en las pistas. El ejemplo se muestran 4 conexiones adyasentes a una LUT en un dispositivo Virtex 26

Figura 2.12. Reducción de tensión y paralelismo para bajo consumo [Lan96] 28

Figura 2.13. Reducción de tensión y *pipeline* para bajo consumo [Lan96]. 30

Figura 2.14. Ejemplos de cambio de signo en complemento a dos (C2) y signo magnitud (SVA) 33

Figura 2.15. Gráfica del consumo y la energía 40

Figura 2.16. Proyecciones para la industria de semiconductores. 41

Figura 2.17. Distribución del consumo en XC4003 según [Kus98] 42

Figura 2.18. Distribución del consumo en XC4k-E según [Gar00]. 42

Figura 2.19. Distribución del consumo en Virtex 2 según [Sha02] 42

Figura 3.1. Medidas de la relación lineal de la frecuencia con el consumo y cuadrática con la tensión de alimentación en la familia XC4K.....50

Figura 3.2. Medidas de la relación lineal de la frecuencia y cuadrática con la tensión en dispositivos de la familia Virtex.....51

Figura 3.3. Corriente de sincronización en función de la cantidad de registros en diferentes dispositivos de la familia XC4K.....54

Figura 3.4. Relación del consumo de sincronización respecto del de la ruta de datos en XC4K.....55

Figura 3.5. Consumo de sincronización en función de la cantidad de registros en el dispositivo Virtex XCV800PQ240-4.....56

Figura 3.6. Distribución del reloj en Virtex II [Xil04b].....57

Figura 3.7. Consumo por flip-flop en el dispositivo Virtex II XC2V1500fg676-6, expresados μW/MHz.....57

Figura 3.8. Relación ancho de banda-consumo para multiplicadores segmentados [Boe96].....61

Figura 3.9. Consumo dinámica de los multiplicadores medidos.....63

Figura 3.10. Relación velocidad-consumo para la misma topología para el multiplicador Guild.....64

Figura 3.11. Relación velocidad-consumo dentro de la misma topología. (multipl. VHDL comportamental).....65

Figura 3.12. Estudio de las fluctuaciones en el consumo para circuitos de similares velocidades (Multiplicador Guid).....66

Figura 3.13. Transiciones a la salida de los multiplicadores seleccionados y obtenida por Simulación.....67

Figura 3.14. Transiciones a la salida de los multiplicadores, obtenido por medición con el analizador lógico.....68

Figura 3.15. Relación entre actividad a la salida de los multiplicadores y consumo para implementaciones con similar velocidad.....69

Figura 3.16. Relación entre actividad a la salida de los multiplicadores y consumo para implementaciones con similar velocidad. 70

Figura 3.17. Arreglo segmentado Hatamian-Cash de 4 bits con entradas *b* locales y *a* globales [Boe95]. 73

Figura 3.18. Reducción de consumo. Areglos *Hatamian-Cash*.en XC4010PC84. 76

Figura 3.19. Reducción de consumo en función de la profundidad de lógica de los circuitos. XC4010PC84 76

Figura 3.20. Efecto de las líneas globales. XC4010PC84. 78

Figura 3.21. Reducción de consumo para diferentes topologías. XC4010PC84. 79

Figura 3.22. Reducción de consumo para diferentes circuitos familia XC40XXPC84 (multiplicador Hatamian-8) 79

Figura 3.24. Diferencia en el consumo estimado y medido multiplicadores 16 bits..... 85

Figura 3.25. Diferencia en el consumo estimado y medido multiplicadores de 32 bits 85

Figura 3.26. Consumo respecto de la profundidad lógica para diferentes dispositivos de la familia XC4K 88

Figura 3.27. Relación del consumo de sincronización y de la ruta de datos en función de la profundidad lógica en XC4010 89

Figura 3.28. Consumo dinámico respecto de la profundidad lógica en Virtex 91

Figura 3.29. Consumo, área, retardo normalizados respecto de la profundidad lógica en Virtex 91

Figura 3.30. Consumo medido y estimado en función de la profundidad lógica. 93

Figura 3.31. Consumo respecto de la profundidad lógica en Virtex II 95

Figura 3.32. Distribución de la línea global de reloj. a)Flip-flop en las patas de entrada salida; b) usando los Flip-Flop de los CLBs 97

Figura 3.33. Distribución de la línea de entrada a<7> con *fan-out* de 16. a)Flip-flop en las patas de entrada salida; b) usando los Flip-Flop de los CLBs 98

Figura 3.34. Consumo normalizado con registros en IOBs y CLBs. 99

Figura 3.35. Ejemplo de la utilización de aislamiento de operandos 103

Figura 3.36. Esquema para la deshabilitación de partes del circuito 104

Figura 3.37. Riesgo de doble captura en la implementación de gated clock en FPGAs [Xil03a] 105

Figura 3.38. Forma de implementar el bloqueo de datos equivalente al gated clock según Xilinx [Xil03a]. 106

Figura 3.39. Inhabilitación de las entradas con puertas. a. puertas AND. b. Puertas OR. 107

Figura 3.40. Circuito original para la evaluación de las técnicas de deshabilitación de partes inactivas 108

Figura 3.41. Modificación del circuito para la evaluación de las técnicas de deshabilitación de partes inactivas utilizando inhabilitación con CE. ... 108

Figura 4.1. Ejemplo código KISS. 125

Figura 4.2. Resumen del código generado por la herramienta KISS2VHDL para el código KISS2 de la Figura 4.2 126

Figura 4.3. Ahorro de consumo en función de la cantidad de estados. One-Hot respecto de Binario. 129

Figura 4.4. Ahorro de consumo en función de la cantidad de estados. One-Hot respecto de "Out-oriented" 130

Figura 4.5. Consumo por cantidad de estados. Codificación binaria. 131

Figura 4.6. Consumo por cantidad de estados. Codificación One-Hot 131

Figura 4.7. Consumo por cantidad de estados. Codificación Out-oriented 131

Figura 4.8. Consumo por cantidad de estados. Codificación Two-Hot. 132

Figura 4.9. Relación retardo- consumo para las FSMs 132

Figura 4.10. Relación área – consumo en las máquinas de estados. 133

Figura 4.11. Descomposición de máquinas de estados. a. Paralela. b. Cascada. c. General 136

Figura 4.12. Diagrama de estados de una máquina de estados y diagramas de las particiones en la aproximación tradicional 137

Figura 4.13. Diagrama de estados y partición ortogonal de una FSM [She92] 138

Figura 4.14. Arquitectura I para descomposición de máquinas de estado en FPGAs. 140

Figura 4.15. Arquitectura II para descomposición de máquinas de estado en FPGAs. 141

Figura 4.16. Diagrama de transición de estados y grafos de probabilidades. a. FSM de ejemplo. b. Probabilidad condicional para cada arco. c. probabilidad estática de cada estado y la probabilidad total de transición. d. grafo de probabilidades de transición. 144

Figura 4.17. a) Diagrama de transición de estados (State Transition Graph - STG), b) Probabilidad estática y probabilidad total de transición. 145

Figura 4.18. Gráfica de frecuencia, donde se observa el impacto negativo de los registros de bloqueo 151

Figura 4.19. Área de la partición ortogonal respecto de las arquitectura 1 y 2 152

Figura 4.20. Retardo de la partición ortogonal respecto de las arquitectura 1 y 2 152

Figura 4.21. Consumo de la partición ortogonal respecto de las particiones según la arquitectura 1 y 2. 153

Figura 5.1. Reductor módulo *M*. a. Celda elemental del algoritmo con restauración. b. Celda elemental algoritmo sin restauración c. Implementación combinacional algoritmo sin restauración 169

Figura 5.2. Multiplicador modular *shift and add*. a. Celda de cálculo. b. arreglo combinacional 170

Figura 5.3. Celda de cálculo del multiplicador de Montgomery. 171

Figura 5.4. Esquema de la implementación secuencial del algoritmo de sumas y desplazamientos 173

Figura 5.5. Área – retardo – consumo para los multiplicadores modulares 177

Figura 5.6. Celda básica de un sumador, que utiliza la cadena de acarreo en las FPGAs de Xilinx 180

Figura 5.7. Grupo sumador de *s* bits para el carry-skip 181

Figura 5.8. Multiplexores de salto de acarreo (*carry-skip multiplexers*) 182

Figura 5.9. Generación de la condición de propagación del acarreo 183

Figura 5.10. Emplazamiento relativo para un Carry-Skip con N=128, S=16. 184

Figura 5.11. Sumador carry-skip ($n = 4.s$) 185

Figura 5.12. Estructura de los circuitos a medir. 186

Figura 5.13. Estructura de los acumuladores utilizados para medir el consumo 187

Figura 5.14. Divisor con restauración. a. Estructura general. b. Celda básica 194

Figura 5.15. Divisor sin restauración. Estructura general. Celda básica del algortimo. 197

Figura 5.16. Celda de corrección del cociente y resto en el algoritmo sin restauración. 197

Figura 5.17. Circuito de división de naturales binarios con restauración. a. Estructura general. b. Celda elemental de cálculo 199

Figura 5.18. Circuito de división de enteros sin restauración. a. Estructura general. b. Celda elemental de cálculo. c. Celda de corrección para naturales 200

Figura 5.19. Consumo para los divisores con y sin restauración para la secuencia aleatoria (aleatTog) en mW/MHz, ... 203

Figura 5.20. Actividad en un divisor con restauración de 8 por 8 bits 204

Figura 5.21. Actividad en un divisor sin restauración de 8 por 8 bits. 204

Figura 5.22. Algoritmos de división con y sin restauración para números fraccionarios. 208

Figura 5.23. Arquitectura para el algoritmo SRT. a) Paso de división (*division_step*) y circuito combinacional. 209

Figura 5.24. Arquitectura de la celda *srt_r2_division_step* para el algoritmo SRT en base 2. 212

Figura 5.25. Esquema para el divisor SRT base 2. 214

Figura 5.26. Celda para el cálculo de x(n:0) × srn(1:0) 216

Figura 5.27. Celda *Division_step_r4* para el divisor SRT base 4 con resto en complemento a dos....216

Figura 5.28. Divisor SRT base 4 con resto en complemento a la base....217

Figura 5.29. Contenido del *slice* perteneciente a la celda converter para el divisor SRT base 4....218

Figura 5.30. PD-Plot para SRT base 8 con cociente en el rango {-7, -6, -5, ... , -1, 0 , 1, ... , 6, 7}....219

Figura 5.31. PD-Plot para SRT base 16 con cociente en el rango {-15, -14 , ... , -1, 0 , 1, ... , 14, 15}....221

Figura 5.32. Arquitectura del SRT en base 2 con el resto en formato carry-save. 223

Figura 5.33. Celda básica del divisor SRT en base 2 con resto en formato *carry-save*..224

Figura 5.34. Implementación en FPGA de la celda básica del divisor SRT en base 2 con resto en formato *carry-save*. 225

Figura 5.35. Celda de división SRT con resto en *carry-save* (2da versión)....226

Figura 5.36. Arquitectura segmentada de un divisor SRT con LD = 1. 227

Figura 5.37. Arquitectura secuencial del algoritmo de división SRT. 228

Figura 5.38. Retardo de los divisores en función del tamaño de datos en Vitex....232

Figura 5.39. Retardo de divisores en función del tamaño de datos en Virtex II. 232

Figura 5.40. Área de los divisores en Virtex y Virtex II. 233

Figura 5.41. Área y retardo en función de la profundidad lógica para divisores de 32 bits en Virtex II. 235

Figura 5.42. Latencia para los diferentes circuitos secuenciales en Virtex II. 237

Figura 5.43. Relación área × latencia en circuitos secuenciales en Virtex II. 237

Figura 5.44. Retado (en ns) respecto del tamaño de los operandos para diferentes implementaciones de divisores publicadas recientemente y los circuitos descritos en esta sección. 238

Figura 5.45. Relación retardo - latencia - área par las diferentes arquitecturas de divisores fraccionarios. 239

Figura 5.46. Consumo dinámico en mW/MHz para divisores de 16 bits.....240

Figura 5.47. Consumo dinámico en mW/MHz para divisores de 32 bits.....241

Figura 5.48. *Area-Time-Power* para divisores de 32 bits con la secuencia *avg_tog.* 241

Figura 5.49. Consumo dinámico (mW/MHz) con respecto a la profundidad lógica para los divisores SRT base-2 con resto en complemento a dos. 242

Figura 5.50. Consumo dinámico (mW/MHz) con respecto a la profundidad lógica para los diferentes divisores segmentados y la secuencia de prueba *avg_tog.* 242

Figura 5.51. Consumo, retardo y área normalizados respecto de la profundidad lógica para el divisor SRT base-2 con resto en complemento a la base. 243

Figura 5.52. Consumo de energía tanto de sincronización como de la ruta de datos para los divisores secuenciales de 32 bits. 244

Figura 5.53. Área, retardo y consumo (ATP) para diferentes algoritmos división y arquitecturas secuenciales, combinacionales y segmentadas 245

Figura A.1. Esquema de la placa de prueba de la familia XC4000. 274

Figura A.2. Esquema de conexión del arreglo experimental en XC4K 275

Figura A.3. Fotografía del arreglo experimental para XC4K.....275

Figura A.4. Esquema de un generador de patrones en las XC3K 278

Figura B.1. Detalle de la placa de prototipado AFX.....281

Figura B.2. Fotografía de la tarjeta AFX.....282

Figura B.3. Detalle de la interfaz con el puerto paralelo de la tarjeta AFX.....283

Figura B.4. Fotografía del arreglo experimental placa AFX 285

Figura C.1. Diagrama de bloques de la placa de prototipado para Virtex II [Xil03] ..290

Figura C.2. Detalle de la placa de prototipado para Virtex II [Xil03]. 291

Figura C.4. Fotografía del arreglo experimental placa Virtex II.....293

Figura D.1. Un primer ejemplo de descripción KISS2.....298

Figura D.2. Ejemplo de grafo de transición de estados y su correspondiente especificación en KISS2 299

Figura E.1. Declaración de la parte privada de las clases *state* y *transition*. 306

Figura E.2. Código para la partición de máquinas de estados. 308

Figura E.3. Diagrama de transición de estados de la máquina de estados DK27. 309

Figura E.4. a) Código KISS2 de entrada. b) Información de la herramienta con las probabilidades calculadas. 310

Figura E.5. Información de las particiones generadas. 310

Figura E.6. Partición generada para el circuito *dk27* 311

Figura F.1. Aspecto de la interfaz gráfica Xpower. 319

Figura F.2. Opciones en la herramienta de línea de comando Xpwr. 321

Figura F.3. Script de implementación de un diseño. 322

Figura F.4. Script de simulación e invocación a xpwr. 322

Figura F.5. Script para el simulador. 323

Lista de Tablas

Tabla 3.1. Consumo estático para dispositivos XC4K, Virtex y Virtex II medidos. 52

Tabla 3.2. Relación consumo estático respecto del dinámico en XC4K, Virtex y Virtex II. 53

Tabla 3.3. Ejemplos de la relación del consumo de sincronización respecto del de la ruta de datos en Virtex. 56

Tabla 3.4. Principales características de los circuitos de prueba. Relación Velocidad-Consumo 62

Tabla 3.5. Principales características de los circuitos de prueba. 74

Tabla 3.6. Descripción de los vectores de prueba 75

Tabla 3.7. Actividad de las entradas, salidas y totales para la operación A×B y B×A con vectores de máximo *toggle-8* 77

Tabla 3.8. Principales características de los circuitos de prueba. Familia Virtex 80

Tabla 3.9. Diferencia de consumo A×B vs. B×A en multiplicadores de 32 bits. 81

Tabla 3.10. Diferencia de consumo A×B vs. B×A en multiplicadores de 16 bits. 81

Tabla 3.11. Relación medidas – estimaciones para multiplicadores de 32 bits y secuencia MaxTog. 83

Tabla 3.12. Relación medidas – estimaciones para multiplicadores de 32 bits y secuencia AvgTog. 83

Tabla 3.13. Relación medidas – estimaciones para multiplicadores de 16 bits y secuencia MaxTog. 83

Tabla 3.14. Relación medidas – estimaciones para multiplicadores de 16 bits y secuencia AvgTog. 84

Tabla 3.15. Principales características de las diferentes topologías de prueba XC4K. .88

Tabla 3.16. Principales características de los circuitos de prueba para Virtex 90

Tabla 3.17. Consumo medido respecto del estimado para multiplicadores segmentados....93

Tabla 3.18. Principales características de los circuitos de prueba para Virtex II....94

Tabla 3.19. Diferencia de consumo dinámico dependiendo del tipo de registros en Virtex. Consumo dinámico a 5 MHz....100

Tabla 3.20. Diferencia de consumo dinámico dependiendo del tipo de registros en Virtex II. Consumo dinámico a 10 MHz....101

Tabla 4.1. Circuitos de prueba originales y minimizados en cantidad de estados....127

Tabla 4.2. Área-Velocidad y Consumo para el conjunto de circuitos de prueba....128

Tabla 4.3. Datos originales de las máquinas de estados, cantidad de entradas, salidas, estados y arcos. Además información de la partición (probabilidad y número de arcos entre particiones)....147

Tabla 4.4. Consumo de potencia expresado en mW/MHz para la partición de máquinas de estado....149

Tabla 4.5. Área expresada en CLBs para la partición de máquinas de estado....150

Tabla 4.6. Circuitos donde no se logra mejora en el consumo debido a la alta probabilidad de transición entre submáquinas de estados....151

Tabla 5.1. Número de CLBs y Retardo Máximo (ns) para los multiplicadores modulares secuenciales....172

Tabla 5.2. Número de CLBs y Frecuencia máxima (MHz) para multiplicadores modulares secuenciales....175

Tabla 5.3. Área, retardo y consumo (Area-Time-Energy) de los multiplicadores modulares combinacionales....176

Tabla 5.4. Área, retardo y consumo de los multiplicadores modulares secuenciales..176

Tabla 5.5. Resultados de la implementación de circuitos carry skip: Retardos en ns. 186

Tabla 5.6. Resultados de la implementación de circuitos carry skip: Penalidad en área. 187

Tabla 5.7. Consumo sumadores Carry-Skip....188

Tabla 5.8. Área y retardo para la división con y sin restauración.....201

Tabla 5.9. Consumo dinámico para los divisores de números enteros expresados en mW/MHz. 202

Tabla 5.10. Actividad de señales y lógica para los algoritmos divisores de enteros de 8 bits. Los valores expresan frecuencias de las señales en MHz.....205

Tabla 5.11. Operaciones a realizar por el SRT en base 2 212

Tabla 5.12. Función lógica implementada en la celda S_R_Nada del algoritmo SRT base 2.....213

Tabla 5.13. Selección del dígito Qi en el SRT en base 4. 215

Tabla 5.14. Tabla Qsel para el SRT base 2 con *carry-save*.....224

Tabla 5.15. Resultados para implementaciones combinacionales en la familia Virtex 230

Tabla 5.16. Resultados para implementaciones combinacionales en la familia Virtex II. 231

Tabla 5.17. Profundidad Lógica (LD), ciclos necesarios, área en *slices*, cantidad de registros (FF y SRL) y máximo ancho de banda en MHz para diferentes arquitecturas en Virtex. 234

Tabla 5.18. Profundidad Lógica (LD), ciclos necesarios, área en *slices*, cantidad de registros y máximo ancho de banda en MHz para diferentes arquitecturas en Virtex II.....234

Tabla 5.19. Resultados implementaciones iterativas en Virtex.....236

Tabla 5.20. Resultados implementaciones iterativas en Virtex II. 236

Tabla A.1. Conexiones del analizador en la placas de prueba. 272

Tabla A.2. Conexiones del cable de programación y botones.....273

Tabla A.3. Recursos de los dispositivos de la serie XC4K utilizados en las mediciones 273

Tabla A.4: Componentes del consumo y forma de medición en XC4K.....276

Tabla B.1. Conexiones del generador de patrones a la FPGA. 284

Tabla B.2. Conexiones del generador de la FPGA al analizador lógico 284

Tabla B.3: Componentes del consumo y forma de medición en Virtex.....286

Tabla C.1. Componentes del consumo y forma de medición en Virtex II 294

Lista de Acrónimos

ASIC	*Application Specific Integrated Circuit.* Circuito de aplicación especifica
ATP	*Area-Time-Power.* Relación área - retardo - consumo
C2	Sistema de representación de enteros de complemento a dos
CAD	*Computer Aided Design.* Diseño Asistido por Ordenador
CD	Cero Desplazado. Sistema de representación de enteros.
CDSP	*Custom Digital Signal Processor.* Procesador digital de señal a medida.
CLA	*Carry Look Ahead Adder.* Sumador de acarreo anticipado
CLB	*Configurable Logic Block.* Bloque configurable de lógica. Celda elemental en las FPGAs.
CMOS	*Complementary Metal Oxide Semiconductor.*
CPLD	*Complex PLD.*
CSA	*Carry-Save Adder.* Sumador con ahorro de propagación de acarreo
DCM	*Digital Clock Manager.* Administrador digital de reloj. Un elemento de diseño que provee múltiples funciones sobre el reloj.
DCT	*Discrete Cosine Transform.* Transformada discreta del coseno.
DLL	*Delay Locked Loop.* Circuito para realizar manejos del reloj.
DSP	*Digital Signal Processor.* Procesador digital de señal
EDA	*Electronic Design Automation.* Nombre genérico para especificar herramientas para automatizar el diseño electrónico.
EDIF	*Electronic Data Interchange Format.* Un formato de fichero estándar industrial para especificar *netlists.*
EEPROM	*Electrically Erasable Programmable Read Only Memory.* Memoria de solo lectura programable y borrable electrónicamente

EPROM	*Erasable Programmable Read Only Memory.* Memoria de solo lectura programable y borrable.
FA	*Full Adder.* Sumador completo
fc	Frecuencia máxima de funcionamiento o de reloj
FF	*Flip-Flop.* Báscula.
FFT	*Fast Fourier Transform.* Transformada rápida de Fourier.
FIFO	*First In First Out.* Primero entra primero sale.
FIR	*Finite Impulse Response.* Aplicado a Filtros digitales
FPGA	*Field Programmable Gate Array.*
FSM	*Finite State Machine.* Máquina de Estados Finitos
GF	*Galois Field.* Cuerpo de Galois (cuerpo finito)
HDL	*Hardware Description Language.* Lenguaje de descripción de Hardware
I/O	*Input / Output.* Entrada/Salida.
IDE	*Integrated Development Environment*
IOB	*Input Output Block.* Bloques de entrada salida en FPGA de Xilinx.
ISE	*Integrated Software Environment.* Nombre del entorno de desarrollo para FPGAs y CPLDs de Xilinx
JHDL	*Java Hardware description Language.* Lenguaje de descripción de Hardware basado en JAVA
LD	*Logic Depth.* Profundidad lógica.
LSI	*Large-Scale Integration.* Alta escala de integración
LUT	*Look-Up Table.* Tabla de consulta. Implementa funciones booleanas.
MCNC	*Microelectronics Center of North Carolina.* Centro de microelectrónica de Carolina del Norte. Circuitos electrónicos de referencia.
MSB	*Most Significant Bit.* Bit más significativo.
MSD	*Most Significant Digit.* Dígito más significativo

NCD	*Native Circuit Description.* Fichero que representa la representación física de un dispositivo en el entorno de Xilinx.
NGD	*Native Generic Database.* Fichero que describe el diseño lógico en base a primitivas de Xilinx.
NR	*Non-Restoring.* Algoritmo de división sin restauración.
NRE	*Non-Recurring Engineering.* Costes no recurrentes de ingeniería. Utilizado para nombrar los costos fijos en el desarrollo.
OH	*One Hot.* Codificación de máquinas de estados.
PAR	*Place and route tool.* Programa para el mapeo, emplazamiento y rutado en las FPGAs de Xilinx.
PCB	*Printed Circuit Board.* Placa de circuito impreso
PCF	*Physical Constraints File.* Fichero de restricciones en el entorno Xilinx.
PD-Plot	*Partial-Remainder and Divisor Plot.* Gráfica del resto parcial respecto del divisor (selección de cocientes en algoritmos de división)
PLD	*Programmable Logic Device.* Dispositivo Lógico Programable
PSC	*Parallel Serie Converter.* Conversor paralelo/serie
r2	Coeficiente de determinación en el análisis de regresión lineal
RCA	*Ripple Carry Adder.* Sumador de acarreo propagado
RISC	*Reduced Instruction Set Computer.* Ordenador de conjunto reducido de instrucciones.
ROM	*Read Only Memory.* Memoria de solo lectura
RPM	*Relationally Placed Macro.* Define relaciones espaciales entre las primitivas de un diseño en Xilinx.
RSA	*Rivest-Shamir-Adleman.* Sistema criptográfico con clave pública
RTL	*Register Transference Level.* Nivel de descripción de hardware a nivel de transferencia entre registros.
S/P	Serie/Paralelo.

SDA	*Serial digit Arithmetic.* Aritmética de dígitos en serie
SDF	*Standard Delay Format.* Estandar industrial para especificar información de tiempos, usado generalmente en la simulación.
SoC	*System on Chip.* Sistema en un chip
SPC	*Serial Parallel Converter.* Conversor serie/paralelo
SRAM	*Static Random Access Memory.*
SRL	*Shif Register in LUT.* Registros de desplazamiento en LUT.
SRT	*Sweeney - Robertson - Tocher.* Algoritmo de división.
STG	*State Transition Graph.* Grafo de transición de estados
STT	*State Transition Table.* Tabla de transición de estados
SVA	Signo Valor Absoluto. Sistema de representación de enteros, también dominado de signo y magnitud
TH	*Two Hot.* Codificación de máquinas de estados.
UCF	*User Constraints File.* Fichero de restricciones en el entorno Xilinx.
ULP	*Unit in the Least significant Position.* Unidad en las posiciones menos significativas
USB	*Universal Serial Bus.*
UUT	*Unit Under Test.* Unidad bajo prueba
VCD	*Value Change Dump.* Volcado del cambio de valores. Fichero para automatizar la anotación del movimiento de las señales.
VHDL	VHSIC *(Very High Speed Integrated Circuit) Hardware Description Language.* Lenguaje de Descripción de Hardware.
VLSI	*Very Large-Scale Integration.* Circuitos de muy alta escala de integración
XST	*Xilinx Synthesis Technology.* Herramemienta de síntesis HDL de Xilinx.
ZOH	*Zero One Hot.* Codificación de máquinas de estados.

Anglicismos, barbarismos, neologismos y otros defectos

Indudablemente el inglés es el idioma que mejor ha logrado adaptarse a la inevitable ampliación de vocabulario generada por la revolución tecnológica de los últimos años. En la actualidad, difícilmente puede pretender un científico difundir universalmente su trabajo al margen de los medios escritos en inglés y resulta obvio que la práctica totalidad de las revistas científicas más acreditadas se escriben en este idioma.

El liderazgo indiscutible de la lengua Inglesa se basa en que una parte significativa de los progresos científicos y técnicos modernos se han originado y desarrollado fundamentalmente en áreas geográficas anglófonas, y que consecuentemente un notable número de las publicaciones científicas utilizan esta lengua como medio común de propagación e intercambio de información. Su innegable simplicidad fonética y gramatical contribuye en este proceso.

Si bien este fenómeno tiene, por una parte, el valor de facilitar la comunicación y el intercambio universal de información (quizás una versión moderna del utópico esperanto), no cabe duda que, por otro lado, su progresivo asentamiento haya permitido la adulteración de otros idiomas con una sólida tradición gramatical.

Aunque ninguna rama de la ciencia se libra de dicha contaminación, es en el lenguaje informático y de telecomunicaciones donde pueden detectarse nítidamente varios de estos barbarismos.

Esta tesis no esta exenta de estos vicios idiomáticos, los que en ocasiones son casi inevitables. A continuación se listan algunos de estos términos de común utilización en el entorno y área de trabajo del autor y que han sido utilizados durante la redacción del presente documento.

benchmark m. Castellanización del mismo término inglés. Su significado y correcta traducción es la de banco de prueba. Utilizado en el contexto de prueba sistematizada sobre código de descripción de hardware.

búfer m. Castellanización del término *buffer* con múltiples acepciones: Amplificador y restaurador de niveles (electrónica digital), memoria intermedia (programación), etc...

bus m. Castellanización del mismo término inglés. Su significado según el diccionario difiere totalmente con el empleado en electrónica: colección de señales.

core m. Castellanización del mismo término inglés. En ocasiones se utiliza con el significado de núcleo (parte interna del circuito sin la periferia), aunque también se utiliza en la jerga de las FPGAs como circuito prediseñado.

datapath m. Castellanización del mismo término inglés. Se refiere a la ruta de datos de un diseño electronico.

deshabilitación m. Castellanización del término *disable*. Supresión, interrupción o inhabilitación son términos más correctos.

embebido. Castellanización del término *embedded*. El autor prefiere utilizar "dedicado" o "empotrado" aunque el término anterior es ampliamente difundido.

fanin m. Castellanización del mismo término inglés. Expresa el concepto: cantidad de entradas de un circuito lógico.

fanout m. Castellanización del mismo término inglés. Expresa el concepto: conexiones o carga de una señal.

flip-flop m. Castellanización del mismo término inglés. Más utilizado que la correcta palabra castellana biestable.

gated clock m. Castellanización del mismo término inglés. Técnica de diseño electrónico de "bloqueo de reloj".

glitch m. Castellanización del mismo término inglés. Muy empleado al no existir un término español tan compacto. Se refiere a las transiciones espurias.

hardware m. Castellanización del mismo término inglés.

instancia m. Castellanización del término *instance*. La acepción utilizada no coincide con la definición del diccionario. En diseño electrónico ó en programación orientada a objetos: elemento correspondiente a una determinado tipo o clase.

instanciar v. t. Castellanización del verbo *to instantiate*. Se utiliza para indicar la creación de un nuevo elemento de un determinado tipo o clase. Utilizado tanto en diseño software como de hardware.

latch m. Castellanización del mismo término inglés. Más utilizado que la correcta palabra castellana cerrojo.

layout m. Castellanización del mismo término inglés. Se utiliza con el significado de: disposición de los elementos que componen un circuito.

macro. Castellanización del término inglés. Se utiliza como circuito prediseñado, ó bien como conjunto de instrucciones y acciones guardadas en un fichero. También se utiliza con el significado castellano de prefijo que significa "grande".

mapear v. t. Castellanización del verbo *to map*. En el contexto de esta tesis se podría traducir como asociar recursos a elementos lógicos; no sería correcto emplear sólo el verbo asociar, porque no tiene todos los matices de *to map*.

pad m. Castellanización del mismo término inglés. No existe una palabra en español que tenga exactamente los mismos matices, pues no es la pata de un circuito integrado, sino el área donde ésta se conecta

pin m. Castellanización del mismo término inglés. En español se utiliza el término pata de un circuito integrado, pero es esta palabra es del mismo modo muy utilizada.

power management m. Castellanización del término inglés. Se refiere al grupo de técnicas para el manejo del consumo de energía

reconfigurable / reconfiguración adj. Adaptación del término *reconfigurable / reconfiguration*. Castellanización con base en la semántica de reconfigurar.

reconfigurar v. t. Adaptación del término *reconfigure* de raíz latina. Utilizado para indicar la recarga del contenido de un circuito reprogramable.

reset m. Castellanización del término *reset*. La palabra equivalente en español, reiniciación, es mucho menos utiliza.

retiming m. Castellanización del mismo término inglés. Se refiere a una técnica específica para el aumento de velocidad en circuitos digitales. Una posible traducción es retemporización, utilizando la temporización con el significado aquí descrito.

rutar v. t. Castellanización del verbo *to route*. Las acepciones que aparecen en el diccionario son completamente distintas al uso que se le da en esta tesis, crear un camino para establecer las conexiones de una señal.

script m. Aunque existe el término castellano "guión", este término es más utilizado en el área de la ingeniería.

software m. Castellanización del mismo término inglés.

temporización f. Castellanización del término *temporization*. El significado que da el diccionario a temporizar no se corresponde con el concepto para el que se emplea en el diseño electrónico: fijar la evolución en el tiempo de algo.

topología f. Si bien existe el término castellano, el significado de la real academia no soporta aun una de las acepciones del término inglés *topology*: Configuración formada por conexiones entre dispositivos

triestado adj. Castellanización del término *triestate*.

Capítulo 1:

Introducción y Motivación

El objetivo general de esta tesis es dar soluciones en el diseño de bajo consumo en FPGAs: conocer qué circuitos poseen mejores características de consumo, respetando propiedades de área y velocidad. Sin embargo, ésta es una meta demasiado ambiciosa y alcanzarla con éxito requeriría la evaluación de multitud de operandos, variables, arquitecturas y algoritmos. Esta tesis es un aporte más a dicho objetivo y se basa en un fuerte trabajo experimental donde más de 600 circuitos fueron construidos y medidos.

La utilización de FPGAs (*Field Programmable Gate Arrays*) ha aumentando rápidamente debido a la posibilidad de realizar prototipos con tiempos reducidos y con bajos costes de desarrollo (NRE - *Non-Recurring Engineering costs*). El consumo de potencia y la disipación desde hace años son un factor importante en el diseño de VLSI y existen múltiples herramientas y metodologías. Sin embargo, en el área de la lógica reprogramable, los dispositivos, herramientas y metodologías están todavía muy por debajo de los requerimientos del mercado.

1.1 Reducción y estimación del consumo

La utilización de una metodología de estimación y control del consumo de circuitos integrados resulta indispensable. En la actualidad, el calor generado por un circuito integrado, a menudo sobrepasa los límites de disipación de los encapsulados, siendo imperiosa la utilización de refrigeración artificial. Actualmente, el consumo de potencia de algunos chips, fundamentalmente microprocesadores, pueden superar los 50 vatios, con una tendencia de al menos duplicarse en los próximos años

[Jew00][Sia97][Itr03]. Valga como ejemplo el consumo Intel Pentium 4 a 2,4 GHz (Northwood) con tecnología de 130 nm puede superar los 57 W [Act02], en tanto un AMD XP 2200+ con tecnología de 180 nm supera los 67 W [Kai02].

Las ventajas de la reducción de consumo sobrepasan el campo de aplicación natural, relacionado con la electrónica portátil (ordenadores, telefonía, sistemas remotos de adquisición, etc.). En primer lugar, tiene un importante impacto económico, pues permite reemplazar encapsulados cerámicos por plásticos cuyo costo es al menos un 25% menor, y a la vez simplifica o elimina la necesidad de elementos de refrigeración, tales como ventiladores, disipadores o sensores de temperatura. [Kah02] estima que el costo del encapsulado se incrementa en 1$ por vatio de consumo.

Por otro lado, teniendo en cuenta que fallos de los circuitos integrados crecen exponencialmente con la temperatura, la reducción del consumo aumenta la fiabilidad y vida del producto. Según [Sma94][Ped96] cada 10 °C aproximadamente se dobla la tasa de fallos. Además la vida del producto se ve afectada por la temperatura, Intel calcula que el aumento de entre 10-15 °C puede afectar la vida de los circuitos en un factor de dos [Vis00]. Finalmente un valor elevado del consumo, se refleja en picos de corrientes síncronos con el reloj, que pueden afectar al funcionamiento del circuito hasta proyectar su influencia sobre aspectos aparentemente independientes como la complejidad del PCB o la sincronización [Boe95].

Otro aspecto importante para evitar un excesivo consumo, aún en las aplicaciones donde no existen restricciones en ese sentido, es que la velocidad de un circuito CMOS decrece hasta en un factor del 0,35% por °C [Ped96].

1.2 Marco tecnológico

Se utilizarán circuitos programables (*Programmable Logical Devices* - PLDs), más específicamente FPGAs (*Field Programmable Gate Arrays*) de Xilinx. La elección de esta tecnología VLSI está justificada en su bajo coste y su capacidad de reutilización. El mercado de circuitos reprogramables ha crecido muy velozmente, compitiendo en familias de aplicaciones que tradicionalmente eran exclusivos de tecnologías ASIC y DSPs.

El diseño en FPGA exige una metodología propia, en la mayoría de los casos diferente a la utilizada para el diseño de ASICs, que está condicionada por su estructura y

recursos disponibles. Muchos de los principios del diseño de ASICs no se cumplen en el diseño en FPGAs debido, principalmente, a los grandes retardos provocados por el rutado, la división en LUTs de los circuitos y a los recursos fijos de los dispositivos. Por todo esto, la viabilidad de algunos circuitos, métodos de diseño y axiomas clásicos deben reconsiderarse en función de las características de la FPGA elegida.

La justificación del fabricante elegido se basa en varios hechos: Xilinx [Xil04] es el "inventor"[1] de las FPGAs. Fundada en 1984 es líder del mercado y su arquitectura basada en tablas de consulta (*look-up tables*) de cuatro entradas es la más difundida entre otros fabricantes. Xilinx actualmente acapara en torno al 50 % del mercado de la lógica programable [Par04], lo que brinda gran difusión y aplicación a los resultados. Por último, y no menos importante, en la metodología (y "filosofía") de diseño de este fabricante se tiene acceso a todos los niveles de implementación y se puede controlar cada detalle de la implementación física del circuito, elemento indispensable para la evaluación de diferentes técnicas y alternativas.

A fin de disponer de una mayor generalidad de los resultados, se han realizado pruebas sobre diferentes dispositivos que cubren un amplio espectro de tiempo. Se han utilizado dispositivos de las serie XC4K, la familia que más tiempo ha estado en el mercado, introducida en 1991 (hoy sigue en fabricación la versión económica de 3,3V, Spartan XL); Virtex en fabricación desde 1998 y Virtex II cuyos primeros dispositivos datan de fines de 2001.

El problema del consumo en FPGAs

Las FPGAs de hoy en día poseen un altísimo nivel de integración, utilizando la tecnología de fabricación más avanzada del momento. La cantidad de transistores integrados en un único chip está al nivel de los microprocesadores más avanzados. De este modo los dispositivos programables siguen las tendencias de crecimiento de los segmentos punteros en la tecnología VLSI, con lo cual la problemática del consumo no es ni mucho menos ajena a esta tecnología.

[1] La idea original fue desarrollada a fines de los años 60 y patentada por Sven Wahlstrom [Wah67], aunque recien 1984 Xilinx (entonces una compañía *startup*, y como hasta hoy en día *fables*) fabricó y comercializó esta idea.

El alto nivel de reprogramabilidad y versatilidad, genera a su vez una gran incertidumbre respecto del consumo. Las diferentes formas de implementar una misma funcionalidad generan variaciones en el consumo que pueden superar fácilmente un orden de magnitud. He aquí uno de los motivos fundamentales para establecer criterios para el diseño de bajo consumo.

Según [Sha02] el consumo medio en un diseño promedio implementado en un dispositivo Virtex II es alrededor de 1.5 μW/MHz/Slice, con lo que un diseño modesto de unos 8000 *slices* (2000 CLBs) funcionando a 100 MHz puede consumir 1,2 Vatios. No obstante esta métrica, conservadora en opinión el autor de esta tesis, no tiene en cuenta detalles de implementación como la profundidad lógica o la generación de movimientos espurios (*glitches*). Por ejemplo, un divisor *non-restoring* de 32 bits totalmente combinacional, ocupando 576 *slices* en un dispositivo Virtex Consume alrededor de 610 μW/MHz/Slice. Lo que significa que dividir a 5 MHz puede consumir más de 1,75 W. Hay que tener en cuenta que el divisor antes mencionado, con una implementación secuencial cuidadosa, puede reducir el consumo en un factor de 20.

1.3 Selección de técnicas de reducción de consumo

El consumo de un circuito integrado es función de la arquitectura, tecnología, interconexión, secuencia de datos de entrada y estados por los que evoluciona. Al igual que el área o la velocidad, su estudio, estimación y control pueden realizarse en cualquiera de los niveles de la jerarquía de diseño.

En este trabajo, se desarrollarán técnicas de reducción de consumo en los niveles arquitecturales, funcionales y de implementación, que son los que mejor se adaptan al marco tecnológico elegido. Para cada uno de los estratos anteriores, existirán dos categorías de beneficiario de los resultados de la investigación: por un lado, el fabricante del circuito integrado, y por otro, los usuarios finales del mismo a través del desarrollo de heurísticas para la reducción del consumo.

La selección de las técnicas parte de un primer estudio bibliográfico y un refinamiento posterior para adecuarlo al marco tecnológico. Mucho se ha publicado sobre bajo consumo en tecnologías VLSI, los principales aportes a nivel arquitectural y topológico tienen alrededor de una decada de publicación [Dev95] [Sch95] [Liu94]

[Su94] [Pow91] [Su96], con una producción de conocimiento sostenida en los últimos años [Ben99][Alc00][Ben01][Sou02][Sun03][Tsu03][Jac04][Yin04][Kiy04]. Por otra parte, en FPGAs recien en el último lustro se percibe un incremento significativo de las publicaciones.

Una gran parte de las técnicas descritas en las publicaciones son a nivel teórico con resultados obtenidos por simulación. Esta tesis tiene entre otros aspectos que la diferencian con previas contribuciones, la utilización de medidas empíricas sobre los circuitos.

1.4 Objetivos de ésta tesis

Como resumen de lo anterior, el objetivo de ésta tesis es desarrollar una serie de técnicas y heurísticas que permitan reducir el consumo en circuitos programables. Ésta tesis tiene un fuerte carácter experimental y todas las técnicas son validadas en dispositivos concretos alejándose de otros enfoques puramente teóricos o a nivel de simulación.

La tesis se desarrolla sobre dispositivos del fabricante de circuitos programables Xilinx. Sin embargo los resultados son extrapolables a otros fabricantes y a dispositivos futuros. La arquitectura basada en tablas de *look-up* de 4 entradas es la más difundida entre los fabricantes de este tipo de circuitos, y además se han realizado pruebas con circuitos que tienen más de 10 años de diferencia en la salida al mercado obteniéndose conclusiones similares. En resumen, los principales objetivos de este trabajo pueden dividirse en cinco grandes grupos:

a. Estudio de las técnicas de reducción de consumo en FPGAs. A falta de literatura que resuma esta problemática, el primer punto es el análisis de los trabajos en otras tecnologías VLSI y su adaptabilidad al marco tecnológico. Esta tarea se ha de complementar con los aportes de los últimos años en el consumo de FPGAs.

b. Utilización de las herramientas de desarrollo actuales para la reducción de consumo. Estudio del uso de herramientas de desarrollo como métricas indirectas para estimar o reducir el consumo. Este tema incluye explorar las relaciones de la velocidad y el área con el consumo, así como otras técnicas y opciones de diseño que típicamente se asocian y utilizan para reducir el área o aumentar la velocidad.

c. Estudio de máquinas de estados. Las máquinas de estados finitos son omnipresentes en cualquier diseño digital, ya sea para modelar un problema concreto, ó como control de una ruta de datos. Se estudiarán alternativas de reducción de consumo, ya sea a través de la codificación de estados o bien utilizando técnicas de deshabilitación.

d. Estudio de bloques aritméticos. Estudio de diferentes bloques aritméticos a fin de encontrar patrones de consumo respecto a la implementación de diferentes familias de algoritmos. La implementación de bloques aritméticos incluirá el estudio e influencia de diferentes alternativas arquitecturales y algorítmicas.

1.5 Organización y lectura de ésta tesis

Esta tesis está organizada en seis capítulos y seis apéndices. Este primer capítulo presenta los objetivos y alcances de esta memoria. La organización de cada capítulo se estructura en secciones que comprenden un tema autocontenido.

El capítulo 2 repasa los principales conceptos en el estudio del bajo consumo orientado a FPGAs. Se agrega además información actualizada de las tendencias en el consumo, así como el desglose del consumo en FPGAs. Este capítulo es de utilidad para quien intenta interiorizarse en las ideas necesarias para realizar diseño de bajo consumo.

En el capítulo 3 se examinan varios experimentos llevados a cabo sobre distintos modelos de FPGAs con el objeto de determinar relaciones generales en el consumo. Comienza con experimentos elementales sobre la relación de la tensión y la frecuencia en el consumo, para proseguir con la influencia del consumo estático y de sincronización. Más adelante se examinan la relación velocidad-consumo, la conmutatividad de datos, el efecto de la segmentación (*pipeline*) y el uso de registros en general. Finalmente se analizan y cuantifican las alternativas para la inhabilitación de partes de un circuito.

El capítulo 4 aborda el tema de la reducción de consumo en máquinas de estados finitos en FPGAs, por un lado se analiza el efecto de la codificación en el consumo y por otro se propone una arquitectura de partición de máquinas de estado para la reducción del consumo.

En el capítulo 5 se describen experimentos sobre diferentes bloques aritméticos. Por un lado se examinan las opciones para la multiplicación modular, operación central en los cálculos criptográficos, más tarde se presentan experimentos sobre la adición de alta velocidad, en particular el algoritmo de *carry-skip*, a continuación, se examinan los algoritmos y arquitecturas para la división entera. Por último, se analizan las alternativas para la división de números fraccionarios, operación llevada a cabo sobre la mantisa en las operaciones de división de números en coma flotante.

Finalmente, en el capítulo 6 se recopilan las principales conclusiones y aportes generados, las futuras líneas de investigación que se desprenden de ésta tesis, así como una enumeración de las publicaciones generadas a raíz de este trabajo. Adicionalmente, a modo de resumen, se seleccionan las reglas de oro en el diseño de bajo consumo en FPGAs desde el punto de vista del usuario final.

Los apéndices incluyen información complementaria al contenido principal. Los apéndices *A*, *B* y *C* describen los arreglos experimentales utilizados durante las mediciones. El primero describe el arreglo para la familia XC4C, el segundo lo hace para la familia Virtex, en tanto que el siguiente lo hace para la familia Virtex II.

Los apéndices *D* y *E* describen las herramientas de generación de código desarrolladas para los experimentos del capítulo 4. La primera se trata de un traductor de formato de especificación de máquinas de estado KISS a VHDL (*Kiss2VHDL*), en tanto que la segunda herramienta realiza la partición de máquina de estado según un proceso probabilístico generando código VHDL (*part_FSM*).

Finalmente, el apéndice *F* describe y analiza la herramienta de estimación de consumo provista por Xilinx denominada XPOWER [Xpo04]. Adicionalmente, se brindan una serie de *scripts* para automatizar la medición del consumo.

Por facilidad y comodidad, cada sección incluye un resumen de resultados, lo mismo se lleva acabo al final de cada capítulo. Las referencias empleadas en cada capítulo se resumen al final de cada uno de ellos.

1.6 Referencias del capítulo

[Act02] Active Hardware Editor, "Intel 2.4 ghz Pentium 4 Northwood", *Active Hardware electronic publication.* Available at http://www.active-hardware.com/ April, 2002.

[Alc00] J. Alcalde, J. Rius and J. Figueras, "Experimental techniques to measure current, power and energy in CMOS integrated circuits", *Proceedings of XV Conference on Design of Circuit and Integrated Systems (DCIS'2000).* Montpellier, France, November 2000.

[Ben01] Luca Benini, Giovanni De Micheli, Enrico Macii, "Designing Low-Power Circuits: Practical Recipes", IEEE Circuit and System Magazine, Vol. 1, No. 1, pp. 6-25. First Quarter, 2001.

[Ben99] L. Benini, G. de Micheli, A. Macii, E. Macii, M. Poncino, R. Scarsi, "Glitch Power Minimization by Gate Freezing", Design, Automation and Test in Europe (DATE '99) Munich, Germany, March , 1999.

[Boe95] E. Boemo, G. Gonzalez de Rivera, S.Lopez-Buedo and J. Meneses, "Some Notes on Power Management on FPGAs", *Lecture Notes in Computer Science (LNCS)*, No.975, pp.149-157. Berlin: Springer-Verlag 1995.

[Dev95] S. Devadas y S. Malik, "A Survey of Optimization Technique Targeting Low Power VLSI Circuits", *32nd Design Automation Conference (DAC'95)*, pp.242-247, 1995.

[Itr03] ITRS Technology Working Groups. "International Technology Roadmap for Semiconductors 2003 Edition: Executive Summary". *Jointly Sponsored by European SIA, Japan EITLA, Korea SIA, Taiwan SIA, and Semiconductor Industry Association (SIA)*, http://public.itrs.net/, 2003.

[Jac04] Hans M. Jacobson, "Improved ClockGating through Transparent Pipelining", *International Symposium on Low Power Electronics and Design (ISLPED'04)*, Newport Beach, California, USA. August 9–11, 2004.

[Jew00] Jim Jewett, "The International Technology Roadmap for Semiconductors (ESH THRUST)" *Intel Corporation Arlington*, VA; April 2000.

[Kah02] Andrew B. Kahng, "The significance of packaging", *IEEE Design & Test of Computers*, November - December 2002.

[Kai02] Kai Schmerer, "AMD's Thoroughbred leaves the starting gate". http://www.reviews.zdnet.co.uk/. June, 2002.

[Kiy04] Kiyoo Itoh, Kenichi Osada, Takayuki Kawahara, "Low-Voltage Embedded RAMs – Current Status and Future Trends", Lecture Notes in Computer Science, Volume 3254, pp. 3 – 15. Springer-Verlag Heidelberg. Sept, 2004.

[Liu94] D. Liu y C. Svensson, "Power Consumption Estimation in CMOS VLSI Circuits". *IEEE Journal of Solid-State Circuits*, Vol.29, N°6, pp. 663-670, Jun 1994.

[Par04] Karen Parnell, Roger Bryner, "Comparing and Contrasting FPGA and Microprocessor System Design and Development", *Xilinx White paper WP213* (v1.1). July 2004.

[Pow91] S. Powell y P. Chau, "A Model for Estimating Power Dissipation in a Class of DSP VLSI Chips", *IEEE Trans. on Circuit and Systems*, VOL. 38, N°6, pp.646-650, Jun 1991.

[Sch95] T. Schneider, C. Piguet y V. von Kaenel, "Low-voltage / Low-power Parallelized Logic Modules", *Proc. PATMOS'95, Fifth Int. Workshop*, pp.148-160. Oldenburg, Oct 1995.

[Sha02] Li Shang, Alireza Kaviani, Kusuma Bathala, "Dynamic Power Consumption in Virtex™-II FPGA Family", *Proceedings of Tenth ACM International Symposium on Field-Programmable Gate Arrays (FPGA'02)*, Monterey, California, USA. February 2002.

[Sia97] SIA Semiconductor Industry Association, "The National Technologies Roadmap for semiconductors: Technologies Needs". SIA home page: http://www.semichips.org/, 1997.

[Sma94] C. Small, "Shrinking devices put the squeeze on system packaging". *EDN (Electronic Design News), vol. 39, no.4*, pp 41-46, Feb. 17, 1994.

[Sou02] Soudris, Dimitrios *Designing CMOS circuits for low power*, Kluwer Academic Publishers, 2002.

[Su94] C. Su, C. Tsui y A. Despain, "Low Power Architecture Design and Compilation Techniques for High Performance Processors", *Proc. 1994 Spring IEEE International Computer Conference (COMPCON)*,pp.489-498. IEEE Press, 1994.

[Sun03] Sung-Mo Kang, "Elements of Low Power Design for Integrated Systems", *International Symposium on Low Power Electronics and Design (ISLPED'03)*, Seoul, Korea. August, 2003.

[Tsu03] Tsugio Makimoto, Yoshio Sakai, "Evolution of Low Power Electronics and Its Future Applications" *International Symposium on Low Power Electronics and Design (ISLPED'03)*, Seoul, Korea. August, 2003.

[Vis00] Ram Viswanath, Vijay Wakharkar, Abhay Watwe, and Vassou Lebonheur, "Thermal Performance Challenges from Silicon to Systems", *Intel Technology Journal* 3er Quarter 2000.

[Wah67] Sven E.Wahlstrom. "Programmable logic arrays - cheaper by the millions". *Electronics*, 40(25):90-95, December 11, 1967.

[Xil04] Xilinx Inc. http://www.xilinx.com

[Xpo04] Xpower, "Xpower getting started" Release version 6.2.03i, 2004, available at support.xilinx.com.

[Yin04] Yingmin Li, David Brooks, Zhigang Hu, Kevin Skadron, Pradip Bose, "Understanding the Energy Efficiency of Simultaneous Multithreading", *International Symposium on Low Power Electronics and Design (ISLPED'04)*, Newport Beach, California, USA. August 9–11, 2004.

Capítulo 2:

Consumo en circuitos CMOS y su efecto en FPGAs

A fin de introducir en el tema del bajo consumo, se describe en este capítulo los fundamentos y las principales componentes del consumo en los circuitos VLSI de tecnología CMOS. Adicionalmente se presentan las principales técnicas utilizadas tradicionalmente en la reducción del consumo, con especial atención en las consecuencias de la aplicación de dichas técnicas en las tecnologías de lógica reconfigurable.

2.1. Consumo en circuitos CMOS

Como punto de partida se analiza se describe las fuentes de consumo en un circuito CMOS clásico. Para el análisis se considera un inversor CMOS como el descrito en la figura 2.1.a Hay dos tipos de disipación de corriente en los circuitos integrados aquellas generadas durante las operaciones estáticas y las generadas en las operaciones dinámicas.

El consumo estático se refiere a las corrientes que fluyen mientras no hay cambios de estado en los circuitos. El consumo dinámico, que es el más importante en la

tecnología CMOS, depende de las cargas y descargas de las capacitancias en las transiciones.

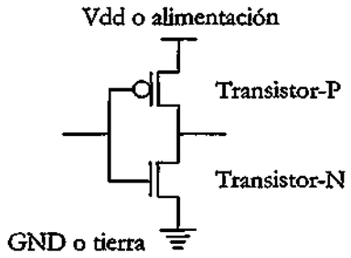


Figura 2.1. Inversor CMOS.

2.1.1 Consumo estático

En estado ideal los circuitos CMOS no disipan corriente estática en el estado de equilibrio, es decir no existe un camino directo desde *Vdd* a *Gnd*. No obstante existen dos fuentes de disipación estática a considerar [Ped96]:

- Corriente de fuga (*leakage current*) determinada fundamentalmente por la tecnología de fabricación y referida a dos fuentes: 1) A las corrientes inversas en los diodos formados entre las difusiones de la fuente (*source*) y el colector (*drain*) y la región del transistor MOS; 2) La corriente de subumbral (*subtreshold current*) que aparecen en las inversiones de tensión en la puerta (*gate*) por debajo de la corriente de umbral
- Corriente standby (*Standby Current*) que es la que siempre fluye desde *Vdd* a *Gnd* (Figura 2.2).

Las corrientes de fuga son proporcionales al área de la difusión de la fuente y el colector y a la densidad de corriente de fuga. La corriente de *subtreshold* se incrementa linealmente con la relación ancho sobre largo del canal y disminuye exponencialmente con la relación Vgs – Vt, donde Vgs es la tensión aje de la puerta y Vt la tensión de umbral.

La corriente de *standby* ocurre cuando tanto el transistor nMOS como el pMOS están continuamente en una fase de pseudo inversor nMOS, o cuando la puerta de un inversor esta en un valor entre *Vdd* y *Gnd*. En general esta potencia es igual al

producto entre V_{dd} y la corriente continua fluida entre la fuente y la tierra. Tradicionalmente el consumo estático se ha consideredo prácticamente despreciable en aplicaciones reales [Ped96][Rab96], no obstante en los últimos procesos tecnológicos por debajo de los 100 *n*m puede alcanzar el 40% del consumo total [Nar03][Liu04].

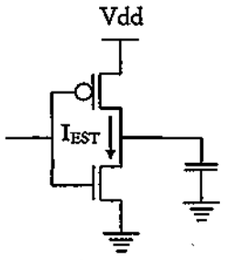


Figura 2.2. Consumo estático en un inversor CMOS

2.1.2 Consumo dinámico

Es la fuente de consumo más importante en los circuitos CMOS, se puede considerar compuesta por dos fuentes principales:

- Corriente de cortocircuito (*short-circuit current*) la que es producida a través del camino que se forma entre la *Vdd* y *Gnd* durante las transiciones.
- Corriente de capacitancia (*Capacitance Current*), la que fluye para cargar y descargar las cargas capacitivas durante los cambios lógicos.

La corriente de cortocircuito surge durante la transición de estados de un dispositivo CMOS cuando al mismo tiempo transmiten tanto los dispositivos pMOS como los nMOS (figura 2.3). La corriente de cortocircuito para una puerta inversora es proporcional al tiempo de subida o bajada de la señal (rampa), la carga y el tamaño del transistor.

Tradicionalmente la corriente de cortocircuito suele ser menor del 15% de la corriente dinámica [Vee84], no obstante los últimos procesos tecnológicos por debajo de los 0,25 *n*m esta componente suele superar el 20 % [Jun01]. Más aún según [Alv98] un diseño descuidado puede aumentar significativamente dicho valor. Tal lo expuesto, si bien no es despreciable, tampoco es un factor dominante en el consumo.

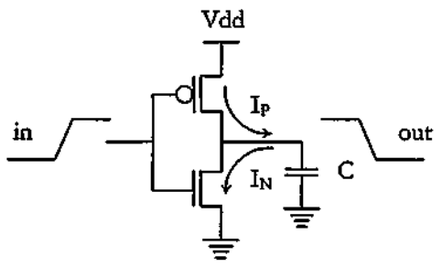


Figura 2.3. Consumo dinámico en un cambio de estado del inversor

El factor dominante en la disipación de potencia en los circuitos CMOS es debida a la carga y descarga de las capacitancias de cada nodo. Esta fuente de disipación es la llamada corriente de capacitancia y viene dada por la expresión:

$$P = \frac{1}{2} \cdot C_L \cdot V_{dd}^{\ 2} \cdot E(sw) \cdot f_{clk} \qquad \text{(Ec 2.1)}$$

Donde:

- C_L = Capacitancia física a la salida de un nodo
- V_{dd} = Tensión de alimentación
- $E(sw)$ = Actividad de conmutación (*Switching activity*), número medio de transiciones
- f_{clk} = Frecuencia del reloj

Es importante destacar que la potencia disipada depende del cuadrado de la tensión con lo cual es la variable principal para la reducción de consumo, aunque cuando no se puede variar la tecnología suele ser un aspecto poco manipulable, en tanto que la reducción de capacitancias, frecuencia de reloj y actividad son lineales y muy atractivas dado que no dependen únicamente de la tecnología para su reducción.

Para el caso de las FPGAs valen las mismas consideraciones anteriores, es decir la corriente estática ha sido mucho menor que la dinámica [Gar02], pero el consumo estático comienza a ser uno de las mayores preocupaciones [Gay04][And04] (ver mediciones de la sección 3.1.2). Se puede afirmar que la corriente estática depende de la tecnología del dispositivo y del tamaño (cantidad de bloques de cálculo, patas, etc). El porcentaje de ocupación no es desde el punto de vista del consumo estático

demasiado relevante (ver experimentos capítulo 3). El consumo dinámico en los dispositivos reprogramables es también proporcional a la frecuencia y al cuadrado de la tensión (ver experimentos sección 3.1.1).

2.2. Espacio de diseño para bajo consumo

Como expresa la ecuación 2.1 el consumo depende de tres factores: Tensión de alimentación, capacidades físicas y actividad de los datos. La intención de reducir consumo invariablemente se relaciona con reducir uno o más de estos factores. Desafortunadamente estos factores no son independientes entre si y en ocasiones no pueden ser optimizados independientemente uno de los otros. A continuación se realiza una clasificación de ellos.

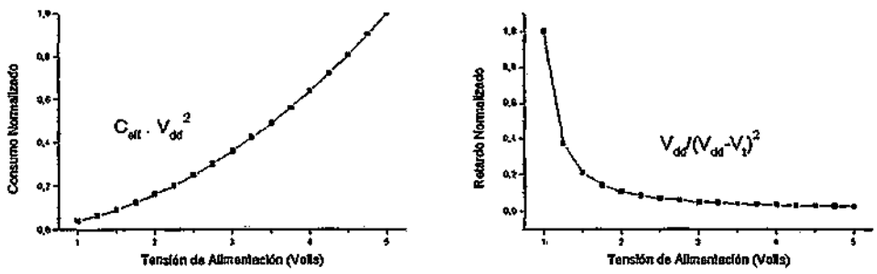


Figura 2.4. Relación energía-tensión y retardo-tensión respectivamente.

2.2.1 Tensión de alimentación.

Sin dudas es el mas atractivo de los parámetros a manipular dada su influencia cuadrática en la ecuación del consumo, es la tensión de alimentación. Muchos fabricantes están dispuestos a sacrificar capacidad en el circuito o actividad de los datos con tal de reducir la tensión. Lamentablemente se paga un costo en velocidad muy alto cuando *Vdd* se acerca *Vt* (Tensión de umbral), esto limita la reducción de consumo a un mínimo de dos a tres veces la tensión de umbral (Figura 2.4.b.). Los retardos en un circuito son proporcionales a $T \propto V_{dd}/(V_{dd}-V_t)^2$.

La aproximación más atractiva a la hora de reducir tensión sin pérdida de velocidad es la reducción de la tensión de umbral (V_t). El límite para la reducción de V_t esta dado por el margen de ruido soportado y el aumento en la corriente de fuga de subumbral.

En cuanto a los circuitos de lógica programable, si bien existen familias operando a diferentes tensiones (5V y 3,3V tradicionalmente, y los más nuevas como la familia Virtex II / Spartan III a 2,5V; 1,8V y 1,2V), no es aconsejable con vista a la interoperación con otros dispositivos operar un circuito fuera de los rangos provistos por el fabricante. Existen no obstante trabajos en esta línea donde se prueban diseños operando a diferentes tensiones [Gar00], los que permiten evaluar la relación consumo tensión respecto de la merma en el rendimiento.

2.2.2 Capacidad

La reducción de la capacitancia ofrece una disminución lineal del consumo, de modo que la opción es otra atractiva fuente de minimización. La capacidad en un circuito CMOS depende mayoritariamente de dos factores, por un lado los dispositivos (puertas lógicas) y por otro las interconexiones. En las tecnologías antiguas las capacitancias de los dispositivos dominaban sobre las interconexiones. Conforme las tecnologías siguen reduciéndose, las capacitancias de las interconexiones empiezan a ser más importantes.

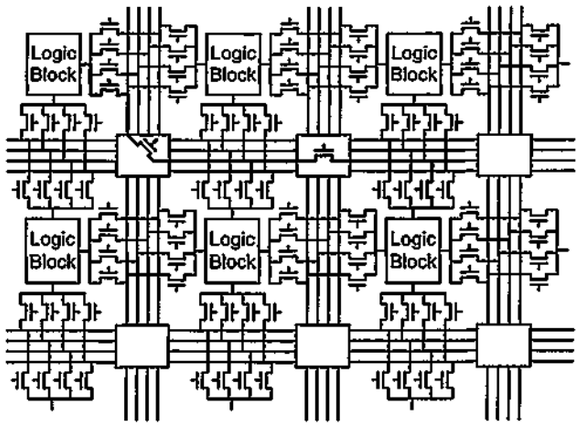


Figura 2.5. Rutado simplificado en dispositivos tipo FPGA

En las FPGAs es muy significativa la capacidad debido a las interconexiones. Es causa de más del 50% del consumo según [Kus98][Sha02][Poo02], tal como se explica en la sección 2.10. En los dispositivos repropramables, los canales de rutado son fijos y se debe pasar por una o múltiples matrices de interconexión para conectar dos elementos de cálculo (figura 2.5), he ahí uno de los motivos para esta gran influencia del rutado.

Para la reducción del consumo debido a las capacidades en FPGAs existen básicamente dos posibilidades. La primera y más evidente es la reducción del largo de las pista. La utilización de herramientas optimizadas de emplazamiento y rutado (*place and route*) ayudan a la disminución del consumo. La segunda resulta de la reducción de lógica de cálculo, aunque esto puede provocar aumentos en la actividad o bien aumentar los recursos de rutado necesario.

2.2.3 Actividad del circuito

Al margen de la tensión y la capacidad, la otra fuente de consumo dinámico es la actividad en el circuito. La actividad del circuito depende de dos factores. Por un lado f_{clk} que especifica la periodicidad promedio de arribo de datos y *E(sw)* que determina cuantos cambios puede determinar cada arribo. La actividad es muy difícil de calcular, dado que depende de las entradas y de la función a computar. Por ejemplo, la actividad dentro de un multiplicador de 16 bits puede variar en un factor de 5 dependiendo de la correlación entre los datos de entrada [Mar95].

En la figura 2.7 se observa la interpretación de los conceptos de f_{clk}, el que se puede ver como la frecuencia de funcionamiento de un circuito síncrono y *E(sw)* que expresa cuantas transiciones se desarrollan en un ciclo de reloj. En un circuito sin *glitches*, *E(sw)* puede ser interpretada como la probabilidad que ocurra una transición durante un ciclo de reloj, en tanto en un circuito con *glitches* especifica tanto la probabilidad de ocurrencia de una transición, así como la cantidad de movimientos espurios generados por ésta.

Los *glitches* se refieren a las transiciones espurias o indeseadas producidas antes que un nodo alcance un estado final (estable). Los *glitches* se producen frecuentemente cuando los caminos con tiempos de propagación desbalanceados llegan a un mismo punto del circuito. Esto produce que algunos nodos realicen varias transiciones que producen consumo (es decir E(sw) > 1), esto naturalmente debe ser evitado siempre que sea posible. Algunos trabajos sugieren que en tecnologías ASIC la actividad espuria suele

ser entre 20% y 70% [She92], en tanto en [Rab96b] se consigna que el consumo debido a los glitches es típicamente de un 15-20%, pero para unidades aritmeticas puede llegar al 65%.

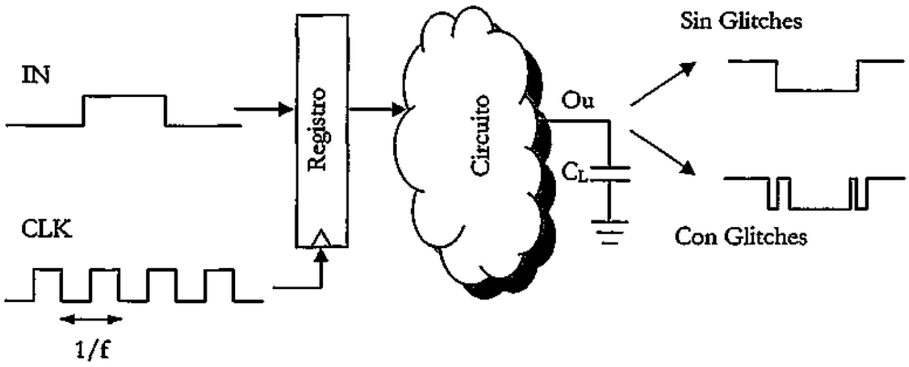


Figura 2.6. Interpretación de la actividad en circuitos síncronos: Glitches

Por comodidad, se suele combinar la actividad de los datos *E(sw)*, con la capacidad física (½.C_L) para obtener la *capacidad efectiva* $C_{eff} = ½ . C_L E(sw)$. La que se refiere a la capacidad promedio cargada en cada ciclo de reloj, transformado así la expresión del consumo en:

$$P = \frac{1}{2} \cdot C_L \cdot V_{dd}^{\ 2} \cdot E(sw) \cdot f_{clk} = C_{eff} \cdot V_{dd}^{\ 2} \qquad \text{(Ec. 2.2)}$$

Esto refleja que el consumo no depende ni de la frecuencia de funcionamiento, ni la capacidad, ni la actividad de conmutación por separado, sino de la integración e interacción de estos factores en la capacidad efectiva.

La figura 2.9 ilustra la existencia de transiciones espurias producidas por la diferencia en el retardo de dos caminos. El efecto se refuerza con el aumento de la profundidad lógica, que produce avalanchas de *glitches* (figura 2.7.b).

Como en el caso de la tensión y la capacidad, existen técnicas para la reducción de actividad como método de reducción del consumo. Por ejemplo, ciertas representaciones de datos como las de signo y magnitud poseen inherentemente menor actividad que el complemento a dos [Cha94]. Dado que la implementación de las operaciones en signo magnitud son más complejas que en complemento a la base

se debe pagar un costo extra en superficie de silicio así como consecuentemente en capacidad del circuito. Como siempre el problema de optimización de las tres variables tensión-capacidad y actividad no puede ser considerado independientemente y sin considerar el impacto sobre los demás factores.

En el caso de las FPGAs la reducción de la actividad puede ser tanto encarada desde la representación de los datos como de un mapeado tendiente a la reducción de actividad, reconociendo partes del circuito con mayor actividad y agrupándolas en lugares vecinos. Casi todas las familias de FPGAs poseen registros y latches entre sus elementos de cálculos. Esto permite implementar exitosamente diseños segmentados los que tienen entre otras la ventaja de reducir *glitches*.

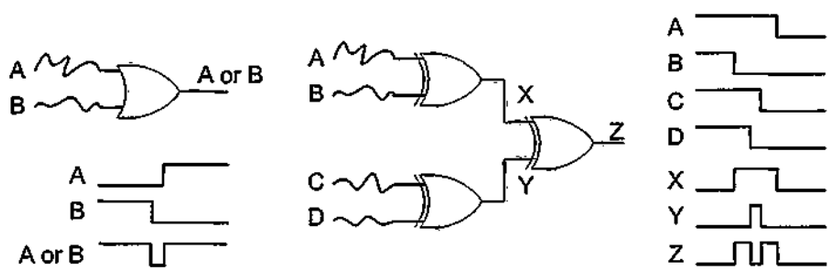


Figura 2.7. Transiciones espureas (*glitches*). a. Producidos por el desbalance en el arribo de las señales. b. Efecto avalancha debido al aumento de la profundidad lógica

Los *glitches* en las FPGAs son muy significativos y se ven magnificados por el efecto de retardo que originan las matrices de interconexión. Para un multiplicador de 16 bits se pueden observar que señales internas pueden tener una actividad del orden de 16 veces superior a la frecuencia de las entradas y uno de 32 más de 36 veces mayor (datos obtenidos por el autor a través de simulaciones *post place & route*). La sección 3.4 analiza y cuantifica la reducción del consumo debido a la reducción de *glitches* por medio de la técnica de segmentación.

2.3 Temas destacados en el diseño para bajo consumo

La primera observación surge de la ecuación 2.1, donde se observa que el consumo es determinado por tres parámetros importantes como son: la tensión, la capacidad del

circuito y la actividad del mismo. No obstante la optimización de estos no puede ser interpretada individualmente sino que debe ser interpretada conjuntamente.

Quizás el tema más importante es el balance entre área-velocidad para bajo consumo. La mayor parte de las técnicas de bajo consumo son a expensas del área o la velocidad, valga como ejemplo la reducción de la tensión de alimentación la que conlleva un ahorro cuadrático del consumo pero afecta directamente a la velocidad de ejecución. Si el diseñador no puede perder performance en la reducción del consumo posiblemente se vea obligado a utilizar técnicas de segmentación (*pipeline*) o paralelización para compensar la velocidad de ejecución, aumentando inevitablemente el área del circuito.

Un tema recurrente en el diseño de bajo consumo es el análisis de la localidad. Las operaciones globales consumen mucha potencia. Los datos que se mueven de una punta a la otra del circuito deben conmutar una gran capacidad de pistas. Un particionado pobre puede significar aun la necesidad de replicar datos en otras partes del circuito aumentando aun más la capacidad de este. En caso de las FPGAs con canales de rutado fijos y pasando por matrices de conmutación este efecto es más importante aún, el aprovechamiento de los diferentes canales de rutado y lugares de almacenamiento posee un fuerte impacto en el consumo. Los árboles de reloj globales tienden a atentar contra el bajo consumo.

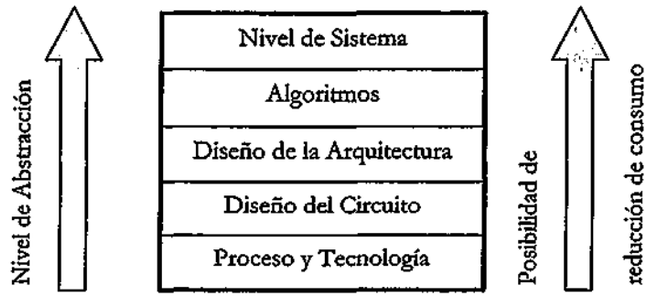


Figura 2.8. Jerarquía en el espacio de diseño

Otro tema importante cuando se habla de bajo consumo es "evitar derroches" (*avoiding waste*), entre estos se cuentan evitar *gliches* ecualizando caminos, "apagar" partes del circuito cuando estos no se utilizan (quitando el reloj o bloqueando los datos de entrada), etc. Otras fuentes de derroches de consumo son la no utilización de algoritmos y arquitecturas regulares, forzando de esta manera más lógica y actividad.

Por otra parte la utilización de procesadores o elementos de cálculos más potentes (rápidos) que los necesarios puede ser una fuente de derroche de consumo, perfectamente evitable.

En lo que resta del capítulo se realiza un análisis de las técnicas de reducción del consumo organizados por el nivel de abstracción. Los niveles a considerar son 5 y se detallan en la figura 2.8.

2.4. Nivel de proceso y tecnología

En el nivel más bajo de abstracción se pueden considerar el proceso de fabricación así como el encapsulado (*package*) del circuito. En cuanto a las FPGAs estos factores son determinados por el fabricante y existe poco margen de elección, mas allá de elegir ciertos dispositivos dentro de la gama de oferta de circuitos reconfigurables, los que eventualmente operen a menor tensión, utilicen un proceso de fabricación más moderno, provean arquitecturas que se adapten mejor al bajo consumo, etc.

2.4.1 Encapsulado (*package*)

Generalmente una fracción importante del consumo total puede ser atribuida a las grandes capacitancias manejadas fuera del circuito (*off-chip power*) en vez de al circuito en si (*core*). Esto se apoya en el hecho de que las capacidades fuera del chip se miden en picofaradios en tanto que normalmente las internas se hacen en términos de femtofaradios. Circuitos con gran interacción de entrada/salida (I/O – *input/output*) pueden consumir hasta ¼ o ½ de la potencia total.

La tecnología de encapsulados ha evolucionado y está evolucionando rápidamente producto del aumento de densidad de los circuitos y la multiplicación que han sufrido las entradas y salidas de los IC (*Integrated Circuit*). Con esto también aparecen nuevos estándares en entrada-salida que reducen considerablemente la capacidad de los patas (*pads*) de conexión.

Los encapsulados de las FPGAs siguen las tendencias de toda la industria de IC. Es importante destacar que el aumento en la potencia de cálculo que experimentan las nuevas familias de dispositivos también hacen posible integran en un solo circuito más lógica, lo que eventualmente redunda en una menor actividad de entrada salida. Las FPGAs han evolucionado de ser una solución para generar lógica de conexión (*glue*

logic) a ser utilizadas para implementar lo que se da en llamar SoC (*System-on-a-Chip*, sistema en un chip) haciendo relativamente menos voluminosa la comunicación con otros dispositivos electrónicos.

2.4.2 Proceso de fabricación

Al margen de las consideraciones del encapsulado, la tecnología de fabricación posee una importante componente en la reducción de consumo. Varias modificaciones en el proceso de fabricación reducen el consumo en los circuitos integrados CMOS. Muchas de estas ocurren con cada nueva generación de procesos de fabricación (reducción del tamaño de las puertas, agregado capas de metal, etc). Estos parámetros no están disponibles para los diseñadores de IC y mucho menos para quien diseña con FPGAs, no obstante es importante conocer las causas del consumo y las posibilidades de reducción en este nivel.

2.4.2.1 Optimización de la tensión umbral (*Threshold Voltaje-Vt*)

Cuando se pretende reducir la tensión como método para la disminución del consumo, la tensión umbral comienza a tener gran importancia dada su directa influencia en la velocidad de operación (inversamente proporcional a $(V_{dd} - V_t)^2$). Lamentablemente, la conducción subumbral y consideraciones de márgenes de ruidos limitan cuan baja puede ser situada V_t.

2.4.2.2 Reducción en la tecnología (*Technological scaling*)

La reducción de las dimensiones físicas es una técnica muy conocida para reducir el consumo. La reducción significa disminuir todas las dimensiones horizontales y verticales por un factor S mayor que uno. De este modo se reduce el alto y ancho de los transistores, grosor de oxido, ancho y alto de las interconexiones, etc. Si se considera que la capacidad de una puerta es $Cg = W.L.C_{ox}$ donde, significan el ancho y largo de la difusión de la puerta y la capacidad por unidad de superficie respectivamente. Y si se reduce por un factor S, cada una de las dimensiones W, L y C_{ox} entonces la capacidad se reducirá por un factor S también, de este modo si la tensión y la actividad de los datos se mantienen la potencia se disminuirá en un factor de S también ($P \propto 1/S$).

Existen no obstante otros factores negativos en las interconexiones, la resistencia de estas es proporcional al largo e inversamente proporcionales al ancho y el grosor. Si escalamos por un factor S, la residencia aumentará también S. Respecto de la capacidad esta es proporcional al ancho y largo e inversamente proporcional al grosor, con lo que la capacidad disminuirá en un factor S. Resumiendo, el retardo debido a las interconexiones no se ve afectado por la reducción de dimensiones, por tanto, los retardos de pistas comienzan a dominar sobre el retardo de las puertas, que es la situación de hoy en día.

2.4.2.3 Trazado (*Layout*)

La técnica más simple a nivel *layout* es colocar las pistas de mayor actividad en las capas superiores. Los niveles superiores de metal están separados por una capa más gruesa de dióxido de silicio. Dado que la capacidad física decrece linealmente con el incremento de del espesor del oxido, existen claras ventajas de rutar las líneas de mayor actividad en las capas superiores. Según [Rab95] en un proceso típico de fabricación la capa tres puede tener hasta cerca del 30% menos de capacidad por unidad de superficie que la capa dos.

2.4.2.4 Dimensiones de los transistores (*Transisitor sizing*)

Otro punto en discusión en bajo consumo es el tamaño de los transistores. Transistores con puertas grandes pueden manejar mas corriente que los pequeños, pero esto contribuye con mayores capacidades en el circuito, con el obvio aumento del consumo. Más aún los dispositivos más grandes poseen corrientes de cortocircuito mayores. La estrategia entonces es disminuir el tamaño de los transistores siempre que sea posible. No obstante la disminución del tamaño conlleva perdida de performance que no siempre son aceptables.

2.5. Nivel de implementación

Se refiere este nivel a la forma de implementar los diseños en base a los recursos que propone la arquitectura de los circuitos reprogramables. Nuevamente aquí se discute la relación de performance, área y consumo ahora desde este nivel de abstracción. Aquí se discuten el mapeo (*mapping*), reducción de *glitches* y actividad, formas de sincronización, así como métodos de concurrencia y redundancia para la disminución de consumo.

2.5.1 Descomposición tecnológica y mapeo

Se denomina descomposición tecnológica al proceso de transformar una descripción booleana a nivel de puertas en un mapeo para la arquitectura elegida. En el caso de implementaciones en ASICs será elegir la configuración de transistores que realizará la función, o en caso de lógica reconfigurable elegir la forma de mapear a los recursos básicos (LUTs, lógica de acarreo, Block RAMs, multiplicadores embebidos, etc) la funcionalidad de la descripción.

Por ejemplo en el caso de ASICs una puerta NAND de tres entradas puede ser implementada por una simple puerta de tres entradas o bien por una cascada de puertas simples de dos entradas. Para el caso de la lógica reconfigurable, la forma de descomponer las funciones en tablas de *look-up* (o multiplexores) ofrece múltiples alternativas. Algunos de los criterios para esta descomposición se ofrecen a continuación.

2.5.2 Reordenar las entradas

Una simple estrategia para disminuir la actividad total, será posponer las señales que poseen mayor movimiento. Por ejemplo, simplemente reordenando las entradas en una cascada de cálculos se puede reducir la cantidad de transiciones totales (figura 2.11). La probabilidad de que exista transición Tr para una señal en la que se conoce la probabilidad de ser uno (p) en un modelo sin retardos viene dado por la expresión:

$$Tr = 2 \cdot p \cdot (1 - p) \qquad \text{(Ec. 2.3)}$$

Naturalmente es la suma de la probabilidad de que pase la señal de 0 a 1 más la que pase de 1 a 0. La P(0 → 1) = P(1→0) = p.(1-p).

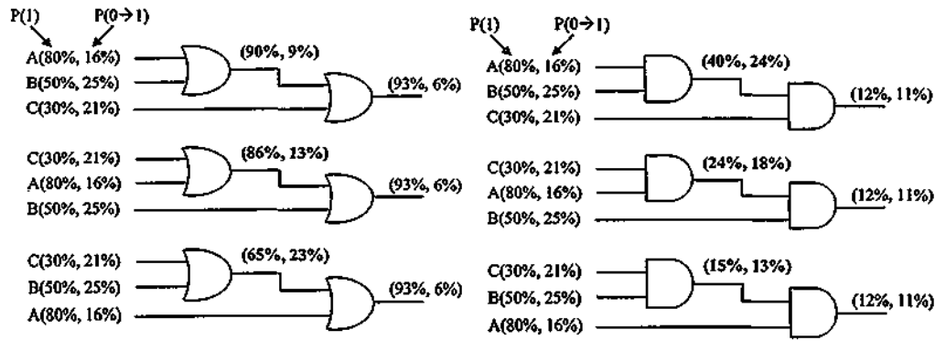


Figura 2.9. Dependencia del orden de entrada para la reducción de actividad

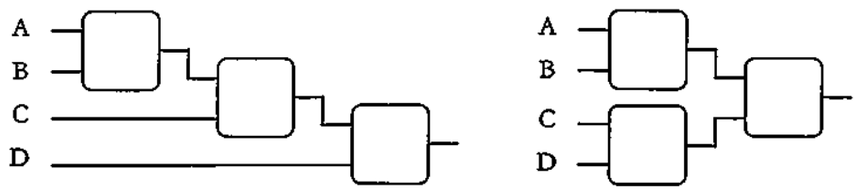
2.5.3 Reducción de *glitches*

Una de las técnicas más importantes para la reducción de consumo es la de evitar las transiciones espurias asociadas a los *glitches* (ver sección 2.3). En la figura 2.10 se puede observar dos implementaciones de una función lógica. Una de las implementaciones posee una estructura en cascada en tanto que en la otra se encuentra balanceada. Si se supone la llegada de las señales al mismo tiempo se puede observar que la opción balanceada realiza menos transiciones que la versión en cascada. Las eventuales transiciones espurias pueden provocar transiciones a las puertas conectadas la salida amplificando el efecto. Basados en este razonamiento se puede inferir una cota superior para las transiciones espurias (*glitches*) de O(N) donde N es la profundidad lógica.

En contrate si se logra una estructura donde la llegada de las señales a cada puerta sucede al mismo tiempo se evitan las transiciones espurias. Este concepto de "balancear" las estructuras arbóreas, se puede extender como técnica de reducción de consumo. Algunos estudios sugieren reducciones de hasta 15-20% del consumo [Ped96].

Al margen de la utilización de estructuras balanceadas en forma de árbol, para lograr el balance de los retardos de caminos se suelen utilizar otras técnicas como la

incorporación de *buffers*, o en caso de lógica programable, la utilización de elementos de cálculo que solo se utilizan como retardos para lograr la ecualización de los caminos.



(a) Estructura de cascada (b) Estructura balanceada

Figura 2.10. Estructuras de cascada y balanceadas.

Como idea general dado que los *glitches* tienen como cota superior la profundidad lógica al cuadrado es interesante lograr circuitos de poca profundidad lógica, este es uno de los argumentos a favor del uso de técnicas de *pipeline* en la reducción de consumo.

Las técnicas de balanceo son difíciles de implementar en FPGAs, ya que la mayor componente del retardo lo representan las pistas (que pasan por matrices de interconexión), las que a su vez son difíciles de balancear. En la figura 2.11 se muestra un ejemplo simple de 4 conexiones que llegan a una LUT para demostrar el inherente desbalanceo que se provoca en estos dispositivos.

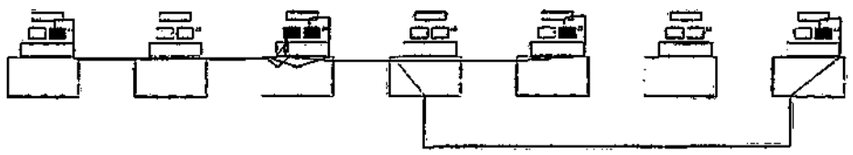


Figura 2.11. Desbalanceo en las pistas. El ejemplo se muestran 4 conexiones adyasentes a una LUT en un dispositivo Virtex.

2.5.4 Concurrencia y redundancia

La idea de la concurrencia es al aumento de las prestaciones del circuito (aunque a expensas del aumento del área) con el objeto de reducir la tensión de alimentación la que tiene una influencia cuadrática con el consumo. Cuando por cuestiones de diseño

del circuito o de interrelación con otros sistemas, la tensión no puede ser modificado esta técnica pierde atractivo.

Por otra parte el objetivo de la redundancia, es básicamente reducir los *glitches*, o bien a través de la ecualización de caminos o bien el evitando conexiones con un *fan-out* demasiado elevado. En el primer caso, si una señal debe llegar a puntos muy distantes de un circuito puede convenir replicar un modulo de modo de evitar retardos indeseables que ocasionan *glitches* y aumenten el consumo. El segundo caso, un *fan-out* muy grande implica puertas y pistas de mayor capacidad y eventualmente *buffers*, esto invariablemente genera retardos que pueden atraer movimientos espurios. La redundancia trae aparejado el aumento del área y la capacidad del circuito, de modo que debe sopesarse correctamente con el ahorro en potencia producida por los *glitches*.

2.6. Nivel de arquitectura y sistema

En el nivel de arquitectura las primitivas son bloques como multiplicadores, sumadores, memorias, *buses*, controladores, etc. En la bibliografía se lo suele llamar nivel de transferencia de registros (*Register-Transfer (RT) Level*) o los científicos del área informática nivel de micro arquitectura (*micro-architecture level*). Las técnicas aquí utilizadas tratan de evitar el derroche en el consumo, explotar la localidad de datos y el balance de área - velocidad para bajo consumo.

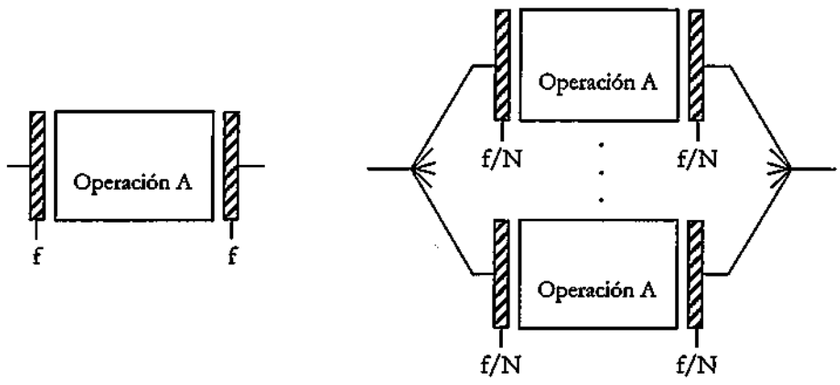
2.6.1 Procesamiento concurrente

A nivel arquitectural el procesamiento concurrente es un excelente ejemplo de balance de área – velocidad para bajar el consumo. A través de técnicas conocidas para aumentar la performance como lo son el paralelismo o la segmentación (*pipeline*) se aumenta el rendimiento para luego bajar la tensión de alimentación y así ahorrar consumo. Naturalmente estas técnicas están limitadas por los costos de interconexión, es decir si se desea realizar un circuito masivamente paralelo llega un momento que los costos de conexión superan a los beneficios.

2.6.1.1 Paralelismo

Se supone el ejemplo de la figura 2.12, donde un cálculo complejo A, se puede llevar a cabo en un cierto tiempo. Los registros de entrada y salida capturan a una frecuencia *f*.

Si se supone que no existe dependencia de los datos de modo que se pueda paralelizar sin restricciones. Si se paraleliza el cálculo, duplicando N veces el bloque de cálculo A, se tendrá N procesadores idénticos los que podrán funcionar a una frecuencia N veces inferior y sin embargo mantener la velocidad total de cálculo.



Circuito de Referencia		Circuito Paralelo	
Capacidad	= C	Capacidad	~ N.C
Frecuencia	= f	Frecuencia	= f / N
Voltaje	= V	Voltaje	~ V/N (despresiando Vt)
Power	**= P ∝ C.V².f**	**Power**	**~ P / N²**

Figura 2.12. Reducción de tensión y paralelismo para bajo consumo [Lan96].

La clave para la reducción del consumo es la posibilidad de reducir en un factor N la frecuencia de cálculo. Dado que la velocidad de cálculo se puede considerar aproximadamente lineal a la tensión de alimentación, esto permite reducir la tensión en un factor N. La capacidad total del circuito se incrementa N veces (dado que hay N procesadores iguales y despreciando la sobrecarga que representa la interconexión). Sobre la base de la ecuación 4 donde se deduce que $P \propto C.V^2 f$, se puede inferir que la potencia para una concurrencia de N niveles (P_n) es:

$$P_n \propto C \cdot N \cdot \left(\frac{V}{N}\right)^2 \cdot \frac{f}{N} \propto \frac{P}{N^2} \qquad \text{(Ec. 2.4)}$$

Este modelo simplificado no tiene en cuenta algunos aspectos que restan valor a la técnica. En primer lugar la inclusión de N procesadores no siempre implica un

aumento en N de la velocidad total. Frecuentemente los algoritmos no son totalmente paralelizables y existen tareas que deben ser necesariamente ejecutas secuencialmente, o bien en nivel deparalelismo es limitado. Otro aspecto importante es existe una sobrecarga de conexiones y distribución de señales tanto para abastecer las entradas como para combinar nuevamente las salidas, aumenta la superficie (y por tanto la capacidad) en un factor mayor a N. Por ultimo cabe mencionar los efectos de la tensión umbral (V_t), que hace que no se pueda reducir indefinidamente la tensión, además que como se mencionó en la ecuación 2.5, la velocidad de un circuito se puede considerar lineal a la tensión cuando $V_{dd} >> V_t$, más precisamente la expresión dice que $T \propto V_{dd} / (V_{dd}-V_t)^2$.

Como se mensiono en la sección 2.2.1 la disminución de la tensión de alimentación en FPGAs no resulta muy atractiva, por tanto esta técnica pierde interés en el marco tecnológico de la lógica reconfigurable.

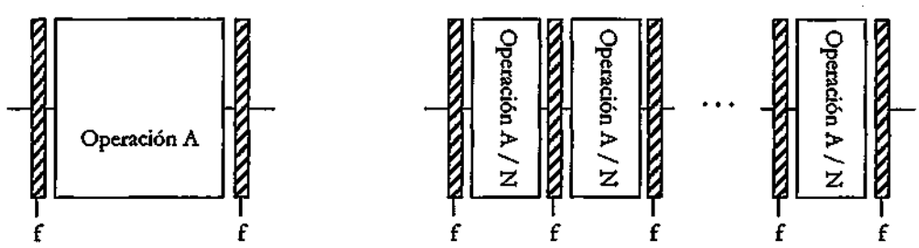
2.6.1.2 Segmentación (*pipeline*)

El *pipeline* es otra de las técnicas de computación concurrente que puede ser explotada para reducir el consumo. En este caso en vez de duplicar hardware se procede a particionar la operación A en N suboperaciones colocando registros entre ellas y logrando un *pipeline* de N etapas. La capacidad total (despreciando los registros) puede considerarse similar a la versión original. En este caso para mantener la misma velocidad de operación se debe mantener la frecuencia f, pero en cada subetapa debe calcularse solo una N-esima parte del cálculo total, lo que permite disminuir en N la tensión de alimentación. De este modo la reducción de la potencia dinámica será un factor N^2.

Como ocurre con el paralelismo, el *pipeline* incurre en alguna sobrecarga aunque no tan notoria como en el primer caso. Aquí se deben agregar tantas etapas de registros como etapas de *pipeline* se utilicen generando una mayor superficie, capacidad y necesidad de distribución de reloj. Al aumentar los niveles de *pipeline* el consumo de registros de sincronización puede superar al consumo debido al cálculo. Desde el punto de vista del rendimiento del circuito se debe tener en cuenta la existencia de la latencia, que es el retardo necesarios para obtener el primer resultado (N ciclos de reloj).

Existe otra ventaja en la utilización de *pipeline* independiente de la reducción de la tensión de alimentación y es la referida a la disminución de *glitches* que genera la

reducción de la profundidad lógica. En efecto, los registros de sincronización detienen la avalancha de *glitches* que se suceden en la lógica combinacional [Boe96]. En esta tesis se ha cuantificado experimentalmente esta influencia la sección 3.3.



Circuito de Referencia

Capacidad = C
Frecuencia = f
Voltaje = V
Power = $\mathbf{P \propto C.V^2.f}$

Circuito con Pipeline

Capacidad ~ C
Frecuencia = f
Voltaje ~ V/N (despresiando Vt)
Power ~ $\mathbf{P / N^2}$

Figura 2.13. Reducción de tensión y *pipeline* para bajo consumo [Lan96].

2.6.2 Manejo de potencia (*power managemet*)

Cualquier consumo de potencia que no produzca una operación útil es un desperdicio de energía. Las estrategias para evitar el desperdicio de energía se denominan manejo de potencia o energía (*power managemet*) entre las técnicas utilizadas se cuenta con apagado selectivo (*selective power down*), modo descanso (*sleep mode*) y sistemas adaptativos de la velocidad de reloj y tensión.

Se denomina apagado selectivo a la desactivación específica de módulos que no están realizando una computación concreta. Esta estrategia exige de circuitos adicionales para detectar la inactividad y para desactivar y activar los módulos individualmente. La forma tradicional de llevar a cabo esta técnica es desactivando el reloj, eliminando de esta manera la componente dinámica del consumo. Los circuitos asíncronos proveen esta técnica incorporada, ya que no producen cálculo a menos que se lo requiera explícitamente.

El modo descanso es una extensión de la técnica anterior y se trata del apagado de todo el circuito. Para esto se monitorea el sistema y en caso de estar ocioso se procede al apagado del sistema en lo que se conoce como modo descanso o sueño (*sleep mode*). Durante este modo se monitorean las entradas en busca de las condiciones que despierten al sistema y pase al modo de uso normal. El cambio de modos de funcionamiento representa una sobrecarga tanto en tiempo como en consumo, de modo que si los periodos de descanso no son suficientemente largos esta estrategia pierde atractivo.

La tercera técnica en cuestión se trata de ajustar tanto sea la frecuencia del reloj como la tensión de alimentación de modo de cumplir los requerimientos de cálculo (la filosofía de no tener más potencia de cálculo que la necesaria). Si bien las tareas a realizar por un circuito varían con el tiempo, el problema reside en como detectar o predecir la potencia de cálculo necesaria en cada momento. Una aproximación es a través de instrucciones software para la reducción de velocidad como en algunos procesadores MIPS (VR4K) que reduce al 25% la velocidad de reloj o por algún sistema de retroalimentación interna. En esta línea Intel en la tecnología *SpeedStep* reduce la velocidad del reloj de sus procesadores al funcionar con baterías en vez de la red eléctrica [Int00].

La línea de microprocesadores MIPS RISC de 64 bits VR4100 de Nec Corporation combina la reducción de velocidad por software con cuatro modos de funcionamiento para reducción de consumo (*full speed, standby, suspend, and hibernate mode*), donde se apagan selectivamente ciertas partes del circuito [nec98].

Los dispositivos programables de Xilinx no poseen modo *sleep, power down* o similares, de hecho fue quitado de los dispostivos Virtex/VirtexII que originalmente lo tuvieron. En la sección 3.6 se estudian algunas alternativas para desactivación de partes inactivas del circuito en FPGAs.

2.6.3 Particionado

Datos globales, significan memorias globales, con señales a través de todo el circuito que conmutan altas capacidades y disipan mucha potencia. Una técnica altamente difundida es la participación para aprovechar de la localidad de los datos, el proceso distribuido es de mayor eficiencia de consumo que los procesos centralizados.

Al margen de los problemas de exactitud y longitud de palabra, el diseñador debe seleccionar también una aritmética de representación para los datos. Por ejemplo, complemento a dos (C2), signo magnitud (también llamado signo valor absoluto: SVA) ó cero desplazado (CD) son posibles representaciones aritméticas de datos. Sin duda el complemento a dos es el sistema más ameno de utilizar, y por ende, el más utilizado. En esta representación, los bits menos representativos (LSBs) son bits de datos, en tanto los bits más representativos (MSBs) son bits de signo. Como resultado los bits MSBs contienen información redundante que conlleva a una mayor actividad (más consumo) cuando hay una cantidad de transiciones de signo importante. Como contrapartida la representación en signo y magnitud, utiliza un solo bit para representar el signo y por ende solo necesita cambiar un bit para cambiar de signo. En la figura 2.14 se muestran dos ejemplos de cambio de signo en complemento a dos (C2) y signo magnitud (SVA).

Número decimal	Numero en C2 de 16 bits	Numero en SVA de 16 bits
10	0000 0000 0000 1010	0000 0000 0000 1010
-10	1111 1111 1111 0110	1000 0000 0000 1010
	Total transiciones: 14	Total transiciones: 1
1245	0000 0100 1101 1101	0000 0100 1101 1101
-1245	1111 1011 0010 0011	1000 0100 1101 1101
	Total transiciones: 15	Total transiciones: 1

Figura 2.14. Ejemplos de cambio de signo en complemento a dos (C2) y signo magnitud (SVA)

No obstante, en la mayoría de los casos, la superior complejidad de las operaciones en signo magnitud supera con creces los eventuales beneficios de la menor actividad. En algunos casos, utilizando líneas globales de mucha capacidad y pocas operaciones aritméticas puede ser conveniente pasar a signo magnitud.

Un tema relacionado es el referente a la codificación de datos. De alguna manera esta discusión trata de evitar el derroche en la elección del esquema de representación. Por ejemplo, los bits de signo en la representación de complemento a dos puede ser considerado un desperdicio en el ancho de datos. Otro problema típico es la necesidad de tener buena precisión en números pequeños pero no tanta en números grandes. Si esto se soluciona con una representación como el complemento a dos que posee una cuantificación lineal no se saca provecho de esa representación, la solución podría

Por ejemplo las memorias tienen un consumo proporcional al tamaño, si para un proceso cualquiera realizado por un único procesador accediendo a una única memoria, se puede dividir el proceso en N procesadores accediendo a N memorias se puede lograr un ahorro conceptual de N veces. Con las maquinas de estados que implementan controladores se puede efectuar un razonamiento análogo. Si un control centralizado debe abastecer todo un circuito las líneas serán globales y de alta capacidad con el correspondiente aumento de consumo respecto de una opción descentralizada que solo intercambian algunos pocos bits necesarios y el resto de proceso se realiza local.

2.6.4 Representación de los datos

Otra técnica para la reducción de consumo es la elección de la representación de los datos. El diseñador dispone de diferentes alternativas a escoger, por ejemplo: punto fijo vs. punto flotante, signo magnitud (signo valor absoluto - SVA) vs. complemento a dos (C2), datos codificados vs. datos sin codificar. Cada decisión involucra ponderar diferentes aspectos como exactitud, facilidad de diseño, prestaciones, área, consumo. En ésta sección se discuten algunos problemas involucrados en la selección de la representación de datos para bajo consumo.

Uno de las decisiones más obvias, es decidir utilizar punto fijo o punto flotante. El punto fijo, como es de esperar, necesita menos hardware y por consiguiente menos consumo. Desafortunadamente, sufre dificultades en cuanto al rango de representación, escalar los datos por software puede ser una solución pero requiere naturalmente de decodificación extra. El punto flotante, por contraste, modera las dificultades del rango de representación a expensas de utilizar mucho más hardware (consecuentemente más capacidad, y más consumo). Por esto, la representación de punto flotante debe ser elegida solamente cuando las necesidades del rango de representación lo exijan.

Una decisión relacionada, involucra selección de la longitud de palabra requerida para la ruta de datos. Frecuentemente se sobrestiman los requisitos del rango de representación, construyendo arquitecturas con grandes márgenes de seguridad y por ende usando anchos de palabras más grandes de los requeridos por la aplicación. Estas decisiones apresuradas traen indiscutiblemente un aumento del área y el consumo.

pasar por usar punto flotante. Otra alternativa es utilizar una codificación logarítmica, que complica muchas operaciones básicas (como la suma), pero que puede hacer multiplicaciones con simples sumadores.

Siguiendo con el tema de codificación de la información, se puede mencionar la codificación de estados en maquinas de estados finitos (FSMs). Las codificaciones binarias ofrecen una manera compacta de expresar estados pero que pueden tener muchas transiciones si la cantidad de estados (bits para representar los estados) es grande. La codificación *one-hot* (solo un bit a 1 el resto a 0) ofrece una distancia constante de dos transiciones entre cualquier par de estados. Dentro de las codificaciones binarias, los códigos de Gray o Johnson ofrecen soluciones con menores transiciones para ciertos tipos de problemas de codificación. Este aspecto se estudia detalladamente en el capítulo 4 sobre máquinas de estado.

Visiblemente existen múltiples alternativas a la hora de elegir una representación de datos para sistemas de bajo consumo. Desafortunadamente no existe una opción ideal para aplicaciones de bajo consumo. Por el contrario, el diseñador debe realizar un análisis del sistema en términos de precisión requerida, prestaciones y operaciones a realizar sobre los datos para determinar el sistema de representación idóneo para cada aplicación.

2.7. Nivel algorítmico

Los algoritmos tienen influencia directa e indirecta sobre el consumo. Por ejemplo, la complejidad algorítmica y la cantidad de operaciones tienen una influencia directa en el consumo. Otras características como la posibilidad de aplicar concurrencia, *pipeline*, deshabilitación parcial del reloj o utilizar operaciones de menor consumo tienen una influencia algo más indirecta. Esta sección evalúa éstas fuentes de ahorro de consumo.

2.7.1. Algoritmos para bajo consumo.

Tres factores primarios afectan directamente a la potencia consumida por un algoritmo independientemente de la arquitectura que se elija. Ellos son la complejidad, la precisión y la regularidad del algoritmo elegido.

Complejidad

La complejidad de un algoritmo puede ser medido de diferentes maneras. Una primera métrica de la complejidad puede ser la cantidad de instrucciones u operaciones realizadas. Dado que cada operación consume potencia, es un indicador útil para medir o comparar las características de bajo consumo de un algoritmo respecto de otro para una arquitectura determinada.

Hay que recordar en realidad que no todas las operaciones consumen lo mismo, claramente una multiplicación o división han de consumir mucho mas que una adición o substracción. Para una correcta comparación no solo se debe medir el total de operaciones, sino también ponderar con el tipo de operaciones realizadas.

Relacionado con esto, otra métrica para evaluar la complejidad de los algoritmos es la cantidad de datos accedidos en memoria. Los accesos a registros y memoria (sobre todo memorias externas) suelen ser caros en término de consumo. Por ello, algoritmos que no solo minimizan la cantidad de operaciones, sino que además la cantidad de accesos a memoria y su tamaño son mas apropiados para implementaciones de bajo consumo.

Precisión

Como es de suponer, arquitecturas con mayores anchos de palabras consumen más potencia. Por tanto una arquitectura debe utilizar solo el ancho de palabra necesario para cumplir con los requerimientos de precisión del algoritmo. Existen diferentes algoritmos que computan la misma función con requerimientos de anchos de palabra internos totalmente diferentes.

Existe en la bibliografía un concepto relacionado llamado procesado de señal aproximado (*aproximate signal processing*) [Ben99c], donde un determinado nivel de ruido puede ser tolerado. La idea central es que se puede reducir drásticamente el consumo permitiendo algunas imprecisiones en el cálculo.

Regularidad

Mientras la complejidad computacional afecta a la potencia consumida por la ruta de datos, la regularidad de un algoritmo puede afectar la complejidad y el consumo del hardware de control y de la red de interconexión. En FPGAs o ASICs un algoritmo

regular requiere menos estados para describir su comportamiento, lo que se traduce en una maquina de estados con menos consumo. En caso de procesadores o microcontroladores conlleva a desperdiciar menos consumo en saltos en el programa (cada salto es una operación que no computa nada efectivo y genera consumo, el que se ve agravado si el procesador utiliza *pipeline*, donde además se ha de descargar toda la "tubería" sin generar computo)

Más aun los algoritmos regulares tienden a tener patrones de comunicación más regulares también, lo que facilita las redes de interconexión. Este es un punto importante dado que las redes de interconexión globales aportan una nada despreciable parte del consumo total.

2.7.2 Algoritmos para arquitecturas de bajo consumo

Anteriormente se reconocían el efecto directo sobre el consumo que poseen la complejidad, la precisión y la regularidad de los algoritmos. Otro aspecto importante, aunque más sutil, es reconocer cuan bien se adecua un algoritmo a una arquitectura de bajo consumo (como las explicadas en las sección 2.5 y 2.6). Para una eficiente implementación de los algoritmos en arquitecturas de bajo consumo, existen características deseables como son la concurrencia y la modularidad.

Concurrencia

Para sistemas donde se puede cambiar la tensión de alimentación sin otros problemas, se puede utilizar la concurrencia para aumentar el rendimiento y de ese modo reducir la tensión de alimentación y consecuentemente el consumo. Esta estrategia carece de sentido si el algoritmo no posee la suficiente concurrencia. Los cuellos de botella suelen ser las sentencias u operaciones secuénciales o recursivas. Es decir aquellas que dependen de resultados de operaciones previas para poder completarse. Por ejemplo, en el cálculo del producto utilizando sumas y desplazamientos (ecuación 2.5), se requiere conocer siempre el resultado anterior haciendo imposible (sin modificar el algoritmo) cualquier tipo de paralelización.

$$Pi+1 := (Pi*2 + X(n-i)*Y) \qquad \text{(Ec. 2.5)}$$

Existen varias técnicas software para eliminar los cuellos de botella recursivos, además hay que tener en cuenta las técnicas de replicación y eliminación de subexpresiones

comunes, retemporización (*retiming*) y las transformaciones algebraicas como posible caminos para lograr un mayor grado de concurrencia.

Modularidad y localidad

Otra técnica arquitectural de bajo consumo es explotar la localidad a través de procesadores, memorias y control distribuidos. Maximizando la modularidad del algoritmo se asegura el uso eficiente de la estructura distribuida de procesamiento. En particular los grafos de computo con grupos de computación fuertemente conexos con relativas pocas conexiones globales es de esperar que puedan ser eficientemente implementadas en arquitecturas distribuidas.

Otro efecto positivo de la modularidad es la facilidad de generar estructuras con *pipeline*, las que como fue señalado tienden a reducir el consumo por reducción de la actividad espuria (*glitches*).

2.8. Nivel sistema

Considerando a un sistema formado por elementos de proceso, memorias y recursos de comunicación, la optimización desde el punto de vista del consumo de dicha interacción es abordado en este apartado. Se consideran dentro de este nivel la optimización de memoria, el particionado del sistema y las técnicas de manejo de potencia (*power management*).

2.8.1 Optimización de memoria

Estas técnicas intentan reducir el consumo tanto en el uso de memoria como en la comunicación y transferencia de datos. Muchas aproximaciones se centran en explotar sistemas de *caches* para reducir el consumo [Su94][Baj97]. Otras técnicas buscan jerarquías mas complejas de diferentes tipos de memoria (SRAMs, DRAMs, etc.) controlando la transferencia entre módulos y el emplazado [Lid94][Geb97] [Lee95].

2.8.2 Particionado hardware-software

Es un concepto relacionado con el codiseño hardware software (*hardware software codesign*) orientado al bajo consumo. A partir de una descripción funcional de alto

nivel, estas técnicas intentan realizar una partición y asignación de las tareas en hardware y software de forma óptima [Wan98][Hwa99][Hen99][Mah01]. El mayor desafió en este tema es contar con potentes herramientas de estimación de consumo, ya que sin una estimación eficaz es imposible optimizar posteriormente.

2.8.3 Optimización a nivel de instrucciones

El enfoque de las optimizaciones a nivel de instrucciones es en general sobre sistemas de procesadores estándares. Estos métodos están basados en modelos del consumo de los procesadores, en los que cada par de instrucciones tiene asociado un costo. La optimización, por tanto se basa en seleccionar una combinación de instrucciones con mínimo consumo [Tiw94b] [Tiw96a] [Tiw97] [Dou98] [Tob98] [Tiw98].

2.8.4 Manejo dinámico del consumo

Si bien estas ideas fueron descritas parcialmente en el punto 2.6.2, aquí se clasifican ideas similares pero desde el punto de vista del sistema y no de un circuito en particular. El manejo dinámico del consumo (*Dynamic Power Management* – DPM) es un método de control que permite a los sistemas, o parte de ellos de ser en estados dormidos (*sleep mode*) cuando están inactivos. Existen actualmente estándares de manejo de energía en ordenadores y dispositivos portátiles [acp02] y prácticamente todos los procesadores modernos tienen algún método de control del consumo. La misma tendencia se observa en diferentes componentes de sistemas como en discos rígidos, módulos de comunicación, pantallas, etc.

2.8.5 Minimización del consumo en las interfaces

Una gran cantidad de energía es consumida en la comunicación de datos sobre buses muy cargados dentro o fuera del chip. Diferentes aproximaciones se han desarrollado para reducir las conmutaciones en los buses vía recodificación de la señal [Sta95b] [Sta94][Sta95a][Pan96][Sta96][Sta97][Ben98].

2.9. Otros conceptos en el tratamiento del consumo

En esta sección se presentan las definiciones de energía y potencia (conceptos que regularmente son confundidos) y alguna de las métricas que se utilizan para su medida. Por otro lado se presentan las proyecciones para la tecnología de semiconductores elaborado por la *International Semiconductor Industry Association* que refuerzan el los argumentos a favor del interés en el desarrollo de técnicas de reducción de consumo.

2.9.1. Energía vs. potencia.

La potencia consumida se expresa en Vatios (*Watts*) y determina el diseño de la fuente de alimentación, de los reguladores de tensión y las dimensiones de las interconexiones, eventualmente la refrigeración de corto tiempo.

Por otra parte, la energía consumida se expresa en julios (*joules*) e indica la potencia consumida en el tiempo, como lo muestra la ecuación 2.6. La energía se asocia con la duración de las baterías. De esta manera menor energía indica menos potencia para realizar un cálculo a la misma frecuencia. La figura 2.15 muestra gráficamente esta ecuación.

$$\text{Energy} = \text{power} * \text{delay (joules} = \text{watts} * \text{seconds)} \qquad \text{(Ec. 2.6)}$$

La energía es por tanto independiente de la frecuencia de reloj. Reducir la velocidad del reloj solo degradará la performance, pero no logrará ahorros en la vida útil de la batería (a menos que se cambie el tensión de alimentación).

Métricas utilizadas: Las principales métricas para el consumo en circuitos integrados utilizadas en la literatura son:

- Potencia-Tiempo (*Power-Delay Product* - PDP) PDP = $P_{avg} * t$. Mide la energía consumida por cada evento de conmutación. También se expresa como P_{avg}/frecuencia, típicamente en mW/MHz.
- Métrica para la eficiencia de la energía [Bur95].

  $$\text{ETR} = (\text{Energy/operation})/\text{Throughput} = \text{Power}/\text{Throughput}^2 \qquad \text{(Ec. 2.7)}$$

  Un menor ETR indica menor energía para un rendimiento constante, o mejor performance para la misma energía.

- Operaciones por unidad de Potencia. Utilizado en el ámbito de los microprocesadores. Típicamente en MOPS/mW o MIPS/mW.

En esta tesis se expresarán los consumos en mW/MHz cuando se comparan circuitos y algoritmos que producen igual cantidad de datos por ciclo de reloj. Por otra parte se utiliza nJoules cuando se compara el consumo total de diferentes alternativas que requieren diferente cantidad de ciclos de reloj para realizar la computación.

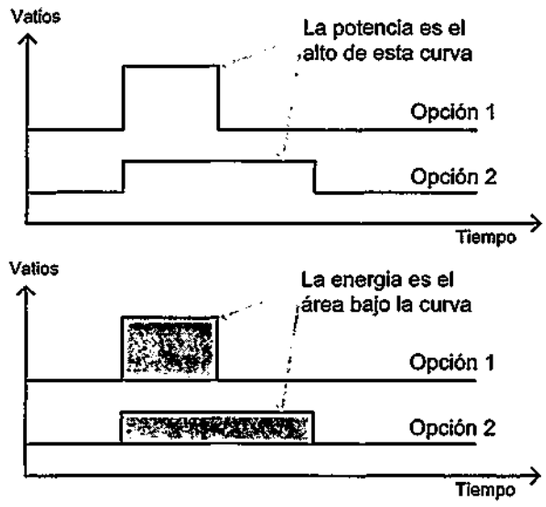


Figura 2.15. Gráfica del consumo y la energía

2.9.2. Proyecciones para la tecnología de semiconductores

La asociación norteamericana de la industria de semiconductores [SIA03] y la asociación internacional de la industria de semiconductores [ITR03] realizan periódicamente proyecciones de la evolución de la industria (*Semiconductor Industry Association Roadmap*) de semiconductores. En apoyado en los informes previos provistos en [Jew00][SIA97], y en el más reciente de las ITRS (*Internacional Technology Roadmap for Semiconductors*)[ITR03] se muestra la tabla con las proyecciones para los próximos años.

Elementos	*1997*	*2001*	*2003*	*2006*	*2009*	*2012*
Ancho de las líneas (micrones)						
Líneas en DRAM	0.25	0.15	0.13	0.10	0.07	0.05
Líneas en Lógica	0.20	0.12	0.10	0.07	0.05	0.035
Memorias						
Bits/chip (DRAM)	256M	1G	4G	16G	64G	256G
Bits/cm^2 (DRAM)	96M	380M	770M	2.2G	6.1G	17G
Cost/bit (packaged - microcents)	120	30	15	5.3	1.9	0.66
Lógica						
High volume MPU (trans/cm^2)	3.7M	10M	18M	39M	84M	180M
Low volume ASIC (trans/cm^2)	8M	16M	24M	40M	64M	100M
Memorias						
Chip-to-package pads (cost performance)	800	1195	1460	1970	2655	3585
Chip-to-package pads (high performance)	1450	2400	3000	4000	5400	7300
Frequencias de los Chips (MHz)						
On-chip (local), high performance	750	1500	2100	3500	6000	10000
On-chip (across-chip), high performance	750	1400	1600	2000	2500	3000
On-chip (across-chip), cost performance	400	700	800	1100	1400	1800
On-chip (across-chip), ASIC high perform.	300	600	700	900	1200	1500
Chip-to-board, high performance	750	1400	1600	2000	2500	3000
Tamaños de los Chip (mm^2)						
DRAM	280	445	560	790	1120	1580
Microprocessor	300	385	430	520	620	750
ASIC	280	445	560	790	1120	1580
Proceso de Fabricación						
Nro. de niveles cableados on-chip	6	7	7	7-8	8-9	9
Nro. mínimo de mascaras	22	23	24	24-26	24-26	28
Max. diámetro de los "Wafer" (mm)	200	300	300	300	450	450
Tensión (V) y Potencia (watts)						
Tensión de alimentación de la lógica. Vdd.	1.8 - 2.5	1.2 - 1.5	1.2 - 1.5	0.9 - 1.2	0.6 - 0.9	0.5 - 0.6
Potencia (alta perf. con refrigeración)	70	110	130	160	170	175
Potencia (a baterías y handheld)	1.2	1.7	2.0	2.4	2.8	3.2

Figura 2.16. Proyecciones para la industria de semiconductores.

2.10 Desglose del consumo en FPGAs

Resulta importante conocer la influencia detallada de los diferentes componentes de la FPGA en el consumo. Esto permite concentrarse en los puntos importantes a la hora de reducir el consumo. El análisis del consumo en FPGAs ha tenido poca importancia respecto de las *standart cell*. No obstante, en [Kus98][Gar00] se presentan resultados del desglose del consumo para la familia XC4K, en tanto en [Sha02][Poo02] se realiza lo propio en Virtex II.

En todos los trabajos existe la coincidencia en cuanto a la alta influencia de la interconexión. Solo [Gar00] estima el consumo de la interconexión por debajo del 50%, en tanto, [Kus98] lo sitúa en un 65%. Para la familia de dispositivos Virtex II [Sha02] lo calcula en torno al 60 % en tanto que [Poo02] en el 57 %.

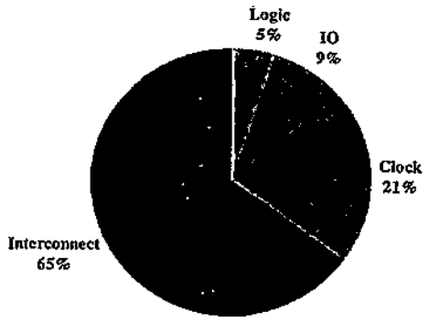


Figura 2.17. Distribución del consumo en XC4003 según [Kus98]

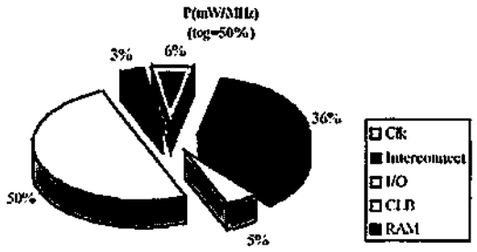


Figura 2.18. Distribución del consumo en XC4k-E según [Gar00].

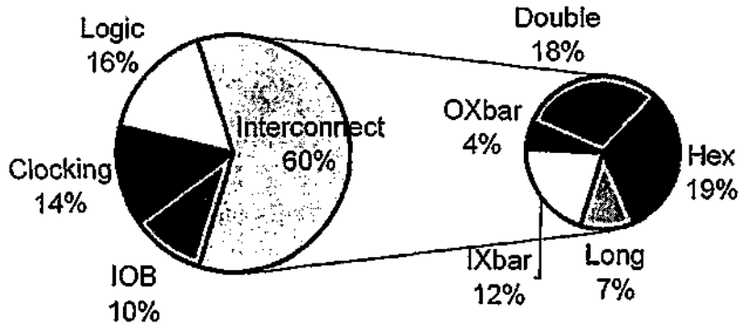


Figura 2.19. Distribución del consumo en Virtex 2 según [Sha02]

2.11 Resumen del capítulo

Este capítulo brinda una introducción al diseño de bajo consumo en CMOS y su aplicación al marco tecnológico de los circuitos reprogramables. Comienza con una introducción al consumo en los circuitos CMOS en general, siguiendo por un repaso de los temas recurrentes en el diseño de bajo consumo. Se hace un repaso de las principales técnicas de reducción de consumo en los diferentes niveles de abstracción (nivel de sistema, de algoritmos, arquitectura, diseño del Circuito, proceso y tecnología)

En la discusión precedente queda claro que la principal componente del consumo en los circuitos CMOS es debido a la carga y descarga de las capacidades parásitas, obteniéndose la conocida expresión del consumo dinámico:

$$P = \frac{1}{2} \cdot C_L \cdot V_{dd}^{\ 2} \cdot E(sw) \cdot f_{clk}$$

La reducción del consumo se puede obtener reduciendo la capacidad C_L la actividad $E(sw).f_{clk}$ o la tensión de alimentación V_{dd}. Las técnicas reseñadas durante el capítulo tratan de reducir estos factores siguiendo ciertos temas recurrentes como el balance área y velocidad para reducir consumo, evitar derroches, aprovechar la localidad de los datos. Varias técnicas utilizan el aumento de performance, para luego reducir la tensión de alimentación y aprovecharse así de la dependencia cuadrática de la tensión en el consumo.

Por otro lado queda claro que las mayores reducciones de consumo se logran cuanto mayor sea el nivel de abstracción. Mientras a bajo nivel (nivel de puertas, rutado y emplazado y tecnológico) como mucho se puede lograr reducciones en un factor de dos, optimizaciones a nivel de arquitectura, algoritmo o sistema pueden llegar a reducir ordenes de magnitud el consumo de potencia.

Por último en este capítulo se presentan algunos conceptos claves en el tratamiento de consumo, así como el desgloce del consumo en FPGAs realizado por diferentes autores. Todos coinciden en la gran influencia del rutado en el total del consumo, lo que brinda un punto importante a la hora de reducir el consumo.

2.12 Referencias del capítulo

[Acp02] Compaq, Intel, Microsoft, Phoenix, Toshiba, "ACPI Advanced Configuration & Power Interface", Specification Revision 2.0b, October 11, 2002. http://www.acpi.info/

[Alv98] Atila Alvandpour, Per Larsson-Edefors, Christer Svensson, "Separation and extraction of short-circuit power consumption in digital CMOS VLSI circuits", *International Symposium on Low Power Electronics and Design archive Pp 245 - 249*, Monterey, California, United States, 1998.

[And04] J. H. Anderson, F. N. Najm, T. Tuan, "Active Leakage Power Optimization for FPGAs" *Twelfth International Symposium on Field Programmable Gate Arrays (FPGA 2004)* pp.33 - 41, Monterrey, California, Feb. 2004

[Baj97] R. S. Bajwa, M. Hiraki, H. Kojima, D. J. Gorny, K. Nitta, A. Shridhar, K. Seki, and K. Sasaki, "Instruction Buffering to Reduce Power in Processors for Signal Processing" *IEEE TVLSI Systems, Volume 05, Number 04, p. 417*, December 1997.

[Ben98] Luca Benini, Giovanni De Micheli, Enrico Macii, Donatella Sciuto, and Cristina Silvano; "Address Bus Encoding Techniques for System-Level Power Optimization"; *Proceedings of Design, Automation and Test in Europe (DATE'98)*, Paris, France. Feb 1998.

[Ben99c] Luca Benini and Giovanni De Micheli, "System-Level Power Optimization: Techniques and Tools", *International Symposium on Low Power Desing (ISLPED99)*, San Diego, CA, USA 1999.

[Boe95] E. Boemo, G. Gonzalez de Rivera, S.Lopez-Buedo and J. Meneses, "Some Notes on Power Management on FPGAs", *Lecture Notes in Computer Sciences (LNCS), No.975, pp.149-157.* Berlin: Springer-Verlag 1995.

[Boe96] E. Boemo, "Contribution to the Design of Fine-grain Pipelined VLSI Arrays", *Ph.D. Thesis, ETSI Telecomunicación, Universidad Politécnica de Madrid*, January 1996.

[Boe98] E. Boemo, S. Lopez-Buedo, C. Santos, J. Jauregui and J. Meneses, "Logic Depth and Power Consumption: A Comparative Study Between Standard Cells and FPGAs", *Proc. XIII DCIS Conference (Design of Circuit and Integrated Systems)*, Madrid, Universidad Carlos III: November 1998.

[Bur95] T. Burd, R. Brodersen, "Energy Efficient CMOS Microprocessor Design," *Proceedings of the 28th Annual HICSS Conference, Vol. I, pp. 288-297* January 1995.

[Cha94b] A. Chandrakasan, R. Allmon, A. Stratakos, and R. W. Brodersen, "Design of Portable Systems", *Proceedings of Custom Integrated Circuits Conference (CICC '94)*, San Diego, May 1994.

[Dou98] William E. Dougherty, David J. Pursley, Donald E. Thomas, "Instruction Subsetting: Trading Power for Programmability", *IEEE Workshop on VLSI 1998*, Los Alamitos, CA, pp.42-47, 1998.

[Gar00] A. Garcia, "Power consumption and optimization in field programmable gate arrays," *Ph.D. thesis, Département Communications et Électronique, Ecole Nationale Supérieure des Télécommunications*, 2000.

[Gay04] A. Gayasen, Y. Tsai, N. Vijaykrishnan,M. Kandemir, M.J. Irwin, T. Tuan, "Reducing Leakage Energy in FPGAs Using Region-Constrained Placement", *Twelfth International Symposium on Field Programmable Gate Arrays (FPGA'04)*, Monterey, California, USA. February, 2004.

[Geb97] C. Gebotys, "Low Energy Memory and Register Allocation Using Network Flow", *ACM/IEEE Design and Automation Conference (DAC '97)*, Anaheim, California. 1997.

[Hen99] Jörg Henkel, "A Low Power Hardware/Software Partitioning Approach for Core-Based Embedded Systems". *Design Automation Conference (DAC '99), pp 122-127*, June 1999.

[Hwa99] Enoch Oi-Kee Hwang, "Functional Partitioning for Low Power". *PhD thesis in computer science at University of California at Riverside.* June 1999.

[Int00] Intel corp., "Mobile Intel Pentium III with SpeedStep". *Intel web page Second Quarter 2000*, www.intel.com. 2000.

[ITR03] ITRS Technology Working Groups. "International Technology Roadmap for Semiconductors 2003 Edition: Executive Summary". Jointly Sponsored by European SIA, Japan EITIA, Korea SIA, Taiwan SIA, and Semiconductor Industry Association (SIA), http://public.itrs.net/; 2003.

[Jew00] Jim Jewett, "The International Technology Roadmap for Semiconductors" *(ESH THRUST) Intel Corporation Arlington*, VA; April 2000.

[Jun01] Seung-Ho Jung, Jong-Humn Baek, and Seok-Yoon Kim, "Short circuit power estimation of static CMOS circuits", *Proceedings of the 2001 conference on Asia South Pacific design automation*, January 2001.

[Kao02] J. Kao, S. Narendra, and A. Chandrakasan. Subthreshold leakage modeling and reduction techniques. *In IEEE International Conference on Computer-Aided Design*, pages 141–148, 2002.

[Kus98] E. A. Kusse, and J. Rabaey, "Low-energy embedded FPGA structures," *International Symposium on Low Power Electronics & Design, pp.155-160*, August 1998.

[Lee95] T. C. Lee and V. Tiwari, "Memory Allocation Technique for Low- Energy Embedded DSP Software". Proceedings of the *1995 IEEE Symposium on Low Power Electronics*, San Diego, CA, October 1995.

[Lid94] David B. Lidsky, Jan M. Rabaey, "Low-power design of memory intensive functions Case Study: Vector Quantization". *In Proceedings of the IEEE Symposium on Low Power Electronics.* Sept., 1994.

[Liu04] Michael Liu, Wei-Shen Wang, and Michael Orshansky, "Leakage Power Reduction by Dual-Vth Designs Under Probabilistic Analysis of Vth Variation" *International Symposium on Low Power Desing (ISLPED'04)*, Newport Beach, California, USA. August, 2004.

[Mee95] J. Leijten, J. Meerbengen, and J. Jess, "Analysis of Reduction of Glitches in Synchronous Networks", *Proceedings of IEEE European Design and Automation Conf (EDAC95)*, IEEE Press, 1995.

[Mah01] Rabi Mahapatra, "Hardware-Software Codesign: Issues & Challenges", *Seminar at Department of Computer Science Texas A&M University.* September 2001.

[Mar95] R. Marculescu, D. Marculescu and M. Pedram, "Efficient Power Estimation for Highly Correlated Input Streams", *Proceeding of 32th Design and Automation Conference (DAC).* 1995.

[Nar03] S. Narendra, D. Blaauw, A. Devgan, F. Najm, "Leakage issues in IC design: trends, estimation, and avoidance," *Proc. of International Conference on Computer Aided Design (ICCAD)*, 2003.

[Nec98] NEC Electronics Inc, "Power Management Modes In the VR4100 Family of 64-Bit MIPS RISC Microprocessors", *NEC Application Note*, July 1998

[Pan96] Panda, P.R. and N.D. Dutt. Reducing Address Bus Transitions for Low Power Memory Mapping. *In Proc. of EDTC-96: IEEE European Design and Test Conference*, Paris - France, pp. 63-67, March 1996

[Ped96] Massoud Pedram, "Tutorial and Survey Paper - Power Minimization in IC Design: Principles and Applications" *ACM Transaction on Design Automation of Electronic Systems, Vol 1, No 1*, Jan 1996

[Poo02] K. Poon, A. Yan, and S. J. E. Wilton. "A Flexible power model for FPGAs". *In Intertnational Conference on Field-Programmable Logic and Applications (FPL02), pages 312–321*, La Grande Motte, France, 2002.

[Rab96a] Rabaey, Jan M. "Low power design methodologies", *Kluwer Academic publishers*, 1996.

[Rab96b] Dirk Rabe, Wolfgang Nebel, "Short Circuit Power Consumption of Glitches", *In proc. of International Symposium on Low Power Desing (ISLPED 96)* Monterey CA USA. 1996

[Sha02] Li Shang, Alireza Kaviani, Kusuma Bathala, "Dynamic Power Consumption in Virtex™-II FPGA Family", *FPGA'02*, Monterey, California, USA. February 2002.

[She92] Amelia Shen, Abhijit Ghosh, Srinivas Devadas, and Kurt Keutzer. "On average power dissipation and random pattern testability of CMOS combinational logic networks". *In IEEE Int. Conf. on Computer-Aided Design, pages 402–407*, 1992.

[SIA03] Semiconductor Industry Association (SIA) home page, http://www.semichips.org/

[SIA97] SIA Semiconductor Industry Association, "The National Technologies Roadmap for semiconductors: Technologies Needs", 1997.

[Sta94] Mircea R. Stan; Wayne P. Burleson, "Limited-weight codes for low-power I/O", *International Workshop on Low Power Design*, April 1994.

[Sta95a] Mircea R. Stan; Wayne Burleson, "Bus-Invert Coding for Low-Power I/O", *IEEE Transaction on VLSI Systems, volume 3, number 1, pages 49-58.* March 1995.

[Sta95b] Mircea R. Stan; Wayne P. Burleson, "Coding a Terminated Bus for Low Power", *Proceedings of Great Lakes Symposium on VLSI, Buffalo, NY, USA, pp. 703.* March, 1995.

[Sta96] Mircea R. Stan; Wayne P. Burleson, "Two-Dimensional Codes for Low Power", *In proc. of International Symposium on Low Power Desing (ISLPED'96),* Monterey CA USA. 1996.

[Sta97] M. Stan, W. Burlesson, "Low-Power Encodings for Global Communication in CMOS VLSI", *IEEE TVLSI Systems, Volume 05, Number 04, pp. 444,* December 1997.

[Su94] C. Su, C. Tsui y A. Despain, "Low Power Architecture Design and Compilation Techniques for High-Performance Processors", *Proc. IEEE 1994 Spring COMPCON, pp.489-498.* IEEE Press, 1994.

[Tho98] Scott Thompson, Paul Packan and Mark Bohr. "MOS Scaling: Transistor Challenges for the 21st Century", *Intel Technology Journal Q3'98,* 1998.

[Tiw94b] V. Tiwari, P. Ashar and S. Malik, "Compilation Techniques for Low Energy: An Overview", IEEE Solid States Council Symposium on Low Power Electronics, 1994.

[Tiw96a] V. Tiwari, S. Malik, A. Wolfe and M. Lee, "Instruction Level Power Analysis and Optimization of Software", Journal of VLSI Signal Processing, Kluwer Academic Publishers 1996.

[Tiw97] V. Tiwari, R. Donnelley, S. Malik and R. Gonzalez, "Dynamic Power Management for Microprocessors: A Case Study", IEEE VLSI Design, January 1997.

[Tiw98] Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel and Franklin Baez, "Reducing Power in High-performance Microprocessors" *In Proceedings of 35th Design Automation Conference (DAC '98)* San Francisco, CA USA. June 1998.

[Tob98] M. C. Toburen, T. M. Conte, and M. Reilly. "Instruction scheduling for low power processors". *In Proceedings of the Power Driven Micro-architecture. Workshop in conjunction with the ISCA'98*, June 1998.

[Vee84] H. J. M. Veendrick, "Short-Circuit Dissipation of Static CMOS Circuitry and its Impact on the Design of Buffer Circuits," *IEEE Journal of Solid-State Circuits*, pp. 468-473, August 1984.

[Wan98] Marlene Wan, Yuji Ichikawa, David Lidsky, Jan Rabaey, "An Energy Conscious Methodology for Early Design Exploration of Heterogeneous DSPS" *Proceedings of the Custom Intergrated Circuit Conference* , Santa Clara, CA, USA, May 1998.

Capítulo 3:

Experimentos sobre Bajo Consumo

Este capítulo examina diferentes experimentos llevados cabo sobre el consumo en FPGAs con el objeto de determinar relaciones básicas, y de ser posible derivar recomendaciones generales a nivel usuario. Otro objetivo implícito es demostrar que la utilización de herramientas de desarrollo puede servir como métricas indirectas para estimar o reducir el consumo. Comienza con una serie de experimentos básicos donde se establece la relación de la tensión y la frecuencia en el consumo, para proseguir con la influencia de la corriente estática y la corriente de sincronización. Más adelante se examinan la relación velocidad-consumo, la conmutatividad de datos, el efecto de la segmentación (*pipeline*) y el uso de registros en general. Finalmente se analizan y cuantifican las alternativas para la inhabilitación de partes inactivas de un circuito.

3.1 Experimentos generales

En el capítulo 2 se han enumerado las causas del consumo y establecido las formulas correspondientes, en esta sección se realizan experimentos para corroborar la linealidad con la frecuencia, así como la dependencia cuadrática con la tensión. En [Gar00] se sugiere la existencia de una componente V_{dd} de tercer orden en el consumo debido a las corrientes de cortocircuito, aunque en las medidas realizadas en esta tesis no conducen a dichos resultados. Por otra parte se ha medido la corriente estática en los diferentes dispositivos y la influencia del consumo de sincronización (debida al árbol de distribución del reloj).

Para cada familia de dispositivos se utilizan diferentes grupos de circuitos de prueba atendiendo a las características de densidad de cada uno. El objetivo de estos experimentos generales es, en primer lugar, corroborar las relaciones básicas de consumo y en segundo termino comparar la evolución relativa de las componentes estática y de sincronización en el consumo total.

3.1.1 Relación de la frecuencia y tensión en el consumo

Dispositivo XC4K

Utilizando el arreglo experimental descrito en el Apéndice A, se han medido la relación del consumo con la frecuencia y la tensión de alimentación. Los circuitos presentes en la Figura 3.1 son los multiplicadores modulares *mult_red, shift&add* y *montg* descritos en la sección 5.1, y los circuitos de multiplicación entera *guild, hatamian y wallace* de la sección 3.2. Las medidas se llevaron a cabo sobre el dispositivo XC4010PC84-4C. Los análisis de regresión dan un coeficiente de determinación (R^2) superior a 0,98 en todos los casos.

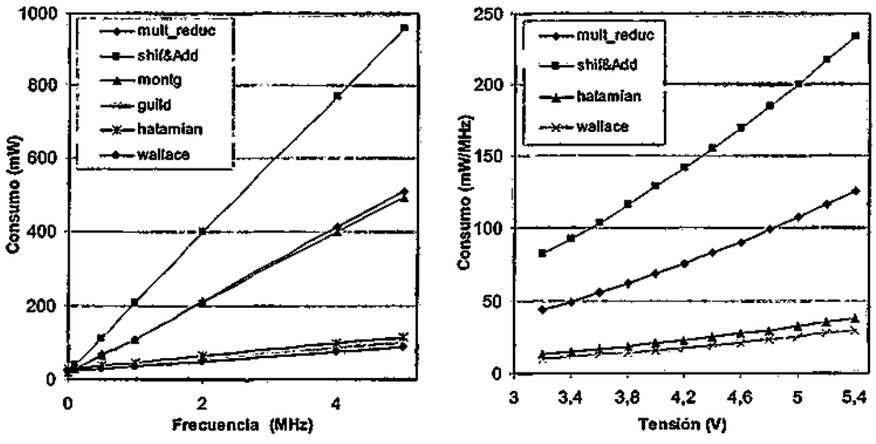


Figura 3.1. Medidas de la relación lineal de la frecuencia con el consumo y cuadrática con la tensión de alimentación en la familia XC4K.

Dispositivo Virtex

Se han medido la relación del consumo con la frecuencia y la tensión de alimentación para Virtex utilizando el arreglo experimental del Apéndice B. Se muestran en las figura Figura 3.2 el consumo de corriente del *core* (Vccint = 2,5V). Los circuitos utilizados en la son los algoritmos de división de números fraccionarios descritos en la sección 5.4. La Figura 3.2.a utiliza divisores de 16 bits y fueron medidos con la secuencia de actividad aleatoria. Por otra para la Figura 3.2.b utiliza divisores de 32 bits con secuencia de entrada de máxima actividad. En este caso, al igual que en la familia XC4K los análisis de regresión dan un coeficiente de determinación (R^2) superior a 0,98 para todos los circuitos medidos.

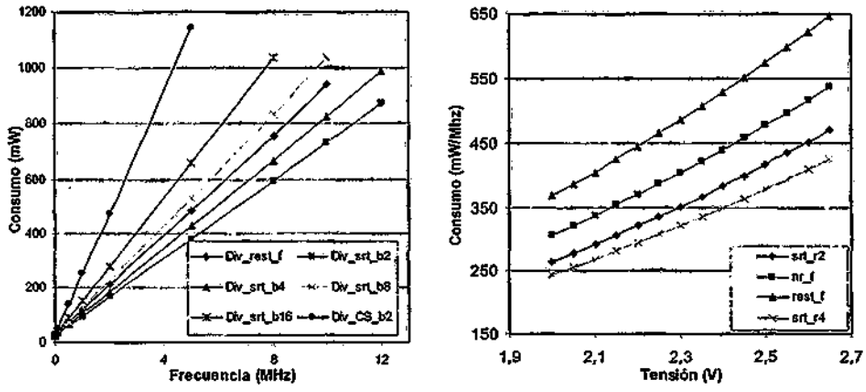


Figura 3.2. Medidas de la relación lineal de la frecuencia y cuadrática con la tensión en dispositivos de la familia Virtex.

Dispositivo Virtex II

Utilizando el arreglo experimental del Apéndice C, se han medido la relación del consumo con la frecuencia y la tensión de alimentación. Las medidas corresponden a la corriente Vccint de 1.5 V correspondiente al *core*. Los circuitos medidos son multiplicadores tipos *shift&add* de 16 bits (sección 3.2) y divisores de números enteros. Para la medición de la relación de la tensión (Vccint) se vario entre 1,25 y 1,65 obteniéndose una dependencia cuadrática con coeficiente de determinación (R^2) superior a 0,97. La linealidad con la frecuencia en el rango 10 Hz – 100 MHz brindo un R2 mayor a 0,99 para todos los circuitos medidos.

3.1.2 Corriente Estática

Conforme avanzan los procesos tecnológicos la influencia de la componente estática del consumo (sección 2.1.1) comienza a ser más importante [Nar03][Liu04] llegando a representar en diseños con procesos de fabricación por debajo 100 *nm* hasta un 40 % del consumo total. Por cuestiones tecnológicas la corriente estática varía con la configuración de la FPGA. Existe un interesante trabajo en la reducción de las corrientes estática en las nuevas familias de FPGA en [And04] basado en las ideas de [Kao02] que muestra una reducción media de la corriente estática de hasta un 25%.

En esta sección se muestran los valores medidos para la corriente estática en los diferentes dispositivos y su influencia relativa sobre el consumo total. La Tabla 3.1 muestra el consumo de corriente estática para los tres dispositivos de la familia XC4K utilizados en esta tesis (Apéndice A), el dispositivo Virtex (Apéndice B) y Virtex II (Apéndice C). Se muestra la media y la desviación estándar en función de los circuitos medidos.

Cabe destacar que los valores de corriente estática medidos en el dispositivo Virtex II es mucho menor que el informado por Xilinx en la herramienta de estimación de consumo Xpower [Xpo04] (apéndice F) y por la utilidad de estimación de consumo on-line de Xilinx [Xil04a]. Según estas herramientas el consumo estático en estos dispositivos llega aproximadamente al medio Vatio.

Dispositivo	Cantidad Circuitos	Corriente estática	Desv. Estándar	Tension Alimentación	Potencia Total (mW)
XC4003PC84-4	10	4,29	0,03	5	21,4
XC4005PC84-4	10	4,60	0,10	5	23,0
XC4010PC84-4	25	4,68	0,38	5	23,4
Virtex XCV800					
Vccint	50	10,07	0,40	2,5	25,2
Vccout	50	1,81	0,07	3,3	6,0
Virtex II XC2V1500					
Vccint	20	10,35	0,25	1,5	15,5
Vccout	20	11,20	0,15	3,3	37,0
Vccaux	20	2,70	0,09	3,3	8,9

Tabla 3.1. Consumo estático para dispositivos XC4K, Virtex y Virtex II medidos.

Medir el impacto del consumo estático sobre el total es complejo de llevar a cabo dado que se comparan dispositivos con densidades muy dispares. No obstante se lleva a cabo una primera aproximación escogiendo circuitos combinacionales de gran actividad como multiplicadores y divisores. La Tabla 3.2 muestra la relación porcentual del consumo estático respecto del total para tres ejemplos en cada tecnología, solo se calcula sobre el consumo en el *core*. Para el caso de XC4K se presentan tres multiplicadores de enteros de 8 bits (ver sección 4.3.2), en el dispositivo XC4010PC84-4 (matriz de 20 × 20 CLBs). Para Virtex (112 × 84 slices) y Virtex II (96 × 80 slices), se presentan los consumos para un multiplicador "*shift and add*" (sección 3.4), un divisor sin restauración (*non restoring*) y un divisor SRT base 2 (sección 5.4.4).

Para cada circuito se informa, la ocupación medido en *slices* (CLB en XC4K), el porcentaje del total de lógica que utiliza, el consumo en mA/MHz y el consumo en mW para cierta frecuencia. Por último se expresa la relación consumo estático respecto del total.

	Circuito	Área (slices)	Ocupa-ción	Consumo mA/MHz	Frec. (MHz)	Tension (V)	Consumo (mW)	Estático / Total
XC4000	Guild comb	60	15%	6,0	5	5,0	149,4	13,6%
	m-vhdl8	56	14%	3,9	5	5,0	98,1	19,3%
	corem8	52	13%	3,4	5	5,0	84,6	21,7%
Virtex	mult_S&A	545	6%	76,0	2	2,5	380,2	6,6%
	div_nr	576	6%	169,8	1	2,5	424,4	5,9%
	srt_r2	561	6%	135,1	1	2,5	337,8	7,3%
Virtex II	mult_S&A	585	8%	38,2	2	1,5	114,5	11,9%
	div_nr	576	8%	28,4	1	1,5	42,6	26,7%
	srt_r2	561	7%	33,2	1	1,5	49,8	23,8%

Tabla 3.2. Relación consumo estático respecto del dinámico en XC4K, Virtex y Virtex II.

La Tabla 3.2 permite percibir la disminución relativa del consumo estático que se evidencia en Virtex respecto de XC4K, y el gran aumento relativo que representa esta componente en Virtex II. El consumo dinámico en Virtex II se reduce notablemente respecto de Virtex, en un factor que va desde 3 hasta casi 10.

3.1.3 Corriente de Sincronización

La corriente de sincronización es la que se origina por el árbol de distribución del reloj y los registros conectados a ellos. En la Figura 3.3 se puede ver la relación de la corriente de sincronización respecto de la cantidad de registros para dispositivos de la familia XC4K. El análisis de regresión de los datos nos brinda un consumo debído del *buffer* y árbol de reloj de 7,3 mW/MHz y de 0,03 mW/MHz por cada FF extra.

La Figura 3.4 muestra la relación consumo de sincronización respecto del de la ruta de datos para algunos circuitos en XC4K. Se han graficado dos multiplicadores con diferente profundidad lógica. Uno totalmente combinacional, solo registrado a la entrada y salida (*Hatam-15*) y otro con profundidad lógica de tres LUTs (*Hatam-3*). Por otro lado se han graficado tres algoritmos de multiplicación modular implememtados sequencuialmente, es decir con una ruta de datos y una máquina de estados que lo controla (*mult_red*, *S&A* y *Montg*). En la Figura 3.27 de la sección 3.4.1 se muestra además la relación de la corriente de sincronización respecto de la ruta de datos en la segmentación (*pipeline*).

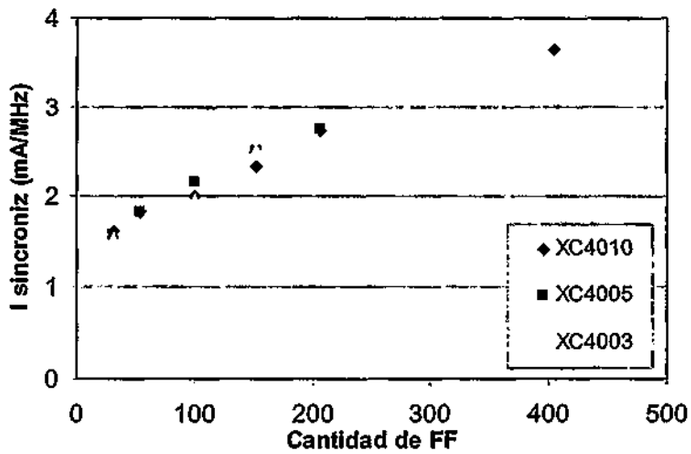
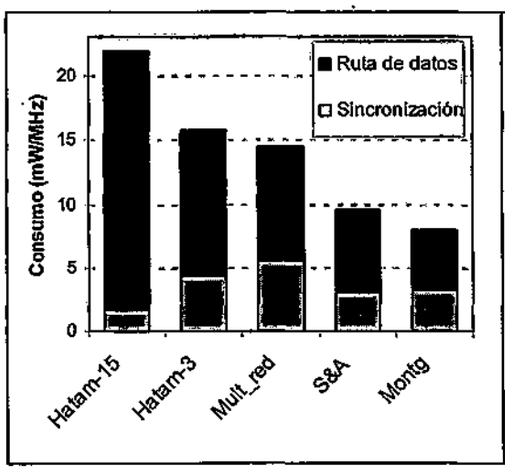


Figura 3.3. Corriente de sincronización en función de la cantidad de registros en diferentes dispositivos de la familia XC4K.



Circuito	CLBs	FF	Sinc/Total
Hatam-15	80	32	8%
Hatam-3	112	207	28%
Mult_red	57	67	39%
S&A	33	37	32%
Montg	34	31	41%

Figura 3.4. Relación del consumo de sincronización respecto del de la ruta de datos en XC4K.

Para el caso de Virtex el análisis de regresión lineal de los datos graficados en la Figura 3.5 indica un consumo para el buffer y árbol de reloj de 1,43 mW/MHz y un consumo por FF de 1,5 μW/MHz. En la Tabla 3.4 se pueden observar en algunos ejemplos la relación del consumo de sincronización respecto del de la ruta de datos en Virtex. Se han seleccionado circuitos segmentados (multiplicadores y divisores con diferentes profundidades lógicas) y circuitos secuenciales (divisores fraccionarios). Adicionalmente, en la Figura 3.28 de la sección 3.4.2 se puede observar la relación del consumo total y de sincronización respecto de la profundidad lógica en la segmentación.

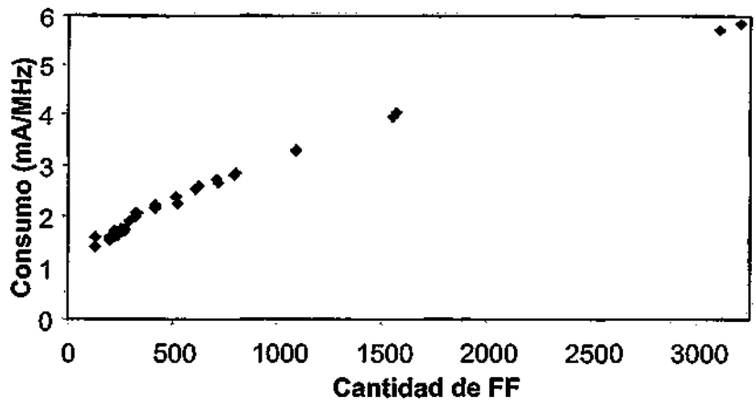


Figura 3.5. Consumo de sincronización en función de la cantidad de registros en el dispositivo Virtex XCV800PQ240-4.

	Circuitos	Área		Consumo			Syncro / Total
		Slices	FF	Sincro	ruta datos	Total	
Segmentados (*pipelined*)	srt_r2_pipe04	969	794	3,3	44,9	48,2	6,8%
	srt_r2_pipe03	1121	1089	3,8	39,4	43,1	8,7%
	srt_r2_pipe01	2179	3202	6,3	30,6	36,9	17,1%
	mult_pipe4	776	800	2,84	31,6	34,47	8,2%
	mult_pipe3	875	1.088	3,28	27,8	31,05	10,5%
	mult_pipe1	1.584	3.104	5,73	26,6	32,35	17,7%
Secuenciales (ruta de datos + FSM)	srt_r2_g1	135	240	1,66	2,96	4,62	35,9%
	srt_r2_g2	139	236	1,62	2,89	4,51	35,9%
	srt_r2_g4	169	236	1,61	3,13	4,74	34,0%
	srt_r2_g8	229	236	1,69	4,74	6,42	26,3%
	srt_r16_g4	336	255	1,74	3,41	5,16	33,8%
	srt_r16_g8	603	294	1,89	6,45	8,34	22,7%

Tabla 3.3. Ejemplos de la relación del consumo de sincronización respecto del de la ruta de datos en Virtex.

Los dispositivos Virtex II poseen un sistema de distribución de reloj bastante más avanzado que los dispositivitos predecesores (Figura 3.6) [Xil04b][Eto03]. Estos

dispositivos pueden conectar hasta 16 relojes externos y poseen hasta 12 manejadores digitales de reloj (*digital clock managers* - DCM) [Xil04b], existen además 8 árboles de reloj por cada cuarto del dispositivo. Cada cuadrante posee estos ocho relojes globales organizados en filas de reloj. Una fila de reloj soporta hasta 16 filas de CLBs (8 arriba y 8 abajo), en dispositivos mayores se agregan nuevas filas de reloj. Para ahorrar consumo cada fila no utilizada permanece desconectada.

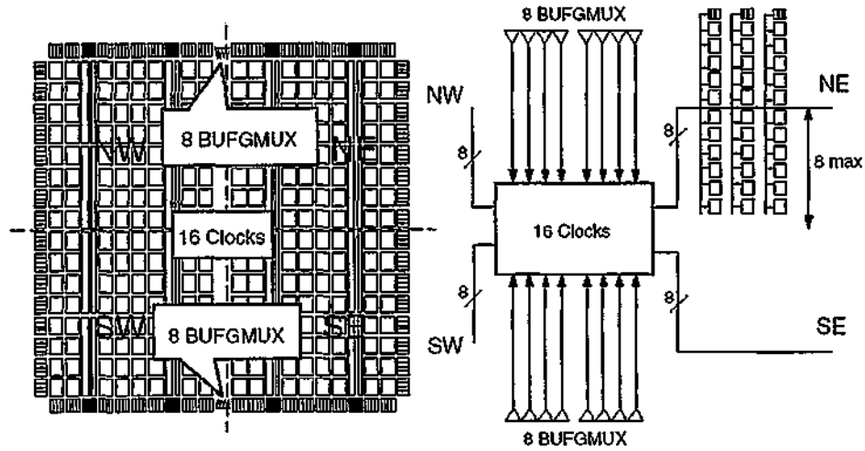


Figura 3.6. Distribución del reloj en Virtex II [Xil04b].

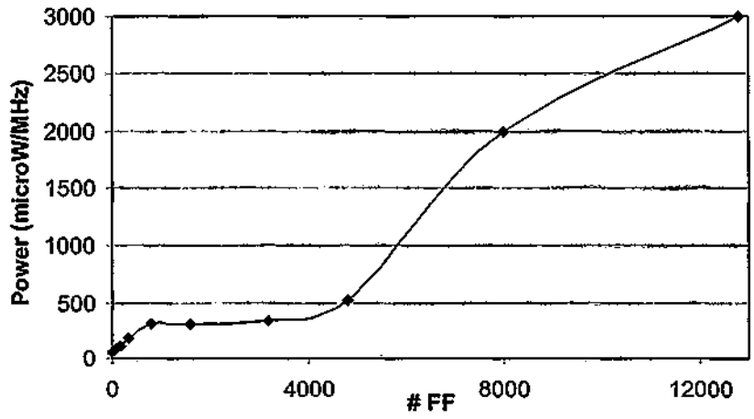


Figura 3.7. Consumo por flip-flop en el dispositivo Virtex II XC2V1500fg676-6, expresados μW/MHz.

La Figura 3.7 muestra el consumo (en μW/MHz) en función de la cantidad de flip-flops. El consumo por cada registro llega ser inferior a 0,1 μW/MHz. Se puede observa que se trata de un orden de magnitud respecto a Virtex y casi 3 ordenes de magnitud sobre los dispositivos XC4K. El aspecto discontinuo del consumo se justifica por la conexión (ó desconexión) de las líneas de reloj que realiza Virtex II.

La Figura 3.31 de la sección 3.4.3 muestra la relación del consumo total y de sincronización respecto de la profundidad lógica en la segmentación de multiplicadores de 32 bits.

3.1.4 Conclusiones sobre experimentos generales

Se han corroborado la linealidad del consumo con la frecuencia y su dependencia cuadrática con la tensión en las tres familias de dispositivos estudiadas. Si bien esta es una conclusión obvia, resulta indispensable corroborar este hecho experimentalmente como punto de partida de las futuras mediciones.

El consumo estático es del mismo orden de magnitud en XC4K y Virtex (23 mW vs 31 mW), a pesar que el dispositivo Virtex tiene una capacidad unas ochenta veces más grande. El salto a la tecnología Virtex II conlleva un incremento en el consumo estático (más de 60 mW) a pesar de la importante disminución en la tensión de alimentación. La influencia relativa del consumo estático respecto del total disminuye en Virtex respecto de XC4K, pero aumenta considerablemente en Virtex II.

Posteriormente se ha analizado la corriente de sincronización y la influencia en algunos diseños típicos. Claramente se observa una menor influencia relativa de esta componente en los dispositivos más modernos. En XC4K cada flip-flop extra implica un consumo de unos 30 μW/MHz, en tanto que en Virtex se reduce hasta 1,5 μW/MHz y para Virtex II puede llegar a representar menos de 0,1 μW/MHz. El esquema avanzado de distribución de reloj de Virtex II permite desconectar partes del arbol de reloj que no se utilicen que se refleja claramente en la menor influencia de esta componente en los diseños.

3.2 Relación entre velocidad y consumo en FPGAs

Si bien los diseñadores de sistemas basados en FPGAs cuentan con herramientas para optimizar los circuitos en área y velocidad, y a pesar de los resultados obtenidos en la investigación realizada en los últimos años [Naj94] [Ped96], los IDE (*Integrated Development Environment*) de los fabricantes de FPGAs no proveen herramientas de diseño para bajo consumo.

Nótese que la alternativa de medir el consumo de una FPGA montada en una tarjeta suele no ser viable por ser la FPGA, normalmente, un componente de un sistema mayor. Y aunque esta medición fuera posible se trata, como principio general de diseño, de estimar y optimizar el consumo de un circuito desde las primeras fases de diseño. En esta dirección Xilinx a partir de la versión 4 del software de desarrollo ISE [Ise01] ha introducido una herramienta de estimación del consumo denominada XPOWER [Xpo02] (Apéndice F) para mitigar esta deficiencia aunque aun no hay herramientas para la optimización

Frente a esta falta de herramientas de diseño para bajo consumo, interesa saber si las herramientas que se proveen y los informes que se generan para área y retardo pueden ayudar o ser útiles también en diseño para bajo consumo.

Existen trabajos en la literatura que muestran que los circuitos con mayor frecuencia máxima de operación son los que menos energía consumen. En esta sección se pretende verificar experimentalmente esta proposición para FPGAs, lo cual abre una posibilidad para la estimación y optimización de consumo frente a la falta de herramientas de diseño para bajo consumo en los dispositivos programables.

Debido a la falta de correlación entre área y consumo en FPGAs observada en [Tod00], en el presente sección sólo se analiza la relación entre frecuencia máxima de operación y consumo. Esta relación fue considerada por investigadores y fabricantes de ICs. Por ejemplo Xilinx recomienda rediseñar los circuitos haciéndolos más rápidos para disminuir su consumo, más allá de que la velocidad pedida en las especificaciones de diseño se haya obtenido [Xil95].

En particular, al optimizar los parámetros de los que depende la velocidad, como el *fan-out*, la cantidad de lógica de cálculo (CLBs) o la profundidad de lógica, se optimiza también el consumo de un circuito. En [Boe96] se ha estudiado el efecto de la

segmentación y el particionado manual en multiplicadores sobre velocidad y consumo. Los resultados de este trabajo muestran que estas acciones no sólo aumentan la velocidad de operación de los multiplicadores, sino que también se reduce el consumo para una velocidad de operación fija (Figura 3.8).

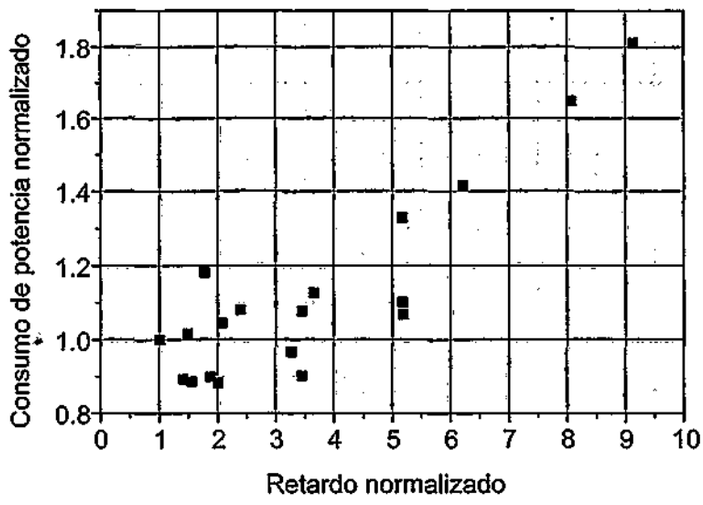


Figura 3.8. Relación ancho de banda-consumo para multiplicadores segmentados [Boe96].

La optimización de un parámetro de un circuito integrado como la velocidad, se puede atacar en diferentes niveles en el ciclo de diseño [Boe96]. A nivel de topología, se puede disponer de varias alternativas funcionalmente equivalentes. Por ejemplo para la división se pueden utilizar algoritmos de *non-restoring, restoring, SRT* en diferentes bases, *Newton-Raphson, Goldschmith*, etc. Cada una de estas alternativas tiene sus propios valores de área velocidad y consumo (ATP: *Área-Time-Power*) en una tecnología dada. Una vez seleccionada una topología, se pueden aplicar modificaciones arquitectónicas como paralelismo, segmentación o secuencialización. Por último, el circuito puede ser transformado a nivel físico: particionado y/o emplazado manual (*floorplaning*, emplazamiento relativo, restricciones de área), selección de un valor para el esfuerzo de rutado, *retiming*, restricciones de tiempo (*timming constraints*), etc.

En esta sección se exploran alternativas topológicas y de implementación para multiplicadores implementados en FPGAs de la serie XC4K de Xilinx. En el siguiente punto se resumen las principales características de los circuitos que se estudiaron y los detalles de realización de los experimentos. Más tarde se presentan y analizan los resultados obtenidos. Finalmente se presentan conclusiones y recomendaciones de diseño para reducir el consumo en FPGAs.

3.2.1 Circuitos de prueba

La relación entre velocidad y consumo y la incidencia de las diferentes decisiones de diseño planteadas en este trabajo se estudian sobre cuatro tipos de multiplicadores. Además de existir bibliografía que estudia este tipo de circuitos a lo largo de toda la historia de la computación, se pueden materializar sobre diferentes topologías lo cual convierte a los multiplicadores en casos de prueba apropiados. Las principales características de cada tipo de multiplicador se resumen en la Tabla 3.4. En estos experimentos se incluyen multiplicadores de *Wallace* [Wall64], *Hatamian-Cash* [Hat86], *Guild* [Gui69] y por último un multiplicador que resulta de la síntesis de un modelo VHDL de alto nivel ($p <= a * b$, donde a y b son las entradas y p la salida del multiplicador) [Syn99].

Test circuit	Topology	Reference	Description language	Number of CLB	Logic depth in critical path
Set A	Foundation 2.1 Synthesis	[Syn99]	VHDL	54	12 LUTs
Set B	Wallace	[Wall64]	VHDL	69	13 LUTs
Set C	Hatamian	[Hat86]	Gate level	96	15 LUTs
Set E	Guild	[Gui69]	VHDL	60	15 LUTs

Tabla 3.4. Principales características de los circuitos de prueba. Relación Velocidad-Consumo

Todas las medidas experimentales fueron realizadas en las mismas condiciones sobre una FPGA *XC4010PC84-4C* de Xilinx. Por este motivo todos los prototipos tienen prácticamente las mismas componentes de potencia estática y externa (*off-chip*). Como estas componentes del consumo no se pueden manipular en ningún momento del diseño para FPGA, se aíslan y no se consideran en los gráficos, para centrar el estudio en la componente dinámica del consumo.

Para estudiar, a nivel de diseño físico, la incidencia de los parámetros de los algoritmos de emplazamiento y rutado, se realizaron 100 implementaciones de cada multiplicador usando un proceso automático provisto en el software del fabricante. De este conjunto de implementaciones se tomo una muestra de 21 casos que fueron los que se midieron, seleccionados de manera de tener las implementaciones más rápidas, las más lentas y las de velocidad media.

3.2.2 Resultados experimentales de la relación velocidad - consumo

Los resultados más relevantes se muestran a continuación. Se analiza la relación del consumo y la velocidad por efecto de las diferentes topologías, el efecto de múltiples implementaciones dentro de la misma topología y por último la información de la actividad como métrica indirecta del consumo.

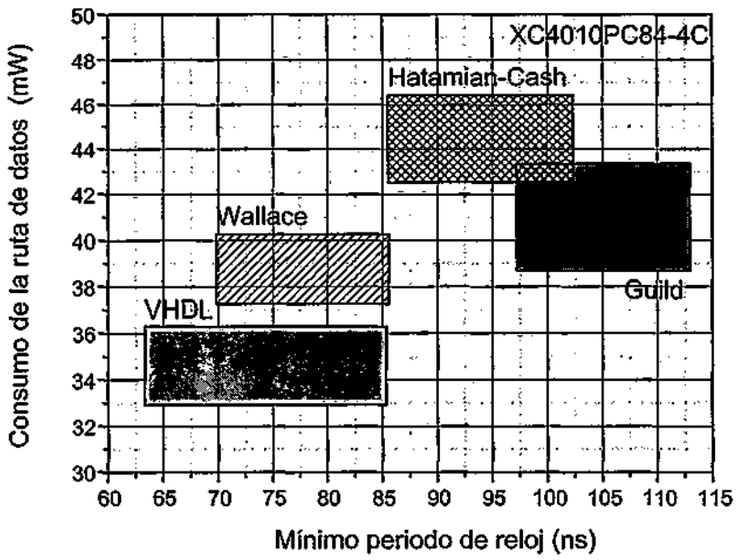


Figura 3.9. Consumo dinámica de los multiplicadores medidos.

3.2.2.1 Correlación velocidad-consumo. Efecto de las diferentes topologías

Cada topología tiene sus propias características en cuanto a área y velocidad. En la Figura 3.9 se muestran los resultados de las medidas de los cuatro conjuntos de

multiplicadores, donde cada región rectangular incluye los valores obtenidos de las 21 implementaciones seleccionadas, que se diferencian en los parámetros de emplazamiento y rutado usados durante la síntesis.

El eje de las abscisas en la Figura 3.9, muestra el mínimo período de operación, que es la inversa de la máxima frecuencia admitida por el circuito. Este valor se obtiene de los reportes generados durante la implementación. Es importante destacar que el consumo de la FPGA fue medido en todos los casos, con el dispositivo operando a 2 Mhz, independientemente de la frecuencia máxima de operación, que indica cuan rápido es el circuito.

La potencia dentro de cada conjunto de circuitos del mismo tipo (topología) varía en un factor de aproximadamente 1.1, y la máxima variación para todos los circuitos medidos es de 1.3.

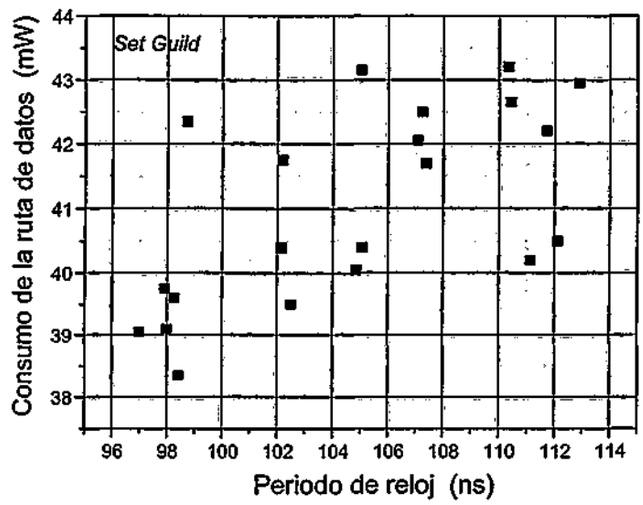


Figura 3.10. Relación velocidad-consumo para la misma topología para el multiplicador Guild.

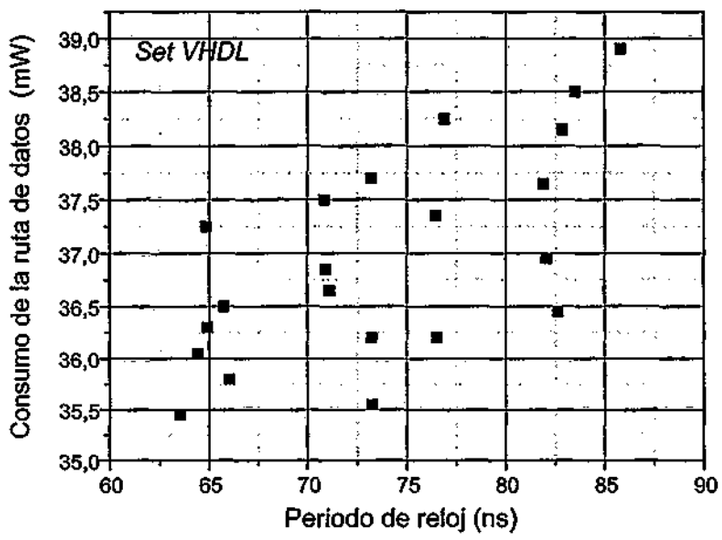


Figura 3.11. Relación velocidad-consumo dentro de la misma topología. (multipl. VHDL comportamental).

3.2.2.2 Correlación velocidad-consumo para implementaciones dentro de la misma topología

Para un mismo tipo de multiplicador, es decir, luego de haber seleccionado una topología, la diferencia entre las distintas implementaciones está en el emplazamiento y el conexionado de los CLBs.

En la Figura 3.10 se muestra el resultados obtenido para el multiplicador *Guild* y en la Figura 3.11 para el descrito en VHDL comportamental. Si bien se observa dispersión en los puntos en los gráficos, vale la suposición planteada: los circuitos rápidos son los que menos consumen. Los coeficientes de regresión son 0.63 y 0.65 respectivamente. Por ejemplo, de las 12 implementaciones con el menor período mínimo de operación, o sea las más rápidas, 10 tienen un consumo menor que la media. Para el multiplicador tipo Guild, 9 de las 10 implementaciones más rápidas, presentan un consumo menor que el valor medio.

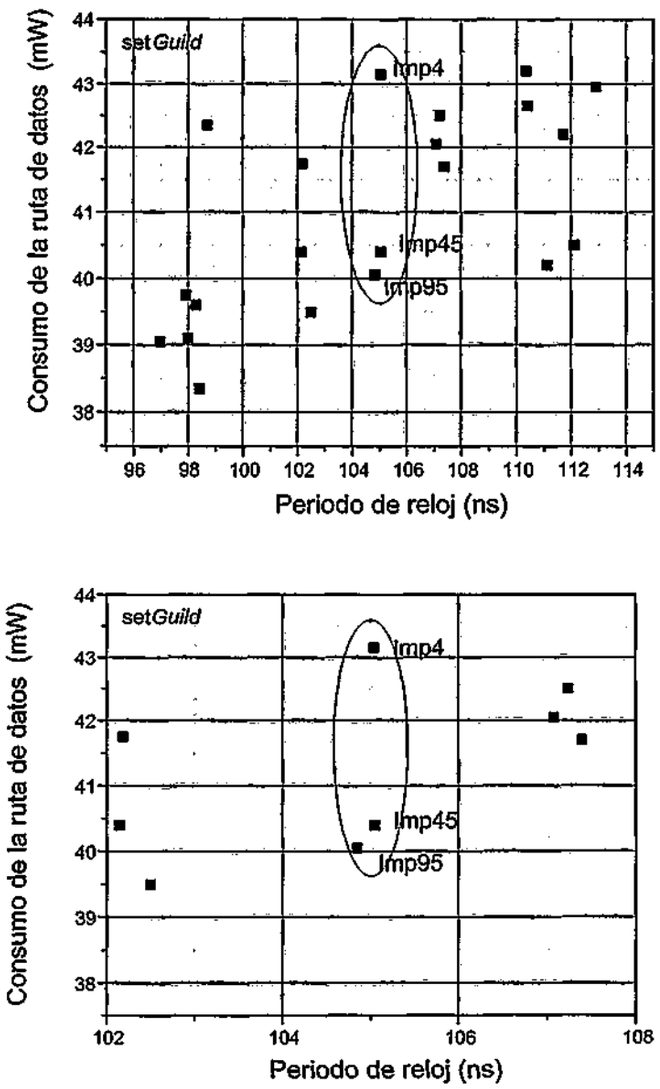


Figura 3.12. Estudio de las fluctuaciones en el consumo para circuitos de similares velocidades (Multiplicador Guid).

3.2.2.3 Importancia de los *glitches*

Mediante los experimentos reportados en esta sección se trata de explicar las fluctuaciones observadas en la relación velocidad-consumo. Esto se realiza analizando los *glitches* que se producen en los circuitos. Para ello se estudiaron algunas implementaciones para cada tipo de multiplicador. Las implementaciones analizadas fueron seleccionadas de manera que tengan casi el mismo periodo mínimo de operación (PMO) y la mayor fluctuación de consumo entre ellas. Por ejemplo para el multiplicador tipo *Guild* se seleccionaron las 3 implementaciones que se ven en la Figura 3.12 (imp4, imp45, imp95).

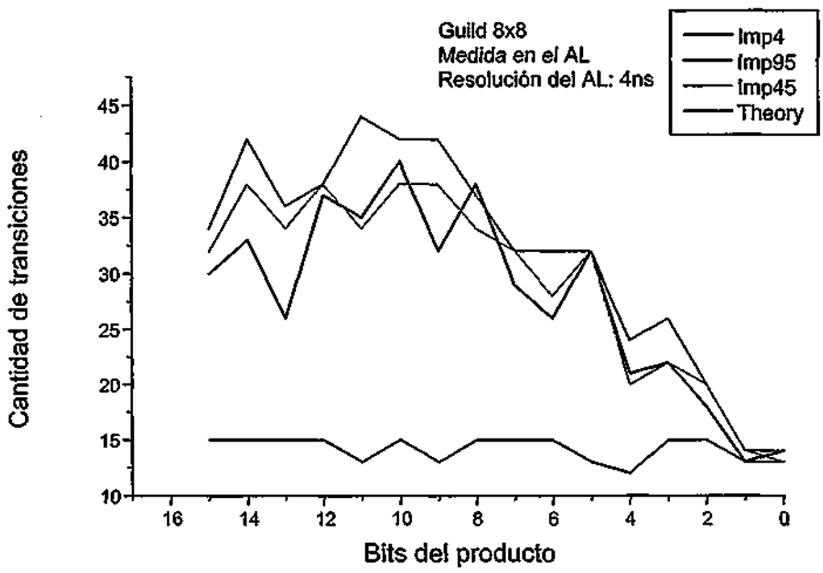


Figura 3.13. Transiciones a la salida de los multiplicadores seleccionados y obtenida por Simulación

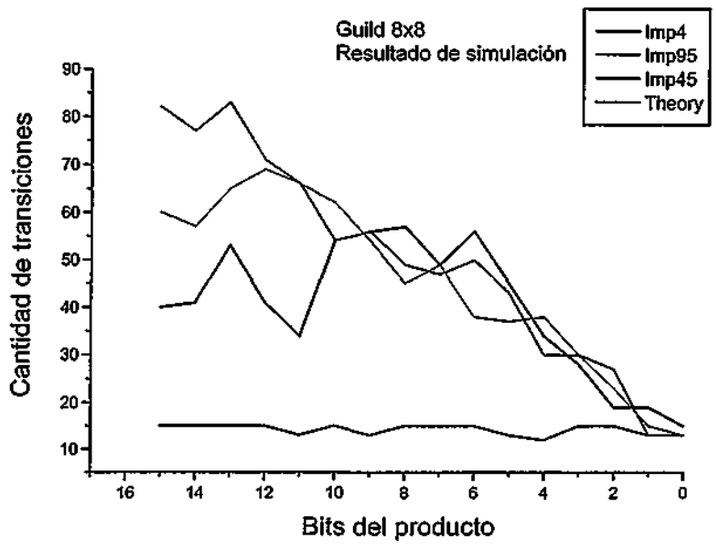


Figura 3.14. Transiciones a la salida de los multiplicadores, obtenido por medición con el analizador lógico.

Dado que las implementaciones originales tenían sus salidas registradas, se sacaron los registros (FF) de la salida usando el *FPGA Editor* de manera que no se altere el resto de la implementación. Luego se uso un analizador lógico (AL) [Tek02] para contar todas las transiciones a la salida para una serie de 16 pares de operandos que generan máxima actividad a la salida. Sin embargo, como la resolución del instrumento es de 4 ns, los valores obtenidos son una cota inferior de los verdaderos, aunque se supone que el porcentaje de error se mantiene constante para poder comparar los resultados obtenidos por este método. Además de realizar estas mediciones, se hizo un estudio idéntico en simulación. Los resultados de las Figura 3.13 y Figura 3.14 muestran la cantidad de transiciones en cada bit de salida del multiplicador por simulación y medición respectivamente.

Para resumir esta información se ha calculado la cantidad de transiciones por operación (TPO) según la ecuación $TPO = cantidad\_transiciones / número\_de\_operaciones$. Luego para la Figura 3.15 se ha definido el cociente entre la cantidad de transiciones

por operación según medición (TPOM) y la cantidad de transiciones por operación según simulación (TPOS).

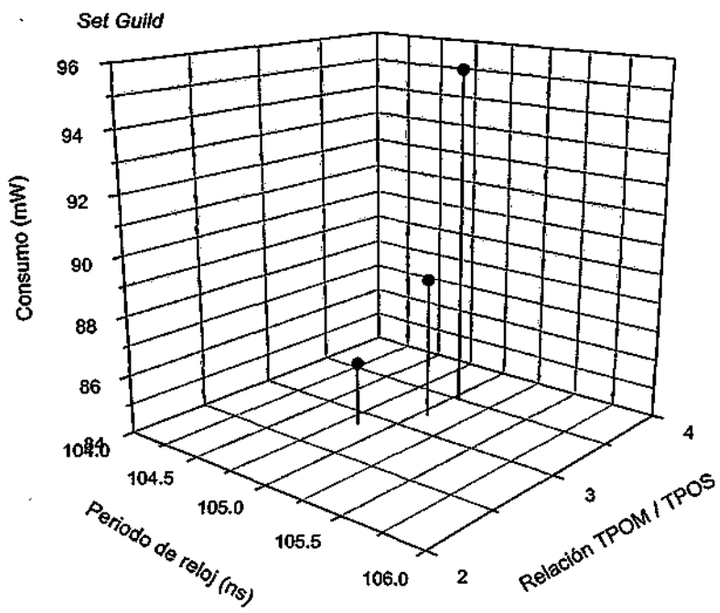


Figura 3.15. Relación entre actividad a la salida de los multiplicadores y consumo para implementaciones con similar velocidad.

En la Figura 3.15 se observa que la simple cuenta de transiciones se puede utilizar para detectar el circuito con menor consumo (los datos son tomados de un conjunto de implementaciones con velocidades prácticamente iguales). En la figura se pueden observar el consumo, el mínimo periodo de reloj y la relación TPOM / TPOS (la cantidad de transiciones por operación según medición, sobre la cantidad de transiciones por operación según simulación). Esto nos permite afirmar que, contando las transiciones espurias que brinda un simulador convencional, se puede utilizar como métrica indirecta para analizar que circuito consume menos.

3.2.2.4 Correlación área-velocidad-consumo (*Area-Time-Power*)

No existe una clara correlación entre estos parámetros. Si se comparan los cuatro multiplicadores (Figura 3.16). En términos de velocidad y consumo, los resultados pueden ser separados en dos zonas, una de ellas conteniendo los circuitos más rápidos y que menos consumen (VHDL y *Wallace*) y otro con los que más consumen y más lentos (*Guild* y *Hatamian*). La tercera variable, ocupación medida en CLBs, no es significativa en términos de consumo. Los circuitos de los conjuntos VHDL y *Guild* son los que menos ocupan, 54 y 66 CLBs respectivamente, teniendo a la vez la máxima brecha de consumo entre ellos. Por el contrario *Hatamian* y *Wallace* ocupan más pero poseen una figura del consumo más ajustada respecto del consumo.

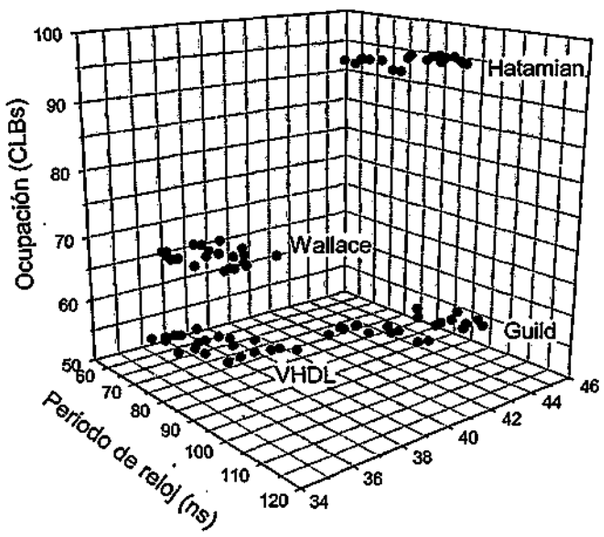


Figura 3.16. Relación entre actividad a la salida de los multiplicadores y consumo para implementaciones con similar velocidad.

3.2.3 Conclusiones de la relación velocidad - consumo.

En esta sección se estudiaron algunas de las alternativas accesibles para los usuarios del software de los fabricantes de FPGAs cuando tratan de obtener diseños con menor consumo de energía. La idea es que el usuario pueda emplear las herramientas e información disponible a partir de los informes de tiempos generados durante el ciclo de diseño para, indirectamente, mejorar el consumo. Estas recomendaciones se pueden complementar con las propuestas del trabajo de [Gar99] para el nivel arquitectónico. Las principales conclusiones son:

- Para una topología dada, el circuito con mayor máxima frecuencia de funcionamiento normalmente implica el mejor circuito en término de consumo. Esta optimización se puede obtener de manera gratuita, por ejemplo usando emplazado y rutado repetitivo (RPR - *Repetitivo Placement & Routing*) o ajustando las opciones de optimización.

- No obstante, si el diseñador debe elegir entre diferentes topologías, ni la velocidad de reloj, ni el área son parámetros por si solos validos para predecir el ahorro de consumo.

- La cuenta de las transiciones espurias que brinda un simulador convencional, se puede utilizar como métrica indirecta para analizar que circuito consume menos.

- La relación entre área y consumo no parece ser nada evidente. Algunas técnicas que ganan velocidad a expensas de mayor utilización de recursos (opciones del tipo duplicar hardware para disminuir *fan-out*) contribuyen a reducir consumo sin ser significativo el aumento de consumo por la mayor utilización de CLBs.

3.3 Conmutación de los datos de entrada (propiedad conmutativa y diseño de bajo consumo)

En las primeras clases de aritmética se menciona la conmutabilidad de las operaciones básicas, es decir, el orden de los operandos no altera el resultado del cálculo. Sin embargo, el diseñador de circuitos integrados debe cuestionarse el alcance de esta propiedad, cuando uno de los objetivos es la reducción de consumo de potencia.

En esta sección se muestra que, en términos de consumo de potencia, los circuitos digitales aritméticos, no siempre cumplen la propiedad conmutativa. En consecuencia, es posible obtener una reducción de consumo adicional, simplemente permutando las entradas del circuito. El fenómeno se refuerza cuando se cumplen algunas de las siguientes condiciones: el volumen de datos a procesar no es elevado y de carácter no aleatorio, el circuito tiene una implementación irregular y finalmente, los bits de uno de los datos se distribuyen a través de líneas globales. Para verificar experimentalmente este fenómeno, se han construido y medido multiplicadores binarios utilizando FPGAS de la serie XC4K y Virtex de Xilinx.

Como se mencionó en las secciones precedentes, dependiendo de la profundidad de lógica del circuito, *glitches* generados en las primeras etapas del circuito producen un efecto avalancha en la actividad, que se constituye en la principal componente del consumo del circuito [Ped96][Rag96][Rab96][Mee95]. Este fenómeno conduce a que el consumo dinámico varíe con el orden de los operandos, dado que éste modifica la cantidad de actividad espuria.

De lo antedicho se puede deducir que si el *layout* del circuito no es simétrico y por lo tanto, los retardos de las señales de entrada son diferentes, la cantidad de *glitches* que se generan cuando se realiza A×B, será distinta de la correspondiente a B×A. Aún así, en términos de potencia media no se debería detectar una variación importante si la secuencia de operandos fuese suficientemente larga y aleatoria. En tal caso, cada par de operandos debería aparecer repetido varias veces de la forma A×B y B×A. Por el contrario, el efecto debería ser visible en la mayoría de los casos reales de procesamiento de señal, donde existe una fuerte correlación entre datos sucesivos, o cuando el número de vectores a procesar es pequeño.

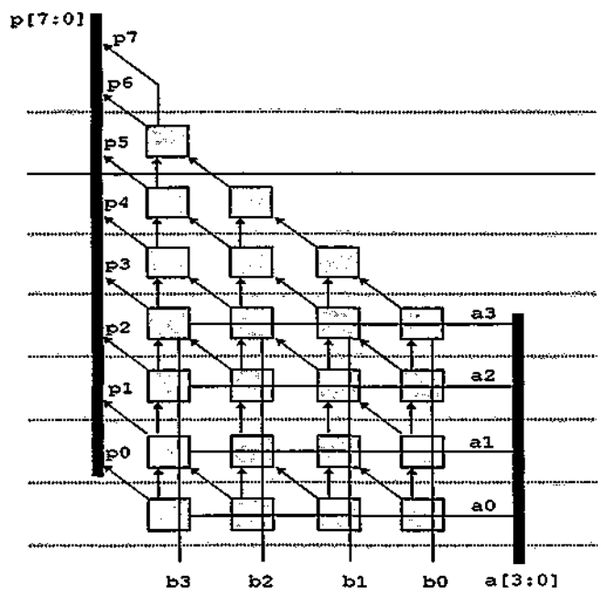


Figura 3.17. Arreglo segmentado Hatamian-Cash de 4 bits con entradas *b* locales y *a* globales [Boe95].

La asimetría del consumo se puede reforzar por un segundo mecanismo. Para ello, se deben combinar dos factores que aparecen a menudo en procesamiento digital de señal. En primer lugar la difusión (*broadcast*) de uno de los operandos se realiza a través de pistas globales, y en segundo lugar una de las señales tiene una frecuencia de variación menor (por ejemplo, los coeficientes de un filtro). En tal caso, se reducirá el consumo medio si por las pistas globales (las de mayor capacidad) se introducen los datos que tienen menor variabilidad. Muchos circuitos segmentados (*pipelines*) presentan entradas globales y locales. En la Figura 3.17 [Boe95] se muestra un ejemplo: el multiplicador de *Hatamian y Cash*, (por simplicidad, su tamaño se ha reducido a 4 bits), cada PE (*Product Element*) está formado por una AND y un sumador completo (*full-adder*). Las líneas de entrada *a* y *b* son globales sólo en la versión combinacional. Si se segmenta (por ejemplo, como indican las líneas punteadas), las entradas de *b* se transforman en locales, al ser seccionadas por los registros de segmentación. Por lo tanto, a medida que aumenta el tamaño de palabra, aumenta la diferencia de capacidad entre las dos entradas.

En esta sección se realizan varios experimentos sobre los argumentos anteriores. A continuación se resumen las características de los circuitos de prueba, y más tarde se presentan los métodos de medida y se muestran los principales resultados.

3.3.1 Circuitos de pruebas familia 4K

Para demostrar el efecto sobre el consumo de la permutación de los operandos, se han construido y medido 7 prototipos utilizando FPGAs de Xilinx de la serie XC4K. El arreglo experimental se describe en el Apéndice A. Las pruebas se han realizado utilizando cuatro juegos diferentes de vectores. Como operación aritmética, se ha seleccionado la multiplicación binaria, debido a que es la tarea central en cualquier circuito de procesado de señal, es habitualmente utilizada como *benchmark* para evaluar una determinada tecnología VLSI, y finalmente, su evolución puede rastrearse a lo largo de la historia de las máquinas de computación al menos durante los últimos 150 años [Gol93].

Las principales características de los circuitos de prueba se resumen en la Tabla 3.5. La profundidad lógica se expresa en cantidad de LUTs en el camino crítico. Los circuitos VHDL-12 y Xcore-9 se han obtenidos directamente de las herramientas del fabricante; el primero sintetizado a partir de una descripción VHDL a nivel comportamental y el segundo mediante la herramienta de generación de *cores* de Xilinx. Complementariamente, se incluye un juego de arreglos *Hatamian* [Hat86] segmentados con diferentes profundidades de lógica.

Topología	Ref.	CLBs	FF	Prof. de lógica (max.) LUTs	Max. Frecuencia MHz
Guild-16	[Gui69]	60	32	16	21.0
VHDL-12	[Xil00a]	56	32	12	32.1
Wallace-12	[Wal64]	71	32	12	29.5
Hatamian-8	[Hat86]	75	54	8	25.8
Hatamian-3	[Hat86]	112	207	3	66.2
Hatamian-2	[Hat86]	207	404	2	70.9
Xcore-9	[Xil00b]	52	96	9	78.1

Tabla 3.5. Principales características de los circuitos de prueba.

3.3.2 Resultados experimentales familia 4K

En esta sección se resumen los principales resultados experimentales. Los circuitos han sido implementados usando FPGAs Xilinx modelos XC4010, XC4005 y XC4003,

con encapsulados PLCC84. Han sido compilados con la herramienta Xilinx Foundation 3.1i utilizando la opciones de *maximum place & route effort* y *tied*. Todos los circuitos tienen la E/S registrada (con FF de CLBs). Adicionalmente, todas las salidas pasan a través de un *buffer* de control de alta impedancia.

Todas las medidas fueron realizadas a 2 MHz. La reducción de potencia PR se ha calculado como la reducción porcentual del consumo que se obtiene respecto de circuito original, cuando se aplica una determinada modificación, en este caso la permutación de los datos de entradas. Es decir, $PR = 100 * (P_{original}-P_{modificiado})/P_{original}$.

3.3.2.1 Esquema de medición

Todos los prototipos fueron medidos en la misma placa de prueba, utilizando idéntica asignación de patas (*pins*) de E/S. La única carga a la salida corresponde a la punta de prueba del analizador lógico, inferior a 3 pF [Tek02]. Por lo tanto, todas las versiones tiene idéntica potencia *off-chip*, alrededor de 0.13 mW/MHz por pata de salida (para números aleatorios) y 0,18 mW/MHz para la secuencia de máxima actividad.

Se utiliza una FPGA de la serie XC3K (ver apéndice A), para generar cuatro juegos diferentes de vectores, cuyas características se resumen en la Tabla 3.6. El consumo medio se midió indirectamente, determinando la corriente de entrada del circuito bajo prueba. Las tres componentes de la potencia (ruta de datos, sincronización y *off-chip*) fueron separadas según la técnica descrita en el apéndice A.

Nombre	Descripción
Random-1	64 pares de vectores aleatorios.
Random-8	Uno de los operandos posee una frecuencia 8 veces menor. Conforman 512 pares de valores
Toggle-1	16 pares de vectores sintetizados para maximizar la actividad del multiplicador a la entrada y salida del mismo. En cada ciclo varían del 38% al 81% de los bits de entrada y del 50% al 100% de los bits de salida.
Toggle-8	El operando A posee una frecuencia 8 veces menor. Son 128 pares de valores

Tabla 3.6. Descripción de los vectores de prueba

3.3.2.2 Resultados

En la Figura 3.18 se muestra la reducción de potencia obtiene en la ruta de datos, para el juego de multiplicadores *Hatamian-Cash*. En todos los casos, se obtiene mayores reducciones de potencia para los vectores que maximizan la actividad en la E/S. El

valor de PR para números aleatorios es bajo pero no es cero, debido a que la secuencia se aleja del caso teórico al limitarse a solamente 64 vectores.

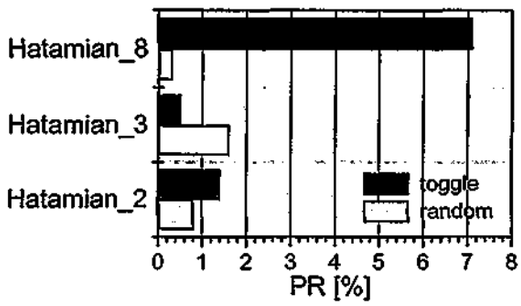


Figura 3.18. Reducción de consumo. Areglos *Hatamian-Cash*.en XC4010PC84.

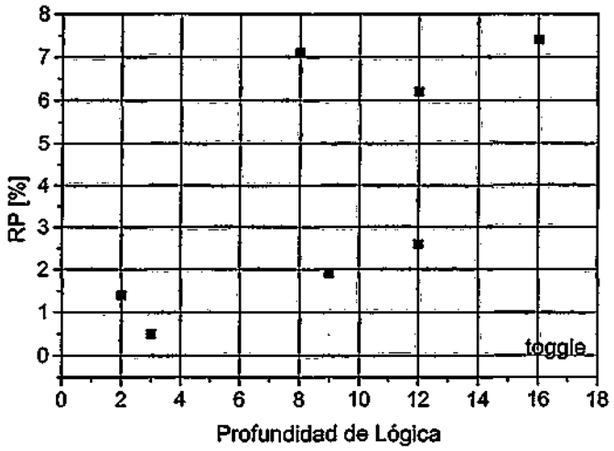


Figura 3.19. Reducción de consumo en función de la profundidad de lógica de los circuitos. XC4010PC84

Por el contrario, los vectores de máxima actividad enfatizan la diferencia de potencia producida por el orden de los operandos. La mayor actividad en las entradas se traduce en mayor cantidad de *glitches*, la causa central de esta asimetría. La figura también muestra que el efecto es más significativo cuando la profundidad de lógica es

mayor, debido al efecto avalancha de los *glitches*. Este último fenómeno se repite en la Figura 3.19, donde se muestra la reducción de potencia para todos los circuitos de prueba. Aunque los puntos distan de ser una recta, existe una clara correlación.

Utilizando una herramienta informática [Tod02] se mide la cantidad de transiciones para un multiplicador *Hatamian-8* con la secuencia *toggle-8*. Esta herramienta obtiene la cuenta de transiciones en función de los datos de salida de un simulador convencional. La Tabla 3.7 muestra los valores de actividad de cada entrada y salida y resume el total de actividad comparándolo con los valores correspondientes a la medición sobre el dispositivo XC4005.

Al igual que en la sección anterior, se pone de manifiesto la utilidad de la cuenta de transiciones espurias obtenidas por un simulador tradicional como métrica indirecta del consumo. Queda claro además, que ésta métrica nos da una idea de la diferencia de actividad de una secuencia respecto de la otra, pero no puede ser tomada como una medida exacta de la diferencia de consumo, ya que no se tiene en cuenta la capacidad asociada a cada transición.

Nodo	AxB	BxA
A[7]	49	14
A[6]	65	11
A[5]	81	13
A[4]	64	12
A[3]	49	13
A[2]	80	12
A[1]	47	12
A[0]	80	13
Total Pad_A	515	100

Nodo	AxB	BxA
B[7]	14	49
B[6]	11	65
B[5]	13	81
B[4]	12	64
B[3]	13	49
B[2]	12	80
B[1]	12	47
B[0]	13	80
Total Pad_B	100	515

Nodo	AxB	BxA
P[15]	42	42
P[14]	54	54
P[13]	67	67
P[12]	64	64
P[11]	69	69
P[10]	60	60
P[9]	60	60
P[8]	61	61

Nodo	AxB	BxA
P[7]	71	71
P[6]	69	69
P[5]	68	68
P[4]	61	61
P[3]	48	48
P[2]	81	81
P[1]	49	49
P[0]	38	38
Total Pad_P	962	962

	AxB	BxA	Diferencia
Total nodos analizados	580	580	-
Cantidad Total de Transiciones	47476	52158	9,0 %
Promedio transición por nodo	81,86	89,93	9,0 %
Medición consumo ruta de datos (mW/MHz)	21,3	23,2	8,2 %

Tabla 3.7. Actividad de las entradas, salidas y totales para la operación AxB y BxA con vectores de máximo *toggle-8*

El conjunto de arreglos *Hatamian-Cash* también permiten ilustrar la reducción de consumo cuando coinciden las líneas globales con un operando que varía con menor frecuencia. Es decir, el caso más favorable, donde las líneas más cargadas del circuito tienen menos actividad. En la Figura 3.20 se muestran los resultados. Se han elegido las medidas correspondientes al conjunto de vectores aleatorios (*random*), para minimizar un posible enmascaramiento producido por una mayor actividad espuria. En todos los casos, el consumo resultó mínimo cuando el operando de menor frecuencia se introdujo por la entrada global *a[7:0]*. Obsérvese que el efecto es menor para el arreglo *Hatamian_8*, el cual tiene solo dos etapas de *pipelining*. Esto hace que la entraba de datos por *b[7:0]* también sea global, anulándose de este modo la diferencia entre un caso y otro.

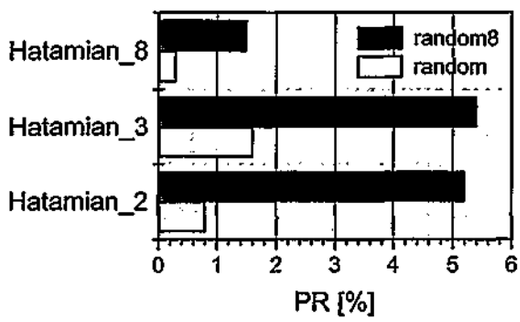


Figura 3.20. Efecto de las líneas globales. XC4010PC84.

Finalmente, en la Figura 3.21 se muestra los valores de reducción de consumo obtenidos para los otros circuitos. Con excepción del multiplicador de *Wallace*, en los demás casos, el efecto se magnifica cuando los datos producen mayor actividad espuria.

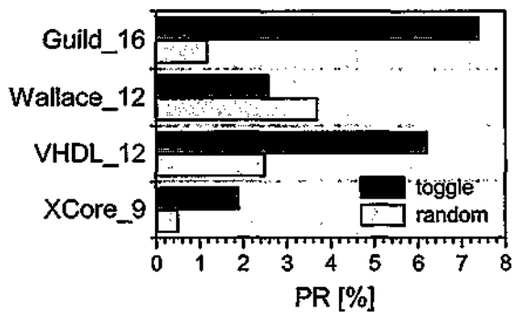


Figura 3.21. Reducción de consumo para diferentes topologías. XC4010PC84.

Tal como se menciona anteriormente se realizaron las implementaciones en diferentes circuitos de la familia Xilinx 4K, concretamente sobre un XC4003EPC84, XC4005EPC84 y XC4010EPC84 obteniéndose similares resultados en todos ellos. En la Figura 3.22 se observa las variaciones de consumo para el multiplicador *Hatamian-8* para las secuencia *toggle-8* y *Random-1* en los diferentes circuitos.

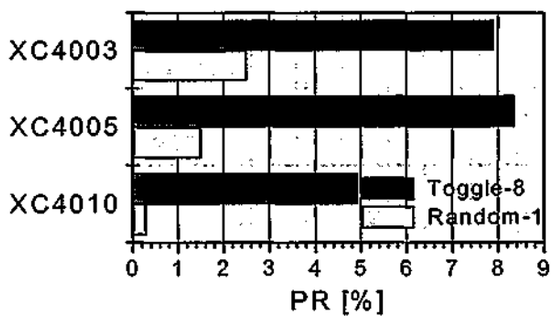


Figura 3.22. Reducción de consumo para diferentes circuitos familia XC40XXPC84 (multiplicador Hatamian-8)

3.3.3 Circuitos de pruebas familia Virtex

Para demostrar el efecto de la permutación de los operandos en Virtex, se han construido y medido 5 multiplicadores de 16 bits y otros 5 de 32 bits utilizando FPGAs Xilinx de la familia Virtex. De forma análoga a las pruebas sobre XC4K, se han realizado utilizando dos juegos diferentes de vectores, uno de máxima actividad en una entrada y dieciséis veces menor en la otra (*MaxTog*) y otro con valores aleatorios pero con diferente frecuencia en una entrada y en la otra (*AvgTog*), como en el caso típico de la multiplicación de matrices o el de muchos filtros digitales.

Circuitos 32 bits			
Circuito	Área (Slices)	Min. period. (ns)	Max. Frec. (MHz)
Core32	580	48.2	20.7
Exp32	561	38.2	26.1
Leo32	565	44.3	22..5
Syn32	571	47.4	21.1
Xst32	576	47.8	20.9

Circuitos 16 bits			
Circuito	Área (Slices)	Min. period. (ns)	Max. Frec. (MHz)
Core16	157	23.0	43.4
Exp16	149	19.9	50.0
Leo16	150	22.9	43.6
Syn16	152	22.1	45.2
Xst16	156	21.9	45.5

Tabla 3.8. Principales características de los circuitos de prueba. Familia Virtex

Cuatro de los circuitos de prueba son multiplicadores descritos de forma comportamental (*behavioral*) en VHDL y sintetizado con cuatro sintetizadores diferentes (*Syn*: Synplify Pro [Syn02], *Xst*: Xilinx Sintesis Technologies [Xil02a]; *Leo*: Leonardo Spectrum [Men02] y *Exp*: FPGA Express [Syn01]). El multiplicador restante es el obtenido por el generador de circuitos *CoreGen* [Xil02b] provisto por la herramienta ISE. Todos los circuitos son totalmente secuenciales y están registrados en las entradas y las salidas con registros en los *slices* internos. Las características de área y velocidad de los circuitos se resumen en la Tabla 3.8.

3.3.4 Resultados experimentales familia Virtex

El dispositivo utilizado es una Virtex XCV800HQ240. Los detalles del arreglo experimental así como de la metodología de medición se pueden ver en el apéndice B. En la Tabla 3.9 y Tabla 3.10 se pueden observar las diferencias de consumo para la

secuencia de máxima actividad y actividad promedio (*MaxTog* y *AvgTog*) para los multiplicadores de 16 y 32 bits respectivamente.

En las tablas se especifica la componente dinámica del consumo expresado en mW/Mhz. La tercera columna indica el aumento de consumo de una permutación respecto de la otra. El signo positivo indica que P(A×B) es mayor P(B×A), en tanto que el negativo la situación contraria.

Circuitos 32 bits	*MaxTog*			*AvgTog*		
	P_A×B	P_B×A	% P din	P_A×B	P_B×A	% P din
Core32	34,12	27,77	22,86%	11,92	9,94	20,00%
Exp32	23,81	29,39	-23,41%	9,56	10,22	-6,93%
Leo32	31,40	27,87	12,70%	11,70	10,62	10,24%
Syn32	32,31	35,12	-8,70%	10,04	12,00	-19,56%
Xst32	32,29	29,45	9,64%	11,71	10,62	10,24%

Tabla 3.9. Diferencia de consumo A×B vs. B×A en multiplicadores de 32 bits.

Circuitos 16 bits	*MaxTog*			*AvgTog*		
	P_A×B	P_B×A	% P din	P_A×B	P_B×A	% P din
Core16	7,57	5,43	39,31%	2,45	2,20	11,72%
Exp16	6,42	6,98	-8,76%	2,41	2,53	-4,71%
Leo16	7,69	6,01	27,95%	2,53	2,26	11,96%
Syn16	5,82	7,63	-31,13%	2,18	2,37	-8,96%
Xst16	7,21	6,06	18,94%	2,40	2,30	4,12%

Tabla 3.10. Diferencia de consumo A×B vs. B×A en multiplicadores de 16 bits.

Se puede observar que en todos los casos existe una importante diferencia de consumo al invertir el orden de las entradas (hasta un 39% superior). La variación en el sentido o dirección de la diferencia de consumo (signo positivo o negativo en la relación porcentual) responde al hecho de que cada sintetizador utiliza su propio algoritmo para construir los multiplicadores. Es importante destacar que para cada sintetizador se observa que el signo de la diferencia de consumo se mantiene tanto para multiplicadores de 16 y 32 bits, como para secuencias de máximo movimiento (*MaxTog*) y movimientos aleatorios (*AvgTog*).

3.3.4.1 Relación retardo - consumo

En la Figura 3.23 se muestra la relación del consumo promedio de los diferentes multiplicadores de 32 bits en función del retardo, lo que refuerza los argumentos del apartado 3.2, donde se puede ver una clara correlación consumo-retardo para ambos conjuntos de patrones de entrada.

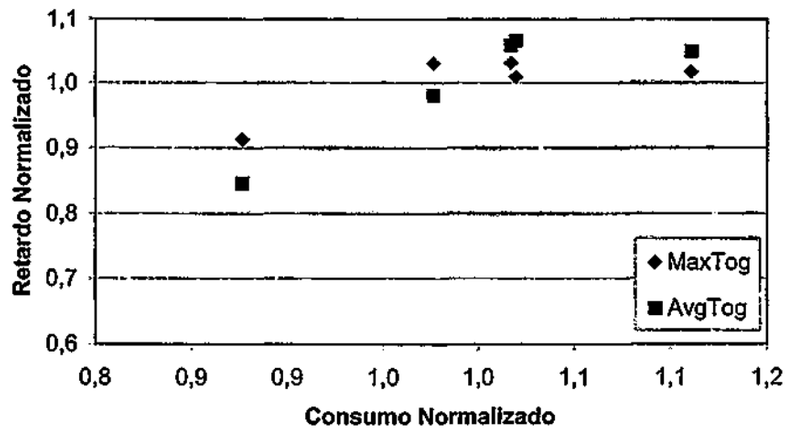


Figura 3.23. Relación retardo - consumo para los multiplicadores de 32 bits.

3.3.4.2 Uso de herramientas de estimación de consumo

Con la conjunción de una simulación *post Place&Route* con Modelsim [Men02] y la herramienta de estimación de consumo XPOWER [Xpo02] se puede obtener una estimación del consumo. El objetivo de este apartado es verificar la precisión de la herramienta Xpower y determinar la utilidad de esta para discernir mejores implementaciones. En el apéndice E, se describe la utilización de la herramienta Xpower, así como algunos *scripts* para la automatización y simplificación en el uso de esta herramienta.

Todos los circuitos medidos en el arreglo experimental del Apéndice B, fueron simulados y estimado su consumo con los mismos patrones de entrada. A continuación se resumen los resultados.

Circuitos	Medidas en Placa			Xpower			Relación Xpow / medidas	
	A×B	B×A	% Pdif	A×B	B×A	% Pdif	A×B	B×A
Core32	34,12	27,77	22,86%	57,4	42,0	36,61%	168%	151%
Exp32	23,81	29,39	-23,41%	36,6	44,3	-20,82%	154%	151%
Leo32	31,40	27,87	12,70%	56,1	39,6	41,64%	179%	142%
Syn32	32,31	35,12	-8,70%	23,3	38,5	-65,59%	72%	110%
Xst32	32,29	29,45	9,64%	61,6	43,3	42,49%	191%	147%

Tabla 3.11. Relación medidas – estimaciones para multiplicadores de 32 bits y secuencia MaxTog.

Circuitos	Medidas en Placa			Xpower			Relación Xpow/medidas	
	A×B	B×A	% Pdif	A×B	B×A	% Pdif	A×B	B×A
Core32	11,92	9,94	20,00%	18,8	16,5	13,64%	157%	166%
Exp32	9,56	10,22	-6,93%	15,0	17,1	-14,17%	157%	168%
Leo32	11,70	10,62	10,24%	18,5	17,0	8,82%	158%	160%
Syn32	10,04	12,00	-19,56%	10,4	12,5	-20,48%	103%	104%
Xst32	11,71	10,62	10,24%	18,5	17,8	4,23%	158%	167%

Tabla 3.12. Relación medidas – estimaciones para multiplicadores de 32 bits y secuencia AvgTog.

Circuitos	Medidas en Placa			Xpower			Relación Xpow / medidas	
	A×B	B×A	% Pdif	A×B	B×A	% Pdif	A×B	B×A
Core16	7,57	5,43	39,31%	12,5	10,4	20,48%	165%	191%
Exp16	6,42	6,98	-8,76%	9,0	10,6	-18,06%	140%	152%
Leo16	7,69	6,01	27,95%	12,1	9,5	27,63%	158%	158%
Syn16	5,82	7,63	-31,13%	6,4	7,9	-23,53%	110%	103%
Xst16	7,21	6,06	18,94%	11,4	9,4	21,33%	158%	155%

Tabla 3.13. Relación medidas – estimaciones para multiplicadores de 16 bits y secuencia MaxTog.

Circuitos	Medidas en Placa			Xpower			Relación Xpow/medidas	
	A×B	B×A	% Pdif	A×B	B×A	% Pdif	A×B	B×A
Core16	2,45	2,20	11,72%	4,8	4,4	8,57%	194%	199%
Exp16	2,41	2,53	-4,71%	4,3	4,5	-5,88%	176%	178%
Leo16	2,53	2,26	11,96%	4,8	4,5	5,56%	188%	199%
Syn16	2,18	2,37	-8,96%	3,1	3,4	-8,00%	144%	142%
Xst16	2,40	2,30	4,12%	4,5	4,4	2,86%	188%	190%

Tabla 3.14. Relación medidas – estimaciones para multiplicadores de 16 bits y secuencia AvgTog.

En las tablas anteriores (Tabla 3.11, Tabla 3.12, Tabla 3.13 y Tabla 3.14) se puede ver el consumo expresado en mW/mHz de consumo dinámico tanto para las mediciones como para las estimaciones de Xpower. En éstas tablas se puede observar la diferencia de consumo A×B vs B×A medidos y estimados, así como la diferencia porcentual en la relación estimado/medido para cada orden de los operandos. La Figura 3.24 y Figura 3.25 muestran la diferencia de consumo estimado y medido para los diferentes circuitos, orden de entrada de los operandos (A×B y B×A) y los diferentes patrones de excitación (*MaxTog* y *AvgTog*). Las principales observaciones son:

- Es importante notar que el "signo" de la diferencia de consumo se respeta en todos los casos, es decir si las medidas indican que es mejor hacer A×B que B×A la estimación vía simulación también lo hace. Esto permite afirmar que el uso de esta herramienta de estimación es adecuado para discernir la permutación de las entradas
- El consumo por estimación es en promedio un 50% mayor respecto de las medidas (Figura 3.24 y Figura 3.25). Lo cual nos permite concluir que las estimaciones de consumo dinámico son de algún modo pesimistas. Solo el consumo por estimación del circuito sintetizado por Synplify [Syn02] es menor que la media.

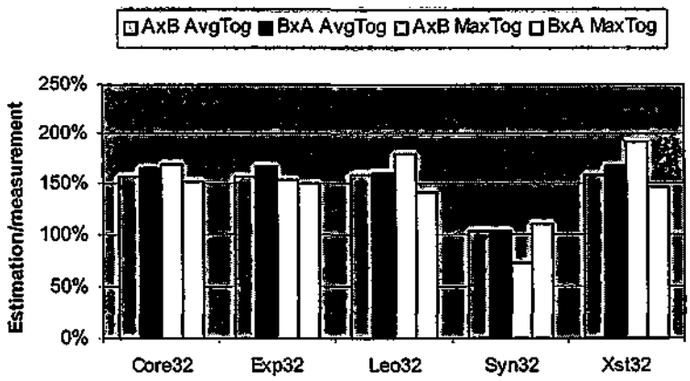


Figura 3.24. Diferencia en el consumo estimado y medido multiplicadores 16 bits

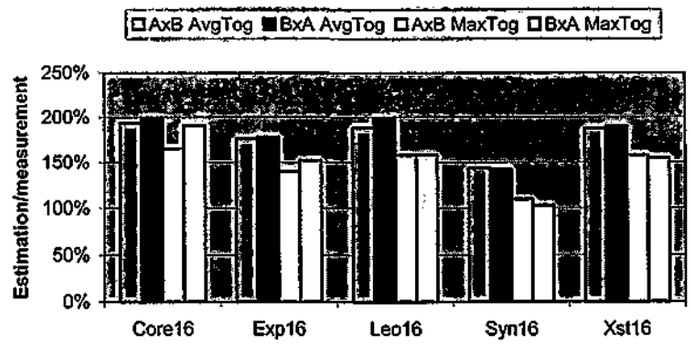


Figura 3.25. Diferencia en el consumo estimado y medido multiplicadores de 32 bits

3.3.5 Conclusiones de la conmutación de datos

En esta sección se ha analizado en el fenómeno de la conmutatividad en la operación de multiplicación respecto al consumo de potencia. Se prueba a través de diferentes ejemplos que bajo ciertas condiciones el solo hecho de permutar las entradas tiene un fuerte impacto en la reducción de consumo del consumo debido al cálculo. En la familia XC4K se utilizaron multiplicadores de 8 bits llegando la diferencia hasta un 8 % en el consumo. En la familia Virtex con multiplicadores de 16 y 32 bits se llega a una diferencia de hasta un 39%.

Las componentes de este desbalance en el consumo puede ser un diseño irregular o la implementación irregular que se genera al mapear un diseño en una FPGA, esto produce que los *glitches* generados en una secuencia de operaciones no sean iguales al permutar las entradas. Otra componente es el uso de líneas globales (que poseen mayor capacidad) para uno de los operandos, lo que puede generar diferencias en el consumo. Por ello, si la secuencia de valores a operar no es aleatoria o la frecuencia de uno de los operandos es diferente al otro (multiplicación de matrices, operaciones sobre video, filtros, etc.) conviene analizar el orden de los operandos.

Cabe esperar que otros bloques combinacionales cuyas entradas sean conmutativas tiendan a tener esta característica de desbalance en el consumo. Este efecto también se producirá en circuitos segmentados, cuando la segmentación produzca que un operando se propague sobre globales y el otro locales.

Se ha validado el uso de herramientas de estimación del consumo a la hora de elegir la mejor el orden de entrada de los operandos para la reducción de consumo. También se observo que una herramienta más simple, y por ende más veloz, que solo cuente la cantidad de transiciones en un caso y otro puede ser suficiente para decidir el orden de las entradas.

3.4 Efecto de la segmentación en el consumo

La segmentación (*pipeline*) es una antigua y eficiente manera de aumentar la performance de los circuitos, popularizada a principio del siglo XX por Henry Ford en la optimización de la cadena de montaje de sus Ford modelo T. La relación del consumo con el *pipeline* proviene del hecho de que la segmentación reduce la actividad espuria (*glitches*) como se describe más adelante.

Si bien los *glitches* no producen errores en los diseños síncronos bien diseñados pueden ser responsables hasta de un 70 % de la consumo del circuito [she92] y como se midió en esta sección puede superar el 80 % de la actividad total. Como se describió en el capítulo 2, los *glitches* pueden ser reducidos básicamente por dos vías, por un lado ecualizando caminos [Sak98] [Ped96] o bien reduciendo la profundidad lógica introduciendo registros [cha92]. La relación profundidad lógica con el consumo fue analizada en detalle en múltiples trabajos [Lei85][Mus95][Boe98][Wil04].

En esta sección se estudian los efectos de la segmentación en el consumo. En primer lugar se presentan resultados para multiplicadores sobre diferentes dispositivos de la familia XC4K (sección 3.4.1), luego se analiza los resultados para multiplicadores de 32 bits implementados en la familia de dispositivos Virtex (sección 3.4.2) y por último se muestran los resultados para los mismos circuitos implementados en Virtex II (sección 3.4.3).

3.4.1 Medidas sobre XC4K

A fin de comprobar la relación del consumo respecto de la profundidad lógica se extendió el grupo de circuitos *Hatamian-Cash* [Hat86] del descritos en el apartado 3.3.3 a los expuestos en la Tabla 3.15. Los circuitos no implementados son debido al tamaño de las FPGAs.

Los resultados comprueban los conceptos expresadas en [Boe86] respecto de la relación del consumo con la profundidad lógica (Figura 3.26) para los diferentes circuitos de la familia XC4000. Se observa que el mínimo consumo se logra para una segmentación media (profundidad 4-5 CLBs, 4-3 niveles de segmentación). Cuando la segmentación es nula (multiplicador totalmente combinacional), o muy pequeña, la avalancha de *glitches* es la principal componente de consumo, a medida que se agrega

segmentación disminuye la corriente producida por los *glitches* pero aumenta la corriente de sincronización, siendo ésta la componente más importante en un circuito altamente segmentado en la familia XC4K (Figura 3.27).

Topología de Circuito	CLBs	FFs	Prof Log LUTs	Retardos (ciclos clk)	BW normalizado (MHz)		
					XC4010	XC4005	XC4003
Hatamian-15	80	32	15	2	16,7	16,0	15,7
Hatamian-8	75	54	8	3	25,8	29,3	37,3
Hatamian-5	87	101	5	5	42,0	47,2	53,7
Hatamian-4	97	153	4	6	53,6	54,6	42,4
Hatamian-3	112	207	3	9	66,2	75,4	—
Hatamian-2	207	404	2	17	70,7	—	—

Tabla 3.15. Principales características de las diferentes topologías de prueba XC4K.

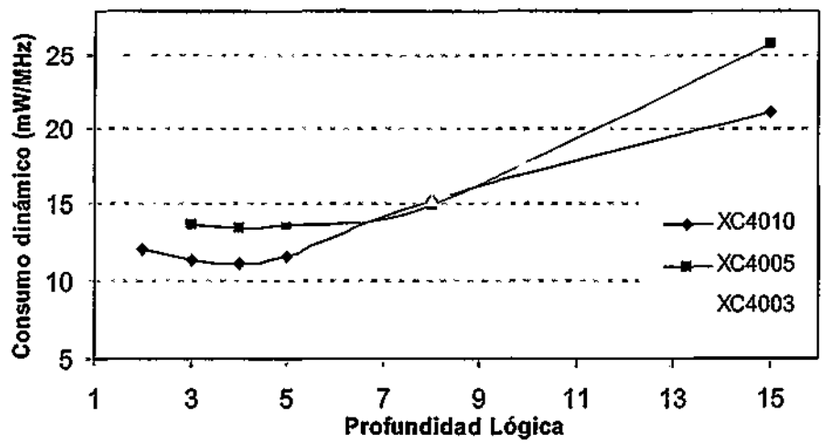


Figura 3.26. Consumo respecto de la profundidad lógica para diferentes dispositivos de la familia XC4K

Cabe mencionar que el aumento en el porcentaje de ocupación para un circuito suele ser contraproducente en términos de velocidad y consumo, ya que comienzan a ser

escasos los recursos y por ende el rutado suele ser más complejo y de peor performance. Obsérvese el caso del *Hatamian-4* cuya ocupación de la XC4003EPC84 es del 97% de los CLBs, allí la máxima velocidad de operación sufre un importante deterioro respecto a los demás circuitos reconfigurables, en tanto el consumo también se ve afectado.a.

La Figura 3.27 muestra la relación del consumo de sincronización y la ruta de datos respecto de la profundidad lógica para el dispositivo XC4010. Se observa que para la segmentación máxima (LD=1) la corriente de sincronización puede llegar a ser alrededor de un tercio del consumo dinámico total. Para el circuito totalmente combinacional la corriente de sincronización no es cero debido a la utilización de registros a la entrada y salida del circuito.

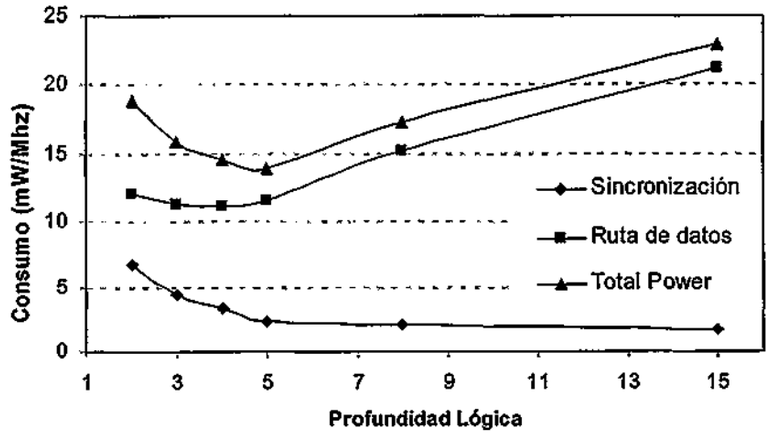


Figura 3.27. Relación del consumo de sincronización y de la ruta de datos en función de la profundidad lógica en XC4010.

3.4.2 Mediciones sobre Virtex

Se han construido multiplicadores tipo *shift & add* de 32 bits con distintos niveles de segmentación. La Tabla 3.16 muestra las principales características de los circuitos. Los circuitos son registrados a la entrada y salida, lo que agrega dos ciclos de reloj para obtener el resultado. El nivel de segmentación, se expresa como la latencia medida en ciclos de reloj menos dos.

Se utilizaron dos tipos de patrones de excitación, uno de actividad aleatoria y otro de máxima actividad. El consumo en función de la profundidad lógica se puede observar en la Figura 3.28. Aquí se puede ver que la segmentación no solo disminuye el camino crítico de manera prácticamente lineal, sino que también reduce el consumo casi linealmente. Solo una segmentación máxima (una LUT de profundidad lógica) aumenta levemente el consumo respecto de una segmentación menor (dos LUTs de profundidad lógica). El efecto que se podía observar en la familia 4K de un importante aumento de la potencia de sincronización, aquí tiene un peso relativo mucho menor.

Circuitos	Slices	Flip Flops	Prof. Lógica (LUTs)	Latencia (ciclos clk)	Min. period (ns)	Max freq (MHz)
mult_pipe32	545	128	32	2	123,4	8,1
mult_pipe16	578	224	16	3	67,0	14,9
mult_pipe12	611	320	12	4	53,5	18,7
mult_pipe11	611	320	11	4	48,8	20,5
mult_pipe10	644	416	10	5	45,6	21,9
mult_pipe8	644	416	8	5	39,7	25,2
mult_pipe7	677	512	7	6	36,1	27,7
mult_pipe6	710	608	6	7	32,7	30,5
mult_pipe5	743	704	5	8	28,6	35,0
mult_pipe4	776	800	4	9	26,6	37,5
mult_pipe3	875	1.088	3	12	21,3	47,0
mult_pipe2	1.040	1.568	2	17	18,7	53,4
mult_pipe1	1.584	3.104	1	33	15,6	64,2

Tabla 3.16. Principales características de los circuitos de prueba para Virtex

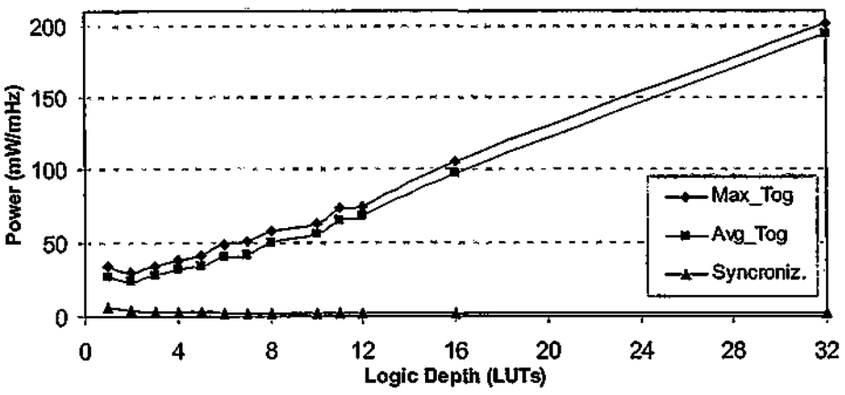


Figura 3.28. Consumo dinámico respecto de la profundidad lógica en Virtex

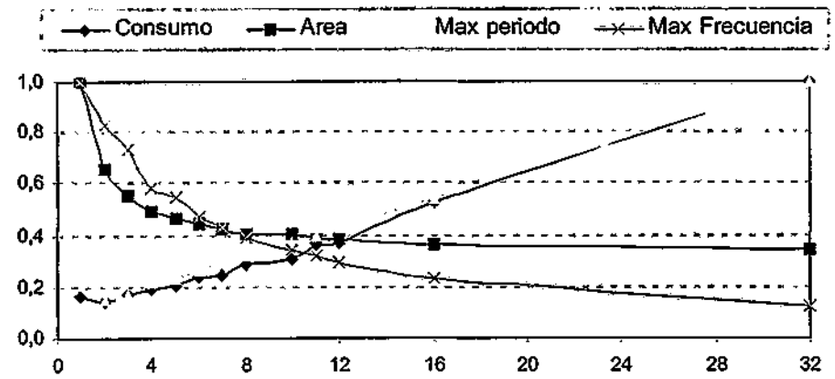


Figura 3.29. Consumo, área, retardo normalizados respecto de la profundidad lógica en Virtex

En la Figura 3.29 se puede ver el consumo, área, periodo y máxima frecuencia normalizados respecto de la profundidad lógica. Se puede observar el efecto de la segmentación, como al disminuir la profundidad lógica (aumentando la segmentación) decae el retardo del camino crítico casi de forma lineal, al igual que sucede con el consumo. El área total del circuito crece lentamente, dado que los *slices* poseen

registros que se reutilizan en la sincronización con un impacto muy débil en el área total. Cuando la cantidad de niveles de segmentación son muy grandes y por tanto la cantidad de registros utilizados mucho mayor que los *slices* utilizados para implementar la lógica, sí se nota el aumento de área total.

3.4.2.1 Actividad espuria (*glitches*)

Con el objeto de estimar cuantitativamente el aumento de la actividad producida por los *glitches* dentro de la ruta de datos, se han simulado algunos circuitos después del emplazado y rutado (*post & place and route simulation*), teniendo en cuenta además, los retardos internos de la FPGA (usando el fichero SDF - *Standard Delay Format* [Men03b]). Luego con herramientas internas del simulador Modelsim [Men03a], tales como *toggle add*, *toggle reset*, y *toggle report* [Men03c] se ha realizado un *script* para contar las transiciones necesarias y las espurias.

Para el multiplicador sin segmentación (*mult_pipe32*), la lista de conexiones (*netlist*) analizada posee 26167 nodos. Utilizando la secuencia de máxima actividad, se encuentra transiciones de estados en que la actividad espuria (*glitches*) llega a superar el 80,2 % de la actividad total.

3.4.2.2 Uso de herramienta de estimación de consumo

Se ha utilizado la herramienta XPOWER [Xpo02] (Apéndice F) para estimar el consumo de los multiplicadores descritos anteriormente. En la Tabla 3.17 se puede ver un resumen del consumo dinámico para los datos medidos respecto de los estimados tanto para estímulos de máximo movimiento (*max_tog*) como para estímulos de actividad aleatoria (*avg_tog*). La columna "sincro" muestra el consumo debido a los registros y al árbol de reloj (consumo de sincronización), en tanto que la siguiente el porcentaje de consumo sincronización respecto al consumo aleatorio. En la Figura 3.30 se observan la comparación de los datos de consumo medido respecto de los estimados.

En este caso se observa que la estimación con XPOWER no ofrece resultados demasiado precisos, sobre todo en la versión totalmente combinacional (*mult_pipe32*) donde la estimación ofrece valores hasta un tercio menores que los medidos. Las estimaciones para segmentaciones intermedias sobreestiman el consumo.

Circuito	Consumo Medido (mW/MHz)				Consumo Estimado (mW/MHz)			
	Max_tog	Avg_tog	Sincro	Sincro/ Avg_tog	Max_tog	Avg_tog	Sincro	Sincro/ Avg_tog
mult_pipe32	201,8	195,0	1,58	0,8 %	124,5	148,0	0,5	0,3 %
mult_pipe16	105,9	96,8	1,72	1,8 %	119,5	141,0	0,5	0,4 %
mult_pipe12	74,7	68,0	1,98	2,9 %	108,5	129,5	0,8	0,6 %
mult_pipe11	73,8	65,2	2,49	3,8 %	110,0	129,0	0,8	0,6 %
mult_pipe10	62,8	55,4	2,22	4,0 %	103,5	122,5	0,8	0,6 %
mult_pipe8	57,9	50,0	2,16	4,3 %	107,0	127,5	0,8	0,6 %
mult_pipe7	50,3	41,8	2,37	5,7 %	102,0	116,0	1,0	0,9 %
mult_pipe6	48,7	40,2	2,51	6,2 %	97,5	113,0	1,0	0,9 %
mult_pipe5	41,1	33,7	2,71	8,0 %	89,0	102,5	1,0	1,0 %
mult_pipe4	38,5	31,6	2,84	9,0 %	88,5	98,5	1,3	1,3 %
mult_pipe3	34,6	27,8	3,28	11,8 %	67,0	71,0	1,3	1,8 %
mult_pipe2	29,7	24,0	4,03	16,8 %	47,5	45,5	1,8	3,8 %
mult_pipe1	33,7	26,6	5,73	21,5 %	33,5	31,5	2,5	7,9 %

Tabla 3.17. Consumo medido respecto del estimado para multiplicadores segmentados.

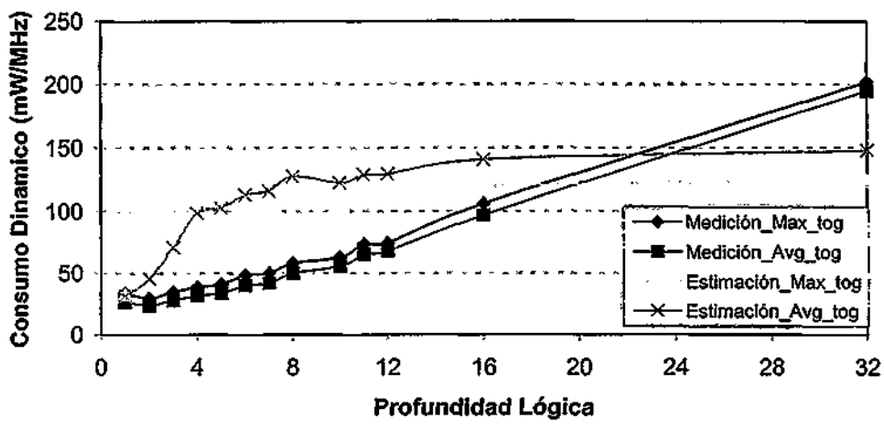


Figura 3.30. Consumo medido y estimado en función de la profundidad lógica.

3.4.3 Mediciones sobre Virtex II

Se han construido los mismos multiplicadores tipo *shift & add* de 32 bits con diferentes niveles de segmentación que anteriormente para los dispositivos Virtex. La Tabla 3.18 muestra las principales características de los circuitos. Al igual que en Virtex, los circuitos son registrados a la entrada y salida, lo que agrega dos ciclos de reloj para obtener el resultado.

Se utilizaron dos tipos de patrones de excitación, uno de actividad aleatoria y otro de máxima actividad. El consumo en función de la profundidad lógica se puede observar en la Figura 3.28, donde también se muestra el consumo de sincronización

La figura que relaciona la profundidad lógica con el consumo es prácticamente igual en Virtex y Virtex II. La profundidad lógica de mínimo consumo es igualmente de dos LUTs. También queda de manifiesto la baja influencia del consumo de sincronización sobre la segmentación en estas dos arquitecturas. Por último cabe destacar que el consumo en Virtex II se reduce respecto de Virtex II en un factor de entre 3 y 5 dependiendo de la profundidad lógica.

Circuitos	Slices	Flip Flops	Prof. Lógica (LUTs)	Latencia (ciclos clk)	Min. period (ns)	Max freq (MHz)
mult_v2_p32	585	167	32	2	97,2	10,3
mult_v2_p16	587	232	16	3	53,6	18,6
mult_v2_p12	618	326	12	4	40,5	24,7
mult_v2_p10	654	425	10	5	34,8	28,7
mult_v2_p8	653	424	8	5	29,6	33,8
mult_v2_p7	686	520	7	6	26,4	37,9
mult_v2_p6	721	618	6	7	22,9	43,6
mult_v2_p5	756	716	5	8	19,3	51,9
mult_v2_p4	793	816	4	9	16,1	62,1
mult_v2_p3	886	1098	3	12	14,8	67,6
mult_v2_p2	1057	1584	2	17	10,7	93,1
mult_v2_p1	1616	3136	1	33	7,9	127,3

Tabla 3.18. Principales características de los circuitos de prueba para Virtex II

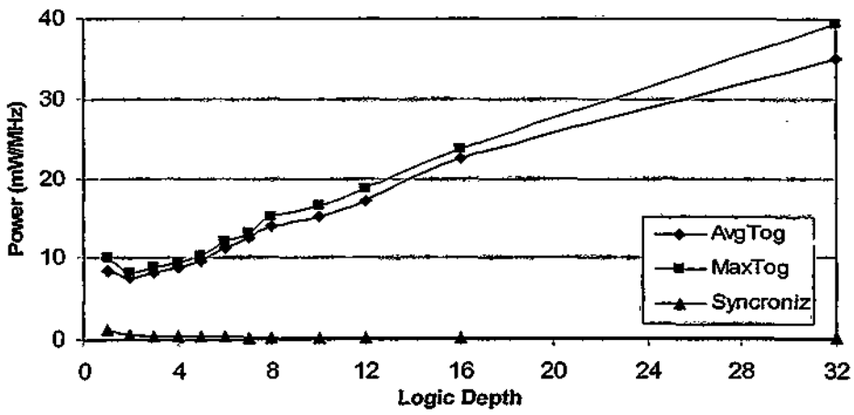


Figura 3.31. Consumo respecto de la profundidad lógica en Virtex II

3.4.4 Conclusiones sobre la segmentación

La segmentación (*pipeline*) se asocia al aumento de velocidad y en otras tecnologías de circuitos integrados posee un fuerte impacto en el área y el consumo. En las FPGAs gracias a los registros distribuidos por el circuito generalmente no suele degradar tanto el área, en tanto que, el consumo producto de la reducción de los *glitches* se mejora ostensiblemente.

La principal observación es que la segmentación disminuye notablemente el consumo, en el caso de la familia 4K una segmentación intermedia, 4-5 LUTs, es lo ideal, en tanto que para las familias Virtex y Virtex II una segmentación cercana al máximo posible es lo ideal. Los mínimos consumo se observan con dos LUTs en estas familias.

En los circuitos analizados en la familia 4K, tal como recomienda el fabricante, no es aconsejable tener un porcentaje de ocupación superior al 90% dado que la escasez de recursos hace degradar la calidad del rutado introduciendo mayores demoras y tal como se medio aquí, aumentando el consumo.

En la familia Virtex y Virtex II se ha observado la relación directa que posee la profundidad lógica con la degradación de velocidad y el consumo. De utilizar una versión totalmente combinacional a segmentar de manera óptima (para estos circuitos profundidad lógica 2 – LD = 2) se puede reducir el consumo dinámico a un 12% del original en Virtex y a un 24 % en Virtex II.

Por otra parte se ha puesto a prueba la herramienta de estimación de consumo Xpower [Xpo02] no obteniéndose resultados aceptables. Cabe destacar que esta herramienta esta en continuo desarrollo y se observan resultados diferentes de versión a versión (ver comentario F.6 en el apéndice F).

Por último destacar la diferencia que se observa entre Virtex y Virtex II. La mejor estructura en el rutado de Virtex-II hace que el aumento de actividad espuria por la profundidad lógica solo multiplique por cuatro el consumo respecto de LD=2, en tanto que en Virtex el consumo dinámico se multiplica por ocho. Relacionado con la evolución tecnológica, cabe destacar que el consumo se reduce en un factor de entre 3 y 5 en Virtex II respecto de Virtex dependiendo de la profundidad lógica.

3.5 Observaciones en la disminución del consumo registrando las entradas y salidas

El hecho de registrar tanto las entradas como las salidas provee una forma significativa de reducción de consumo a través de la eliminación de *glitches*. Por otra parte en sistemas que interactúan con otros dispositivos montados sobre un PCB donde las capacidades son mucho mayores, es altamente deseable eliminar estos pulsos espurios fuera del circuito.

3.5.1 Experiencias sobre la familia XC4K

Un primer experimento fue sobre el diseño de un multiplicador *guild* con sus salidas registradas en los pads de salida y sin alterar el placement and routing (a través del FPGA editor) quitar el registro en las patas. Esto produce un aumento del orden del 7% del consumo dinámico para vectores de máximo movimiento.

Luego se cambio la forma de registro de las entradas y salidas. En un principio se utilizaban registros en los IOBs (*Input Output Blocks*) y luego se comenzó a registrar dentro del circuito a través de los flip-flops que se posee dentro de los CLBs. Esta técnica no hizo aumentar la cantidad de CLBs, dado que los CLBs para el cálculo poseen Flip-Flops sin utilizar pero el consumo disminuyo notablemente. En un principio no parecía muy razonable pero la explicación fue la siguiente.

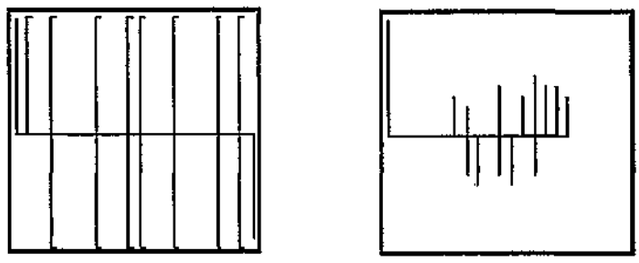


Figura 3.32. Distribución de la línea global de reloj. a)Flip-flop en lás patas de entrada salida; b) usando los Flip-Flop de los CLBs

El uso de pads con registro hace que el árbol de reloj se distribuya por toda la FPGA para llegar a toda la periferia donde se encuentran las patas aumentando notablemente la capacidad del árbol de reloj. Para el este caso el *fan-out* de la línea global de reloj es de 32 (16 entradas y 16 salidas) en tanto cuando se registran las entradas en los CLBs el *fan-out* de reloj es de solo 16, ya que cada CLB posee 2 registros y se usan conjuntamente. Además cada entrada posee un *fan-out* promedio de 16 CLBs lo que hace que las pistas utilizadas para llegar a los CLBs sean más largas que llegar al centro del circuito con una única señal y registrarlo allí mismo. En las figuras se puede observar este hecho. En la Figura 3.32 la línea de reloj necesaria para alcanzar todos los pads es claramente superior a la necesaria para alimentar a los 16 CLBs que registran señales, en la Figura 3.33 se muestra la distribución de uno de los bits de los operandos (concretamente a<7>) y se ve la diferencia en recursos utilizadas.

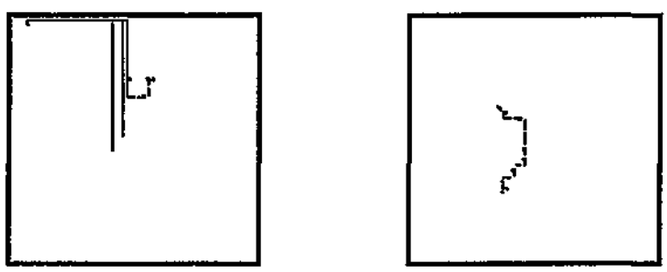


Figura 3.33. Distribución de la línea de entrada a<7> con *fan-out* de 16. a)Flip-flop en las patas de entrada salida; b) usando los Flip-Flop de los CLBs

En la Figura 3.34 se puede observar el consumo dinámico para diferentes alternativas, obsérvese que registrando dentro de los CLBs se logra el menor consumo. Registrar en los pads acarrea un aumento del consumo en alrededor de un 14% en tanto si no se lo registra puede consumir en el orden del 21% más.

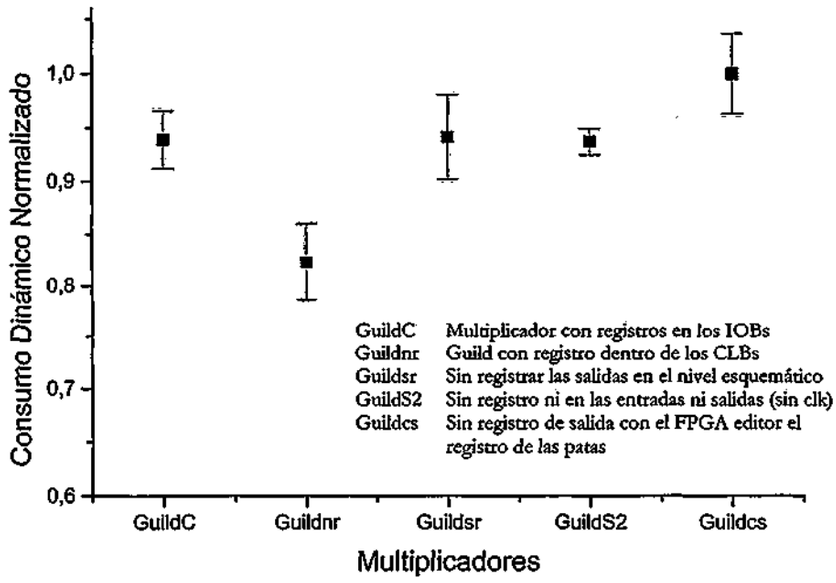


Figura 3.34. Consumo normalizado con registros en IOBs y CLBs.

Al estar registradas las salidas del generador de vectores de test y que tanto el generador como el circuito analizado compartan el reloj, hace que no tenga prácticamente influencia el hecho de quitar los registros a la entrada.

3.5.2 Experiencias sobre la familia Virtex

El efecto de los *glitches* al registrar en las patas o dentro de los *slices* se refuerza en el caso de la familia Virtex, dado que la alimentación del núcleo de la FPGA y la periferia son diferentes. Para el caso del circuito bajo prueba, un XCV800hq240, la tensión del núcleo es de 2,5 V y la periferia de 3,3V.

Los circuitos de prueba son multiplicadores *shif & add* de 32 bits registrados en las entradas y las salidas, estos circuitos tienen una gran profundidad lógica y consecuentemente una gran cantidad de *glitches* los que magnifican el efecto. En la Tabla 3.19 se puede observar la diferencia de utilizar los registros en los slices (Slice-FF), en los IOB de entrada (IOB-FF), quitando los registros de salida (No-FF-out), y por último quitando los registros de entrada también (No-FFs).

Tipo Secuencia	Circuito	I core (mA)	I perif (mA)	P Core (mW)	P Perif (mW)	Pot Total (mW)	Aumento consumo
Max_Tog	Slice-FF	296	13	740	43	783	--
	IOB-FF	300	108	750	356	1106	41%
	No-ff-out	305	219	763	723	1485	89%
	No-FFs	305	217	763	716	1479	88%
Avg_Tog	Slice-FF	249	12	623	40	662	--
	IOB-FF	251	85	628	281	908	37%
	No-ff-out	255	179	638	591	1228	85%
	No-FFs	254	175	635	578	1213	83%

Tabla 3.19. Diferencia de consumo dinámico dependiendo del tipo de registros en Virtex. Consumo dinámico a 5 MHz.

En la tabla se muestran los resultados para dos tipos de secuencia de entrada, por un lado una secuencia denominada de máximo movimiento (*Max_Tog*) y por otra una denominada de movimientos aleatorios (*Avg_Tog*). Para cada circuito se muestra la corriente dinámica del *core* y la periferia y el respectivo consumo (recordar que uno se alimenta a 2,5 y el otro a 3,3 V). En la última columna se muestra el aumento de consumo de cada alternativa respecto del uso de registros en los *slices*. El resultado de quitar los flip-flops a la entrada consume menos potencia que con ellos. Esto se justifica en el hecho que los datos de entrada producidos por el generador de patrones no poseen *glitches* en tanto que los registros consumen cierta potencia de sincronización.

3.5.3 Experiencias sobre la familia Virtex II

Al igual que en Virtex, el efecto de los *glitches* al registrar en las patas o dentro de los *slices* se refuerza en el caso de la familia Virtex II, puesto que en este caso también la alimentación del núcleo de la FPGA y la periferia son diferentes. Para el caso del circuito bajo prueba, un XC2V1500fg676-6, la tensión del núcleo es de 1,5 V y la periferia y tensión auxiliar son de 3,3V. Los circuitos de prueba son multiplicadores *shif & add* de 16 bits registrados en las entradas y las salidas.

La Tabla 3.20 muestra los resultados del consumo dinámico al utilizar registros en los slices (*Slice-FF*), en los IOB de entrada (*IOB-FF*), quitando los registros de salida (*No-FF-out*), y por último quitando los registros de entrada también (*No-FFs*).

Tipo Secuencia	Circuito	I core (mA)	I perif (mA)	P core (mW)	P perif (mW)	Pot total (mW)	Aumento consumo
Avg_Tog1	Slice-FF	15,9	9,1	23,9	30,0	53,9	--
	IOB-FF	19,2	9,0	28,8	29,7	58,5	9 %
	No-ff-out	21,5	24,0	32,3	79,2	111,5	107 %
	No-FFs	21,4	23,9	32,1	78,9	111,0	106 %
Avg_Tog2	Slice-FF	18,0	8,0	27,0	26,4	53,4	
	IOB-FF	21,2	8,0	31,8	26,4	58,2	9%
	No-ff-out	24,3	22,7	36,5	74,9	111,4	109%
	No-FFs	24,2	22,7	36,3	74,9	111,2	108%

Tabla 3.20. Diferencia de consumo dinámico dependiendo del tipo de registros en Virtex II. Consumo dinámico a 10 MHz.

En la tabla se muestran los resultados para dos tipos secuencia de entrada pseudoaletorias. Para cada circuito se muestra la corriente dinámica del *core* y la periferia y el respectivo consumo dinámico operando a 10 MHz. En la última columna se muestra el aumento de consumo de cada alternativa respecto del uso de registros en los *slices*. Al quitar los *flip-flops* a la entrada, el circuito consume menos potencia que con ellos. Esto se justifica en el hecho que los datos de entrada producidos por el generador de patrones no poseen *glitches* en tanto que los registros consumen cierta potencia de sincronización. Cabe destacar que la diferencia observada al pasar los *flip-flops* de los *slices* a los IOBs en Virtex, es mucho menor en el caso de Virtex II.

3.5.4 Conclusiones del registro de entradas y salidas

Esta sección tiene relación con el apartado anterior respecto de la segmentación y la consiguiente disminución del consumo producto de la reducción de *glitches*. La existencia de registros en los datos de salida tiene un fuerte impacto en el consumo, las capacidades internas del chip son del orden de los femto-faradios en tanto que los PCBs se pueden medir en pico-faradios, con la consiguiente implicación en el consumo debido a los *glitches*. Para los ejemplos medidos en la familia 4K existe un aumento del 21% quitando los registros, en tanto que en Virtex del orden del 90% y en Virtex II en torno al 110%

En el caso de la elección del tipo de *flip-flop* para registrar las salidas entre los disponibles en los slices/CLBs y los de los IOBs existe una gran diferencia desde el punto de vista del consumo. En el caso de la famila 4K para multiplicadores de 8 bits puede haber una diferencia del orden del 14 %, en tanto que en Virtex para multiplicadores de 32 bits y teniendo en cuenta el efecto que produce la diferencia de tensión de alimentación del *core* y la periferia puede llegar casi a ser un 40 % superior. Para Virtex II la influencia es algo menor, en torno al 9 %.

Los registros a la entrada no tuvieron gran relevancia en estas pruebas dado que los datos de entrada a la FPGA eran prácticamente carentes de movimientos espurios. En un diseño que interactué con el mundo real es altamente aconsejable de utilizarlo, no solo por la disminución de consumo sino también por la fiabilidad de los datos (evitar metaestabilidades, etc.).

3.6 Reducción del consumo por deshabilitación de partes inactivas del circuito.

La idea es simple, las partes del circuito que no están produciendo datos útiles pueden ser desconectadas o deshabilitadas. Esto se enmarca en el concepto de evitar derroches de la sección 2.3. Para este fin se pueden aplicar diferentes técnicas, las que se estudian y evaluan en ésta sección.

3.6.1 Introducción

En los circuitos síncronos tipo ASIC quizás la idea más conocida sea la inhabilitación del reloj (*gating clock*) [Ish95][Ben96][Lan97][She00][Don93]. De este modo el reloj no llega a la parte del circuito que se desea deshabilitar reduciendo el consumo de sincronización y bloqueando así mismo el funcionamiento de los registros.

Otra alternativa es bloquear los datos de entrada, de modo que al no cambiar los estímulos no se generarán transiciones en los nodos internos y por tanto se reducirá el consumo. Basados en estas ideas existen técnicas como el aislamiento de operadores (*operand isolation*) [Mün00][Syn03a], la pre-computación (*pre-computation*) [Ali94] ó *guarded evaluation* (evaluación cautelosa)[Tiv95].

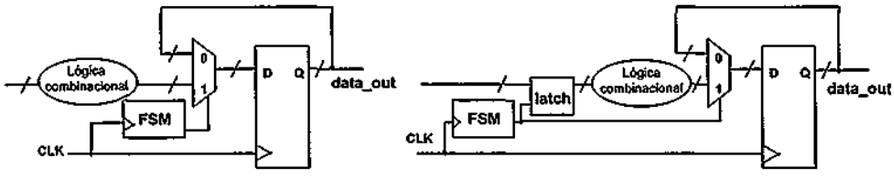


Figura 3.35. Ejemplo de la utilización de aislamiento de operandos

En el aislamiento de operandos los bloques combinacionales son deshabilitados cuando no se requiere su utilización (Figura 3.35). La herramienta para diseño de ASIC Power Compiler de Synopsys [Syn04] junto al Physical Compiler [Syn03a] implementa esta funcionalidad. Solo basta con declarar una directiva del sintetizador en el código HDL. En el caso de la precomputación la reducción del consumo se logra a través de bloquear los datos que no intervienen en el computo. En [Ali94] se

proponen varias arquitecturas y formas de cálculo de las funciones de deshabilitación. La técnica de *guarded evaluation* es similar a la anterior, utilizando para el bloqueo *latches*.

La estructura general de un del circuito (o parte de el) que aplica el bloqueo de datos tendrá una estructura como la que se muestra en la Figura 3.36. La generación de señales de bloqueo (*blocking signals*) depende del tipo de aplicación que se esté desarrollando. Esto puede ser máquina de estados o un circuito puramente combinacional, además puede o no depender de las entradas a los circuitos o ser simplemente ser una secuencia fija.

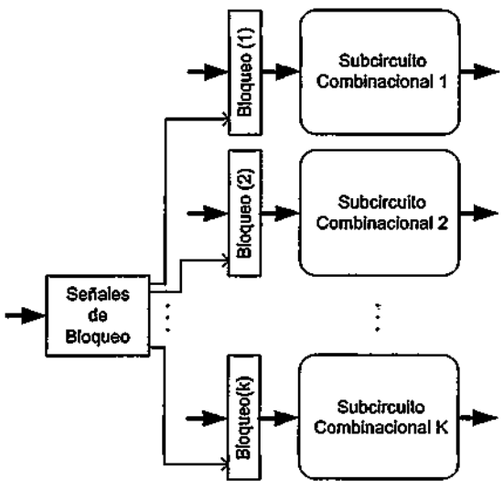


Figura 3.36. Esquema para la deshabilitación de partes del circuito

El esquema de la Figura 3.36 es fácilmente extensible a circuitos combinacionales, donde la señal de inhabilitación actuará (directa o indirectamente) sobre todos los registros del circuito. A continuación se analizan las diferentes alternativas para el bloqueo de los datos.

3.6.2 Técnicas de inhabilitación en FPGAs

A continuación se describen aquellas técnicas de inhabilitación de datos que mejor se adaptan al marco tecnológico. Estas técnicas son triviales de aplicar y no representan más que unas pocas líneas de código HDL para llevarlas a cabo.

3.6.2.1 Bloqueo por inhabilitación del reloj

En los circuitos digitales síncronos implementados en FPGA, la potencia de sincronización puede representar una parte considerable del consumo total (sección 3.1.3). Una técnica ampliamente utilizada y difundida en el diseño ASIC es la conocida como inhabilitación del reloj o *gated clock*. Se han publicado importantes reducciones del consumo con esta técnica [Ish95][Ben96][Mon99]. Pero por otra parte, Xilinx recomienda contundentemente no utilizar esta técnica en FPGAs [Xil03a]. La razón es la posible aparición de *glitches* sobre la señal de reloj que actúa sobre el FF por no utilizar los canales globales de rutado (Figura 3.37). Más aun, existen herramientas comerciales de síntesis que automáticamente transforman el *gated clock* en una estructura orientada a las FPGAs [Syn03b].

En los dispositivos Virtex II, II Pro, y Spartan-3, existe un tipo de *buffer* multiplexor de reloj global denominado BUFGMUX [Xil03b] que permite cambiar entre dos relojes sin *glitches*. Esto permite diseñar circuito que funcionen con dos relojes diferentes. Si una de las entradas del componente BUFGMUX es fija a 0 (ó 1) esto se transforma en un *buffer* global de reloj con *clock enable* (BUFGE). Esto permite implementar *gated clock* sobre todo un árbol de reloj. Cabe recordar que los dispositivos Virtex II poseen 8 árboles de reloj globales por cuarto de la FPGA [Xil04b].

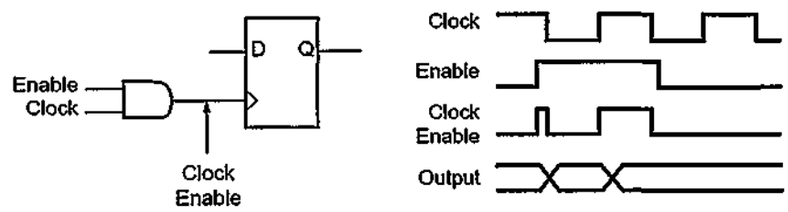


Figura 3.37. Riesgo de doble captura en la implementación de gated clock en FPGAs [Xil03a].

3.6.2.2 Bloqueo de datos mediante la señal de habilitación

Es el uso aconsejado por Xilinx [Xil03a] para implementar el comportamiento del *gated clock* sin riesgo de dobles capturas originadas por los *glitches* que se puedan originar en la línea de reloj (Figura 3.37 y Figura 3.38). Estos dispositivos FDCE (*flip*

flop tipo D con *chip enable*) está disponible en todos los dispositivos de Xilinx desde XC3K en adelante.

Entre las ventajas más importantes es la forma simple de generar estos elementos en cualquier diseño, así como la pequeña influencia en el área dado que los registros con CE están distribuidos en toda la lógica (2 FF por *slice*). La penalidad en el retardo es muy baja, ya que es el rutado más el tiempo de captura del FF. La desventaja desde el punto de vista del consumo es que mientras no se está capturando datos, el árbol de reloj sigue funcionando y consecuentemente consumiendo.

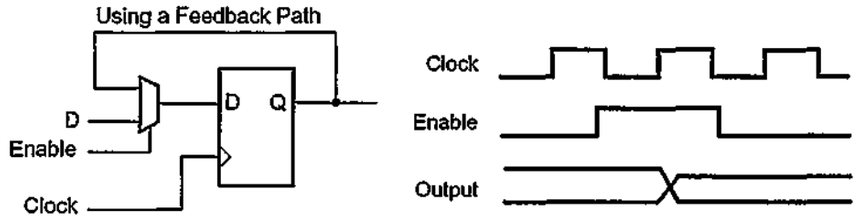


Figura 3.38. Forma de implementar el bloqueo de datos equivalente al gated clock según Xilinx [Xil03a].

3.6.2.3 Bloqueo de datos con latches

La utilización de registros activos por nivel (*latches*) en vez de activos por flanco posee la ventaja que la señal de reloj solo se utiliza cada vez que se cambia de modo transparente a modo captura. Si como es de esperar esta frecuencia es mucho menor que la del reloj, también se estará ahorrando consumo de sincronización. La utilización de *latches* también es atractivo dado que es aplicable a circuitos totalmente combinacionales.

Los registros en los *slices* pueden ser configurados tanto como FF o *latches* en Virtex, Virtex II/II-Pro, Spartan II/III, no así en la familias más antiguas XC4K/Spartan donde la forma de lograr *latches* es utilizando LUTs configuradas como memorias.

La penalidad en área es pequeña por la abundancia de registros distribuidos en la FPGA, en tanto la penalidad en tiempo de un *latch* en modo transparente es del mismo orden de magnitud que atravesar una LUT.

3.6.2.4 Bloqueo de datos con puertas lógicas

Otra alternativa para el bloqueo de los datos es la utilización de puertas AND (u OR). De este modo la señal de habilitación a cero (respectivamente a uno) hará que todas las entradas se mantengan a cero (respectivamente a uno) independientemente del valor de entrada original (Figura 3.39).

Esta técnica puede generar transiciones espurias cada vez que se inhabilita o habilita el circuito. Por ejemplo una entrada que no cambia, y cuyo valor es uno (respectivamente cero) bloqueada con una puerta AND (respectivamente OR), si se genera una habilitación y una deshabilitación provocará dos transiciones innecesarias.

En el entorno de FPGAs puede ser interesante está técnica puesto que, puede no requerir área extra ni aumentar el retardo, si se logran intercalar la puerta AND (u OR) dentro de las LUTs que implementan la lógica. Otro punto a favor de esta técnica es que funciona en circuitos totalmente combinacionales.

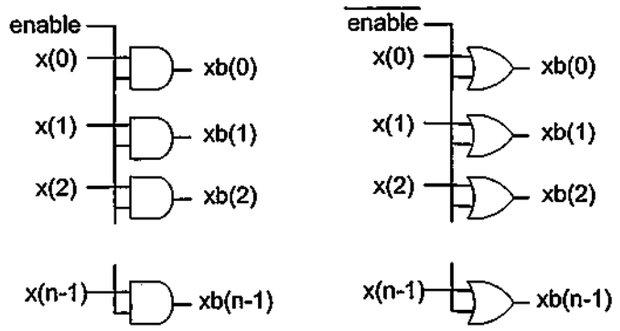


Figura 3.39. Inhabilitación de las entradas con puertas. a. puertas AND. b. Puertas OR.

3.6.2.5 Uso de *buffers* de tercer estado

Esta alternativa tiene varios inconvenientes en FPGA. El primero es el mayor consumo asociado a la desconexión y conexión de las señales. Adicionalmente agregan un retardo bastante más importante que las alternativas reseñadas en los puntos anteriores y por último no todas las FPGAs disponen de abundantes *buffers* de tercer estado. Por ejemplo la línea de bajo coste Spartan 3 [Xil04c] solo tiene *buffers* de tercer estado en las patas de salida y no dentro del arreglo de slices.

3.6.3 Resultados de las técnicas de deshabilitación

Para evaluar las técnicas de deshabilitación se ha construido un circuito con dos bloques aritméticos (multiplicadores), más una lógica de selección que utiliza el resultado de un bloque u otro (Figura 3.41). Luego se aplicaron las diferentes técnicas expuestas anteriormente con el fin de evaluar la disminución del consumo. La Figura 3.41 muestra el ejemplo de la alternativa de utilizar señales de inhabilitación CE (sección 3.6.2.2). La ruta de datos usa operandos de 16 bits y para la medición del consumo se utilizo una secuencia aleatoria de entrada.

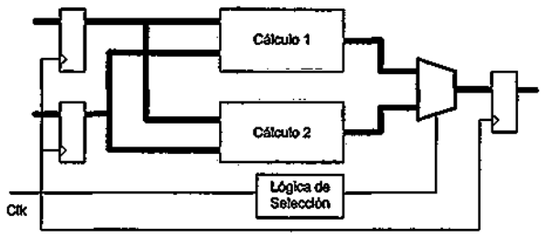


Figura 3.40. Circuito original para la evaluación de las técnicas de deshabilitación de partes inactivas.

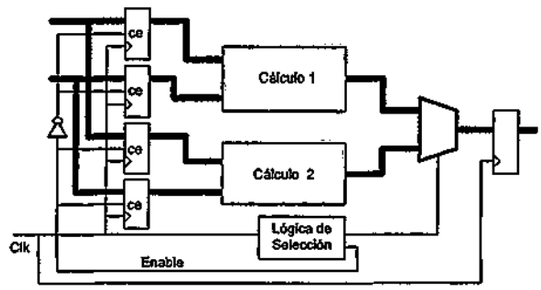


Figura 3.41. Modificación del circuito para la evaluación de las técnicas de deshabilitación de partes inactivas utilizando inhabilitación con CE.

Para la familia Virtex se utiliza el dispositivo XCV800hq240-6, en tanto que para Virtex II un arreglo XC2V1500fg676-6. La Tabla 3.21 resume los resultados para el dispositivo de la familia Virtex, en tanto la Tabla 3.22 lo hace para el dispositivo de la

familia Virtex II. Los circuitos expuestos en las tablas son los descritos anteriormente, es decir: bloqueo de datos por señal de habilitación (*clk_en*) de la sección 3.6.2.2; bloqueo de datos con latches (*latch*) de la sección 3.6.2.3; bloqueo por inhabilitación del reloj (*gate_clk*) de la sección 3.6.2.1; bloqueo de datos con puertas lógicas (*ands, ors*) de la sección 3.6.2.4, y por último uso de *buffers* de tercer estado (*tbufs*) de la sección 3.6.2.5.

Circuito	Área					Retardo (ns)		Consumo(mW/MHz)	
	slices	4-LUT	regist.	latch	aumento	periodo	aumento	dinámico	Ahorro
normal	334	597	81	-	0,0%	24,0	0,0	17,5	-
clk_ce	342	597	104	-	2,4%	24,4	0,4	9,7	45,7 %
latch	357	597	136	64	6,9%	22,8	-1,2	10,8	38,6 %
gate_clk	342	599	103	-	2,4%	31,6	7,6	10,0	42,9 %
ands	357	662	75	-	6,9%	25,0	1,0	9,9	44,9 %
ors	356	661	74	-	6,6%	24,3	0,3	10,3	41,4 %
tbufs	324	597	71	-	-3,0%	30,8	6,8	11,8	32,9 %

Tabla 3.21. Resultados para la deshabilitación de circuitos en Virtex.

Los resultados expuestos en las tablas muestran una muy baja influencia en área, muy poco impacto en retardo e importantes reducciones de consumo para todas las alternativas. La técnica más atractiva y simple de aplicar es la de utilizar la señal de inhabilitación CE (*clk_ce*). La técnica de *gated clock* en Virtex II ofrece buenos resultados utilizando el componente específico BUFGMUX, aunque la penalidad en el retardo no es despreciable. La alternativa de utilizar puertas ANDs es otra alternativa simple de utilizar, que además reporta buenos resultados. Por último la alternativa de los *buffers* de tercer estado, no solo degrada el camino crítico fuertemente, sino que ofrece los peores resultados en cuanto al ahorro de consumo.

Circuito	Área					Retardo (ns)		Consumo(mW/MHz)	
	slices	4-LUT	regist.	latch	aumento	periodo	aumento	dinámico	Ahorro
normal	327	596	74	-	0,0%	16,7	0,0	4,5	-
clk_ce	342	596	104	-	4,6%	16,2	-0,5	2,4	46,7 %
latch	358	598	136	64	9,5%	16,2	-0,5	2,9	36,7 %
gate_clk	342	596	104	-	4,6%	16,2	-0,5	2,6	43,3 %
ands	355	660	74	-	8,6%	16,6	-0,2	2,7	40,0 %
ors	355	660	74	-	8,6%	16,6	-0,2	3,0	33,3 %
tbufs	324	596	71	-	-0,9%	17,9	1,1	3,0	33,3 %

Tabla 3.22. Resultados para la deshabilitación de circuitos en VirtexII.

3.6.4 Conclusiones de las técnicas de deshabilitación

Dejar que partes inactivas del circuito generen consumo, es claramente un derroche que se debe evitar. Las técnicas descritas en las secciones 3.6.2 fueron evaluadas obteniendo importantes reducciones de consumo.

La técnica más simple de implementar y que mejores resultados ofrece tanto en Virtex como Virtex II es la inhabilitación de registros con la señal CE. La alternativa de utilizar puertas ANDs es otra alternativa simple de utilizar, que además reporta buenos resultados.

La técnica de *gated clock* en Virtex II ofrece buenos resultados en la reducción de consumo utilizando el componente específico BUFGMUX, aunque la penalidad en el retardo no es despreciable. Esta técnica gana relevancia cuando se tiene largos periodos de inactividad, ya que elimina también el consumo de sincronización.

Por último, la alternativa de los *buffers* de tercer estado, no solo degrada el camino crítico fuertemente, sino que ofrece los peores resultados en cuanto al ahorro de consumo.

Debido a la gran cantidad de registros distribuidos en la FPGA, la implementación de estas técnicas de inhabilitación de partes inactivas tiene un escaso impacto en el área y la velocidad.

3.7 Conclusiones y recomendaciones a nivel diseñador

Este capítulo examinó diferentes experimentos llevados cabo sobre el consumo en FPGAs con el objeto de determinar relaciones en el consumo, y "consejos" a nivel diseñador. Aquí se ha examinado la relación de la tensión y la frecuencia en el consumo, la influencia de la corriente estática y la corriente de sincronización, la relación velocidad-consumo, la conmutatividad de datos, el efecto del *pipeline* y el uso de registros en general. Finalmente se analizaron y cuantificaron las alternativas para la inhabilitación de partes inactivas de un circuito.

Se han presentado experimentos sobre dispositivos XC4000, Virtex y Virtex II. Estos dispositivos comparten la estructura básica de LUTs de 4 entradas, aunque tecnológicamente han sufrido una gran evolución. La mayor parte de los experimentos muestran que las técnicas propuestas son independientes del dispositivo utilizado, es decir las conclusiones para una familia de dispositivos es aplicable a otras. A continuación se detallan los puntos más importantes del capítulo:

De la relación de la tensión y la frecuencia en el consumo:

- Se han corroborado la linealidad del consumo con la frecuencia y su dependencia cuadrática con la tensión en las tres familias de dispositivos estudiadas. Si bien esta es una conclusión obvia (sigue la ecuación 2.4), es indispensable corroborar este hecho experimentalmente como punto de partida de las futuras mediciones. Existen trabajos en el marco tecnológico de las FPGA que sugieren una dependencia de tercer orden de Vcc [Gar00] la que no ha sido corroborada en estas mediciones.

De la corriente estática en el consumo:

- Se ha visto un fuerte incremento en la corriente estática en las tecnologías más modernas. En los dispositivos de la familia XC4K y Virtex la corriente de estática son del mismo orden de magnitud a pesar que el dispositivo Virtex tiene una capacidad unas ochenta veces mayor. El salto a la tecnología Virtex II conlleva un incremento del consumo estático, aunque no tanto como lo predicho por las herramientas de estimación de consumo.

Del consumo de sincronización:

- Claramente se observa una menor influencia relativa de esta componente en los dispositivos más modernos. En XC4K cada flip-flop extra implica un consumo de unos 30 μW/MHz, en tanto que en Virtex se reduce hasta 1,5 μW/MHz y para Virtex II, puede estar por debajo de 0,1 μW/MHz.

- A su vez desde el punto de vista de la relación consumo sincronización sobre consumo dinámico total se observa una menor influencia en los dispositivos más modernos. Por ejemplo en un multiplicador con máxima segmentación en XC4K la corriente de sincronización representa más del 30 %, en tanto que Virtex es un 17 % y en Virtex II se reduce hasta un 12 %.

De la relación velocidad-consumo:

- Para una topología dada, el circuito con mayor máxima frecuencia de funcionamiento normalmente implica el mejor circuito en término de consumo. Esta optimización se puede obtener de manera gratuita, por ejemplo usando emplazado y rutado repetitivo (*RPR - Repetitivo Placement & Routing*) o ajustando las opciones de optimización.

- No obstante, si el diseñador debe elegir entre diferentes topologías, ni la velocidad de reloj, ni el área son parámetros por si solos validos para predecir el ahorro de consumo.

- La cuenta de las transiciones espurias que brinda un simulador convencional, se puede utilizar como métrica indirecta para analizar que circuito consume menos dentro de una misma topología.

Del efecto de la conmutatividad:

- Se prueba a través de diferentes ejemplos que bajo ciertas condiciones el solo hecho de permutar las entradas tiene un fuerte impacto en la reducción de consumo del consumo debido al cálculo. En la familia XC4K se utilizaron multiplicadores de 8 bits llegando la diferencia hasta un 8 %. En la familia Virtex con multiplicadores de 16 y 32 bits se llega a una diferencia de hasta un 39%.

- Las componentes de este desbalance en el consumo puede ser un diseño irregular o la implementación irregular que se genera al mapear un diseño en una FPGA, esto produce que los *glitches* generados en una secuencia de operaciones no sean iguales al permutar las entradas. Otra componente es el uso de líneas globales (que poseen mayor capacidad) para uno de los operandos, lo que puede generar diferencias en el consumo. Por ello, si la secuencia de valores a operar no es aleatoria o la frecuencia de uno de los operandos es diferente al otro (multiplicación de matrices, operaciones sobre video, filtros, etc.) conviene analizar el orden de los operandos.

- Cabe esperar que otros bloques combinacionales cuyas entradas sean conmutativas tiendan a tener esta característica de desbalance en el consumo. El uso de herramientas de estimación del consumo (y aun solo de la actividad) puede ser de utilidad a la hora de elegir la mejor el orden de entrada de los operandos para estos casos.

De la Segmentación:

- La principal observación es que la segmentación disminuye notablemente el consumo, en el caso de la familia 4K una segmentación intermedia (4-5 LUTs) es lo ideal, en tanto que para la familias Virtex y Virtex II una segmentación cercana al máximo posible resulta ser lo ideal.

- En las FPGAs gracias a los registros distribuidos por todo el dispositivo la aplicación de la segmentación no suele degradar demasiado el área, en tanto que, el consumo producto de la reducción de los movimientos espurios *(glitches)* se mejora ostensiblemente.

- En las familias Virtex y Virtex II se ha observado la relación directa que posee la profundidad lógica con la degradación de velocidad y el consumo. De utilizar una versión totalmente combinacional a segmentar de manera óptima (para estos circuitos - profundidad lógica 2) se puede reducir el consumo dinámico hasta un 12% del original en la familia Virtex y a un 24 % en la familia Virtex II.

De los registros a la Entrada y Salida:

- La existencia de registros en los datos de salida tiene un fuerte impacto en el consumo, las capacidades internas del chip son del orden de los femto-faradios en tanto que los PCBs se pueden medir en pico-faradios, con la consiguiente implicación en el consumo de los *glitches*. Para los ejemplos medidos en la familia 4K existe un aumento del 21 % quitando los registros, en tanto que en Virtex del orden del 90 % y en el dispositivo Virtex II estudiando supera el 110 %.

- En el caso de la elección del tipo de flip-flop la registrar las salidas entre los disponibles en los slices/CLBs y los de los IOBs existe una gran diferencia desde el punto de vista del consumo. En el caso de la famila 4K para multiplicadores de 8 bits puede haber una diferencia del orden del 14 %, en tanto que en Virtex para multiplicadores de 32 bits y teniendo en cuenta el efecto que produce la diferencia de tensión de alimentación del *core* y la periferia puede llegar casi a ser un 40 % superior.

De la inhabilitación de partes inactivas del circuito:

- Dejar que partes inactivas del circuito generen consumo, es claramente un derroche que se debe evitar. Las técnicas de inhabilitación descritas en las secciones 3.6.2 son fácilmente aplicables en el marco tecnológico

- La técnica más simple de implementar y que mejores resultados ofrece tanto en Virtex como Virtex II es la inhabilitación de registros con la señal CE. La alternativa de utilizar puertas ANDs es otra alternativa simple de utilizar, que además reporta buenos resultados.

- La técnica de *gated clock* en Virtex II ofrece buenos resultados en la reducción de consumo utilizando el componente específico BUFGMUX, aunque la penalidad en el retardo no es despreciable. Esta técnica gana relevancia cuando se tiene largos periodos de inactividad, ya que elimina también el consumo de sincronización.

- La alternativa de los *buffers* de tercer estado, no solo degrada el camino crítico fuertemente, sino que ofrece los peores resultados en cuanto al ahorro de consumo.

- Debido a la gran cantidad de registros distribuidos en la FPGA, la implementación de estas técnicas de inhabilitación de partes inactivas tiene un escaso impacto en el área y la velocidad.

3.8 Referencias del Capítulo

[Alc00] J. Alcalde, J. Rius and J. Figueras, "Experimental techniques to measure current, power and energy in CMOS integrated circuits", *Proceedings of XV Conference on Design of Circuit and Integrated Systems (DCIS'2000).* Montpellier, France, November 2000.

[Ali94] M. Alidina, J. Monteiro, S. Devadas, A. Gosh, M. Papaefthymiou, "Precomputation-Based sequential logic optimization for Low-Power", *IEEE Very Large Scale Integration (VLSI) Systems Journal* , Vol. 2, num.4, pp.426-435, December 1994.

[And04] J. H. Anderson, F. N. Najm, T. Tuan, "Active Leakage Power Optimization for FPGAs" *Twelfth International Symposium on Field Programmable Gate Arrays (FPGA 2004)* pp.33 - 41, Monterrey, California, Feb. 2004

[Ben96] L.Benini P.Siegel and G.De Micheli. "Automatic synthesis of low-power gated-clock finite-state machines". *IEEE Transaction on Computer Aided Design of Integrated Circuit*, vol.15, Issue 6, pp. 630– 643, June 1996.

[Boe96] E. Boemo, "Contribution to the Design of Fine-grain Pipelined VLSI Arrays", *Tesis doctoral ETSI Telecomunicación, Universidad Politécnica de Madrid*, Enero 1996.

[Boe98] E. Boemo, S. Lopez-Buedo, C. Santos, J. Jauregui and J. Meneses, "Logic Depth and Power Consumption: A Comparative Study Between Standard Cells and FPGAs", *Proc. XIII DCIS Conference (Design of Circuit and Integrated Systems)*, Madrid, Universidad Carlos III: November 1998.

[Cha92] A. Chandrakasan, S. Sheng and R. Brodersen, "Low-Power CMOS Digital Design", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 4, pp. 473-484. April 1992

[Don03] Monica Donno, Alessandro Ivaldi, Luca Benini, Enrico Macii. "Clock-Tree Power Optimization based on RTL Clock-Gating". *Proceedings of 40th ACM Design Automation conference (DAC 2003)*, Anaheim, California, USA. June 2003.

[Eto03] Emi Eto and Lyman Lewis, "Local Clocking Resourses in Virtex II Devices" *Xilinx Application Note Xapp609 (V1.0).* January 2003.

[Gar99] A. Garcia, W. Burleson and J. Danger, "Power Consumption Model of Field Programmable Gate Arrays", *Proceedings of FPL'99, in LNCS.* Springer-Verlag, September 1999.

[Gol93] H. Goldstine, "The Computer. From Pascal to von Newman", *Princeton University Press*: New Jersey, 1993.

[Gui69] H. H. Guild, "Fully Iterative Fast Array for Binary Multiplication and Addition", *Electronic Letters*, pp.263, Vol.5, N°12, June 1969.

[Hat86] M. Hatamian and G.L.Cash. "A 70-MHz 8-bit x 8 bit Parallel Pipelined Multiplier in 2.5-um CMOS". *IEEE Journal of Solid-State Circuits*, August 1986.

[Ise01] ISE Xilinx 4, "ISE 4 Release Notes and Installation Guide - 0401965", 2001, available at www.xilinx.com

[Ish95] Tohru Ishihara, Hiroto Yasuura. "Some Experimental Results on Low Power Design with Gated Clock", *IPSJ SIGNotes Design Automation Abstract* No.078, 1995.

[Kao02] J. Kao, S. Narendra, and A. Chandrakasan. Subthreshold leakage modeling and reduction techniques. *In IEEE International Conference on Computer-Aided Design*, pages 141–148, 2002.

[Lan97] T. Lang, E. Musoll, and J. Cortadella, "Individual Flip-Flops with Gated Clocks for Low Power Datapaths", *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, Vol.44, No. 6, June 1997.

[Lei95] J. Leiten, J. van Meerbegen and J. Jess, "Analisys and Reduction of Glitches in Synchronous Networks", *Proceedings of European Design and Test Conference. ED&TC 1995*, pp.1461-1464. New York: IEEE Press, 1995.

[Liu04] Michael Liu, Wei-Shen Wang, and Michael Orshansky, "Leakage Power Reduction by Dual-Vth Designs Under Probabilistic Analysis of Vth Variation" *International Symposium on Low Power Desing (ISLPED'04)*, Newport Beach, California, USA. , August, 2004.

[Mee95] J. Leijten, J. Meerbengen, and J. Jess, "Analysis of Reduction of Glitches in Synchronous Networks", *Proceedings of IEEE European Design and Automation Conf (EDAC95)*, IEEE Press, 1995.

[Men02] Mentor Graphics, "LeonardoSpectrum Bookcase v2002a", 2002. www.mentor.com

[Men03a] Mentor Graphics, "ModelSim SE User's Manual Version 5.7d", www.mentor.com, May 2003.

[Men03b] Mentor Graphics, "ModelSim SE User's Manual Version 5.7d: Chapter 14 - Standard Delay Format (SDF) Timing Annotation", May 2003.

[Men03c] Mentor Graphics, "ModelSim SE Command Reference Version 5.7d: toggle Add", May 2003.

[Men99] L. Mengíbar, M. García, D. Martín, and L. Entrena, "Experiments in FPGA Characterization for Low-power Design", *Proceedings of IV Conference on Design of Circuits and Integrated Systems (DCIS '99)*, Palma de Mallorca, Spain 1999.

[Mod02] Model Technologies, "Modelsim SE user Manual", October 2002, available at www.model.com

[Mon99] J. C. Monteiro, "Power optimization using dynamic power management", In *Procedding of the XII Symposium on Integrated Circuits and Systems Design (ICSD)*, pp. 134-139, Sep. 1999.

[Mün00] M. Münch, B.Wurth, R. Mehra, J. Sproch, and N. Wehn. "Automating RT-Level Operand Isolation to Minimize Power Consumption in Datapaths". *Design, Automation, and Test in Europe (DATE 2000)*, Paris, France, March 2000.

[Mus95] E. Mussol and J. Cortadella, "Low-Power Array Multiplier with Transition-Retaining Barriers", *Proc. PATMOS'95, Fifth Int. Workshop*, pp. 227-235, Oldenburg, October 1995.

[Naj94] F. Najm, "A Survey of Power Estimation Techniques in VLSI Circuits", *IEEE Transaction on VLSI Systems*, Vol. 2, number 4, pp. 446-455. December 1994.

[Nar03] S. Narendra, D. Blaauw, A. Devgan, F. Najm, "Leakage issues in IC design: trends, estimation, and avoidance," *Proc. of International Conference on Computer Aided Design (ICCAD)*, 2003.

[Ped96] Massoud Pedram, "Power Minimization in IC Design: Principles and Applications", *ACM Transaction on Design Automation of Electronic Systems*, Vol 1, No 1, Jan 1996.

[Ped97] Massoud Pedram, "Design Technologies for Low Power VLSI", in *Encyclopedia of Computer Science and Technology, vo 36, Marcel Dekker, Inc.* pp 73-96, 1997.

[Rab96] Jan M. Rabaey, "Exploring the Power Dimension". *In Proceeding of Custom Integrated Circuit Conference*, pp 215-220. May 1996.

[Rag96] Anand Raghunathan, Sujit Dey and Niraj K. Jha, "Glitch Analysis and Reduction in Register Transfer Level Power Optimization", *33rd ACM Design Automation Conference (DAC 96)*, Las Vegas, Nevada.1996.

[Sak98] T. Sakuta, W. Lee and P. Balsara, "Delay Balanced Multipliers for Low-Power/Low-Voltage DSP Cores", *in "Low-Power CMOS Design", A. Chandrakasan and R. Brodersen (Eds.), IEEE Press*, 1998.

[She00] R. Shelar, H. Narayanan, and M. Desai, "Orthogonal Partitioning and Gated Clock Architecture for Low Power Realization of FSMs", *IEEE International ASIC / SOC conference*, Sep 2000, pp. 266-270.

[She92] A. Shen, A. Gosh, S. Devadas y K. Keutzer, "On average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks", *Proceeding of International Conference on Computer Aided Design (ICCAD-92)*, pp.402-407. IEEE Press, 1992.

[Syn01] Synopsis Inc, "FPGA Express 3.6.1 User Guide", Agosto 2001. available at www.synopsis.com/fpga/

[Syn02] Synpicity Inc; "Synplify Pro 7.1 Online Documentation", April 2002. Available at www.synplicity.com

[Syn03] Synplicity inc., "Gated Clock Conversion with Synplicity". *Syndicated, Vol 2 Issue 4*, 2003 available at www.synplicity.com

[Syn03a] Synopsys inc. "Power Compiler Physical Compiler Data Sheet: Low power Synthesis with Power Compiler", 2003. available at www.synopsys.com

[Syn04] Synopsys, inc. "Power Compiler Data Sheet: Automatic Power Management within Galaxy™ Design Platform", 2004. available at www.synopsys.com

[Syn99] Synopsis inc, FPGA Express User Guide version 2.3, 1999. FPGA Express home page; http://www.synopsys.com/products/ fpga/fpga_express.htm

[Tek02] Tektronix inc., "TLA 700 Series Logic Analyzer User Manual", available at http://www.tektronix.com, 2002.

[Tiw95] Vivek Tiwari, Sharad Malik, and Pranav Ashar. "Guarded evaluation: pushing power management to logic synthesis/design", *International Symposium on Low Power Electronics and Design.* pp. 221-226, California, USA, 1995

[Tod00] E. Todorovich, G. Sutter, N. Acosta E. Boemo and S. López-Buedo, "End-user low-power alternatives at topological and physical levels. Some examples on FPGAs", *XV Conference on Design of Circuits and Integrated Systems (DCIS 2000)*, Le Corum, Montpellier, France, November 21-24, 2000.

[Tod02] E. Todorovich, M. Gilabert, G. Sutter, S. Lopez-Buedo, and E. Boemo, " A Tool for Activity Estimation in FPGAs", *Lecture Notes in Computer Science, Vol.2438, pp.340-349.* Berlin: Springer-Verlag, 2002.

[Wall64] C. Wallace, "A Suggestion for a Fast Multiplier". *IEEE Transaction on Electronic Computers, pp.14-17*, February 1964.

[Wil04] Steven Wilton, Su-Shin Ang, Wayne Luk, "The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays", *Proceedings of 1th Field-Programmable Logic and Applications (FPL'04), LNCS 3203*, pp. 719-718. Antwerp, Belgium. August/September 2004.

[Xil00a] Xilinx inc, "Software Manual on line: Synthesis and Simulation Design Guide 3.1i" available at http://support.xilinx.com, 2000.

[Xil00b] Xilinx corp, "Software Manual on line: CORE generator Guide 3.1" available at http://support.xilinx.com, 2000.

[Xil02a] Xilinx Inc; "Xilinx Synthesis Technology (XST) User Guide", available at www.xilinx.com, 2002.

[Xil02b] Xilinx Inc; "Core Generator Guide – ISE 5", available at www.xilinx.com, 2002.

[Xil03a] Xilinx inc, "Data Feedback and Clock Enable", *Development system design guide; Chapter 2: Design Flow*, 2003.

[Xil03b] Xilinx inc, "Data Feedback and Clock Enable", *Development system design guide; Chapter 2: Design Flow*, 2003.

[Xil04a] Xilinx Inc, "Xilinx Virtex-II Web Power Tool Version 2.2.0", available at http://www.xilinx.com/cgi-bin/power_tool/web_power_tool.pl

[Xil04b] Xilinx Inc, "Virtex-II Platform FPGAs: Functional Description", *Data sheet DS031 (v3.3)* June 24, 2004. Available at www.xilinx.com

[Xil04c] Xilinx Inc, "Xilinx Online Documentation: Library Guide V6.1 - Global Clock MUX Buffer", 2004. Available at http://www.xilinx.com

[Xil95] Xilinx Inc, "Power Considerations", in *Technical Conference and Seminar Series*, 1995.

[Xpo02] Xpower, "Xpower getting started", Release version 4.2.03i, 2002, available at support.xilinx.com.

[Xpo03] Xpower, "Xpower getting started", Release version 5.1.03i, 2003, available at support.xilinx.com.

[Xpo04] Xpower, "Xpower getting started" Release version 6.2.03i, 2004, available at support.xilinx.com.

Capítulo 4:

Reducción de Consumo en Máquinas de Estado

En este capítulo se examina la reducción de consumo en máquinas de estados finitos (*Finite State Machine - FSM*) implementadas en FPGAs. El capítulo comienza con una revisión de las estrategias utilizadas para la reducción de consumo en máquinas de estados (sección 4.1), más adelante, en la sección 4.2 se muestran resultados experimentales en la codificación de máquinas de estados. En la sección 4.3 se describe una técnica desarrollada en esta tesis para la partición de máquinas de estados y para finalizar, en la sección 4.4 se enumeran recomendaciones para la reducción de consumo en máquinas de estado.

4.1 Estrategias para reducción de consumo en máquinas de estados

Las principales técnicas para la reducción de consumo en máquinas de estados se basan en tres técnicas: la minimización o reducción de estados, la forma en que se codifican los estados y a la descomposición en varias submáquinas de estados. A continuación se describen estas alternativas.

4.1.1 Minimización de estados

El problema de reducción o minimización de estados es común a la minimización de área en los circuitos y ha sido amplia y matemáticamente estudiado. La reducción de estados trae aparejado la reducción en área, la cantidad de interconexiones, así como los registros necesarios. Existen múltiples herramientas informáticas que realizan la minimización de estados, una de las más populares es STAMINA [Hac91] que se distribuye con el paquete SIS [Sen92] basando los algoritmos de minimización propuestos en [Pau59] [Hac91].

4.1.2 Asignación de estados (*state assignment / encoding*)

Las técnicas de asignación de estados es quizás la técnica más extendida para la reducción de consumo en máquinas de estados finitos. El objetivo principal es reducir la cantidad de transiciones entre estados.

4.1.2.1 Enfoques tradicionales para la codificación de estados

Los métodos tradicionales de codificación de estados estuvieron orientados a reducir fundamentalmente área o eventualmente retardos. Por ejemplo la herramienta NOVA implementa la codificación óptima en dos niveles [Vil90], mientras que el sistema de asignación de estados MUSTANG implementa la asignación para múltiples niveles [Dev88]. La herramienta JEDI [Lin89] es un programa simbólico general de codificación (es decir, codifica entradas, salidas y estados) orientado a implementación con redes multinivel. Esta herramienta se distribuye con el paquete de síntesis de circuitos secuenciales SIS [Sen92] desarrollo por la Universidad de California, Berkeley.

Dado que estos métodos tradicionales estaban orientados a implementaciones tipo ASIC y minimización de área, estas codificaciones generaban una cantidad mínima de *flip-flops* pero funciones de codificación grandes.

4.1.2.2 Aproximaciones para asignación de estado de bajo consumo

Los trabajos de asignación de estados para bajo consumo parten del cálculo probabilístico de transiciones de estados y actividad de conmutación [Tsu94a]. La idea común de los diferentes enfoques es la reducción del promedio de conmutaciones, a

través de la minimización de la distancia de Hamming en las transiciones más probables. Benini y De Micheli [Ben95a] utilizan una descripción probabilística de la FSM que modela el grafo de transición de estados (STG) como una cadena de Markov y resuelven el problema de asignación con mínima actividad usando $\lceil \log_2 n \rceil$ bits, donde n es el numero de estados. Noth y Kolla utilizan un método basado en árboles de recubrimiento (*Spaning Trees*) cuya característica más importante es que no solo se limitan a codificaciones de $\lceil \log_2 n \rceil$ bit, sino que la solución puede utilizar entre $\lceil \log_2 n \rceil$ y n bits. Otras contribuciones interesantes en el tema son [wu00][Tsu94b][Mar00].

4.1.2.3 Asignación de estados en FPGAs

Los trabajos de investigación antes mencionados fueron concebidos para circuitos integrados tipo *gate arrays* o *estandar cells*. Los fabricantes de FPGAs y de herramientas de síntesis utilizan *One Hot* como la codificación por defecto para máquinas de estado [Xil00a][Exp99]. La codificación *One Hot* utiliza tantos registros como estados a codificar, con un único bit a '1' y todos los demás a cero. Este sistema de codificación permite crear FSMs más eficientes en FPGA en términos de área y profundidad lógica (velocidad).

Las FPGAs poseen muchos registros pero la generación de funciones a través de tablas *look up* está limitada a unos cuantos bits de ancho. La codificación *One Hot* incrementa la cantidad de *flip-flop* utilizados (uno por estado) pero decrece la cantidad de lógica combinacional, además la distancia de Hamming es siempre dos independientemente de la cantidad de estados. Otra ventaja que posee ésta codificación es la facilidad para implementar la lógica de siguiente estado, con lo que es atractiva para máquinas de estado grandes. No obstante, para máquinas pequeñas las codificaciones binarias pueden tener mejores resultados.

4.1.3 Descomposición de maquinas de estado (*FSM partitioning o descomposition*)

En la descomposición de máquinas de estados se trata de dividir la máquina en otras más pequeñas comunicadas entre si. Las maquinas más pequeñas han de ser más fáciles de asignar estados por separado que todas juntas. Además, si la división se realiza correctamente, la mayor parte del tiempo solo una de ellas ha de estar funcionado pudiéndose desactivar las demás.

Utilizar esta técnica plantea dos problemas a resolver: en primer termino realizar la división de la maquina original en N submáquinas de modo que las transiciones entre ellas sean mínimas. En segundo termino se debe resolver la forma en que se ha de desactivar a las N-1 maquinas que se encuentran inactivas. En la sección 4.4 se estudia en profundidad la aplicación de esta técnica en FPGAs.

4.2 Experimentos sobre codificación de máquinas de estados en FPGAs

En esta sección, se describen los experimentos realizados sobre la codificación de bajo consumo en máquinas de estados síncronas sobre FPGAs. Se han estudiado cuatro sistemas de codificación: En primer lugar, el tradicional sistema binario de codificación, luego el sistema *One-Hot* propuesto por los fabricantes de FPGAs, más tarde un sistema de codificación que minimiza las funciones de salida y finalmente un sistema denominado *Two-Hot*. Como banco de pruebas (*Benchmarks*) se han utilizados las máquinas de estados del MCNC [Lis88] y del consorcio PREP [Pre00]. Una descripción más detallada se encuentra en el Apéndice D.

4.2.1. Experimentos

Los experimentos se llevaron a cabo sobre cuatros sistemas de codificación, dos sistemas densamente codificados y otros dos escasamente codificados los que se describen a continuación:

- *Binario (Bin):* Es el tradicional método de codificación en que a cada código binario se le asocia un estado. Se utilizan $\lceil \log_2 n \rceil$ registros para almacenar los n estados.
- *One-Hot (OH):* Es la codificación por defecto en FPGAs donde se utiliza n registros para codificar n estados y donde solo un registro esta a uno y los demás todos a cero. Posee la ventaja de simplificar la lógica de codificación de próximo estado a expensas de aumentar el uso de registros.
- *Out-oriented (Out-O):* Es un estilo de codificación provisto en la herramienta de codificación de estados JEDI [Lin89] que realiza una codificación binaria

(utiliza $\lceil \log_2 n \rceil$ registros) que minimiza la cantidad de lógica para las funciones de salida.

- *Two-Hot (TH):* Es un intento por reducir la cantidad de registros sin perder la facilidad de codificación de la funcion de proximo estado. Aquí se utilizan dos bits a uno y el resto a cero lo que requiere $\lceil \log_2 n \rceil$ registros para codificar n estados.

Claramente Binario y *Out-oriented* son las alternativas llamadas densamente codificadas (por el uso que realizan de las posibles codificaciones sobre los registros utilizados) en tanto que *One-Hot* y *Two-Hot* son las consideradas codificaciones dispersas o poco densas.

```
# Ex5.KISS example
.i 2
.o 2
.p 16
.s 4
00 S0 S0 11
-1 S0 S2 00
10 S0 S0 11
00 S1 S0 00
01 S1 S2 11
. . .
.e
```

Figura 4.1. Ejemplo código KISS.

Los experimentos de esta sección se base en 26 máquinas de estados del banco de pruebas MCNC [Lis88] junto a las dos máquinas de estado provistas por el consorcio PREP [Pre00]. El conjunto de máquinas de estados seleccionado presentan entre 4 y 48 estados, de 1 a 12 entradas y de 1 a 19 salidas. Las maquinas están descritas en el formato KISS2 [Sen92]. Un ejemplo se muestra en la Figura 4.2.

Para proceder a la síntesis e implementación de los circuitos se realizó un programa llamado KISS2VHDL que traduce la especificación en KISS y genera el código VHDL correspondiente. El programa infiere la máquina de Mealy o Moore según corresponda y realiza la asignación de estados según se haya indicado a través de parámetros externos. El programa genera una entidad VHDL con la máquina de estados (Figura 4.2) y un top-level VHDL con *buffers* de tercer estado en las patas para

Todos los circuitos fueron implementados y medidos en idénticas condiciones, esto es, todos los circuitos se implementaron en la misma FPGA, con idéntica asignación de patas, mismas opciones en las herramientas de síntesis, tarjeta de circuito impreso (Apéndice A), frecuencia de reloj y sondas del analizador lógico. Se utilizó la misma secuencia aleatoria para estimular los circuitos, y la salida de cada pata solo soporta la carga del analizador lógico digital, inferior a 3 pf [Tek02].

Circuitos	FSM caracterist.				Área Bin		Área OH		Área out-o		Área T-H		Retardo (ns)				Consumo (mW/MHz)			
	inputs	outps	rules	states	CLBs	FF	CLBs	FF	CLBs	FF	CLBs	FF	Bin	OH	Our-O	T-H	Bin	OH	Our-O	T-H
bbara	4	2	42	7	11	3	8	7	10	3	15	4	30.0	25.6	29.4	31.2	1.39	1.38	1.87	1.46
bbsse	7	7	208	13	36	4	26	13	27	4	36	5	43.1	36.2	34.6	40.1	4.02	3.37	3.14	3.43
bbtas	2	2	24	6	4	3	4	6	3	3	4	3	16.8	12.7	16.7	15.5	1.08	0.95	0.77	0.97
beecount	3	4	20	4	7	2	10	4	7	2	12	4	21.1	18.6	16.4	28.9	1.33	1.62	1.33	2.36
cse	7	7	91	16	52	4	42	16	48	4	53	5	54.9	39.1	47.4	47.9	3.73	3.50	2.99	3.83
dk14	3	5	56	7	27	3	26	7	25	3	27	4	34.1	32.5	31.7	37.8	4.15	3.88	4.08	3.92
dk15	3	5	32	4	18	2	20	4	20	2	20	4	29.2	28.2	25.6	32.8	3.32	3.02	3.28	3.85
dk16	2	3	108	27	59	5	31	27	50	5	57	7	52.1	35.0	43.3	44.0	8.09	3.73	6.67	6.64
dk17	2	3	32	8	12	3	10	8	13	3	14	4	24.2	27.8	27.3	24.5	2.30	1.94	2.27	2.28
dk27	1	2	14	7	3	3	4	7	3	3	4	4	12.6	20.2	18.6	18.8	0.88	1.08	0.95	1.36
dk512	1	3	30	15	14	4	10	14	9	4	16	5	20.8	20.4	26.0	23.9	2.46	1.54	1.85	2.48
ex2	2	2	56	14	21	4	17	11	12	4	22	5	31.0	21.3	24.4	27.4	3.60	2.03	1.88	3.23
ex3	2	2	20	5	6	3	8	5	7	3	7	3	19.2	18.1	16.7	13.7	1.38	1.52	1.51	1.44
ex4	6	9	21	14	22	4	15	14	19	4	18	5	31.2	29.4	27.0	27.2	2.51	1.66	2.10	2.11
ex5	2	2	16	4	1	2	5	4	4	2	7	4	8.8	20.1	17.7	25.8	0.55	1.26	0.98	1.39
ex6	5	8	34	8	34	3	28	8	29	3	35	4	40.0	31.4	33.6	47.6	4.25	3.59	3.71	4.86
ex7	2	2	16	4	2	2	5	4	2	2	7	4	10.2	14.5	9.5	18.3	0.62	1.16	0.64	1.49
keyb	7	2	170	19	57	5	42	19	50	5	53	6	58.1	41.7	54.9	62.3	6.55	5.05	4.43	6.02
kirkman	12	6	370	16	45	4	43	16	45	4	57	5	38.3	36.2	38.9	36.6	4.14	4.00	3.73	5.21
lion9	2	1	16	4	2	2	2	4	2	2	5	4	8.8	15.1	8.8	25.5	0.44	0.54	0.43	1.04
mark1	5	16	180	12	19	4	15	12	17	4	17	5	30.2	24.6	24.1	30.5	2.50	1.79	2.11	2.41
opus	5	6	29	9	23	4	15	9	20	4	18	4	31.1	33.0	27.8	28.1	2.95	1.74	2.16	2.45
planet	7	19	115	48	113	6	65	48	106	6	99	10	60.6	41.3	54.3	61.1	14.4	6.23	13.2	11.7
prep3	8	8	29	8	13	3	14	8	12	3	18	4	33.3	26.9	26.5	30.9	1.66	2.04	1.42	1.99
prep4	8	8	78	16	39	4	37	16	35	4	41	5	45.9	31.4	41.5	37.7	5.47	5.29	4.37	4.92

Tabla 4.2. Área-Velocidad y Consumo para el conjunto de circuitos de prueba.

Los circuitos fueron medidos a 100 Hz, 2MHz, y 4 MHz para extrapolar la potencia estática. Todos los prototipos incluyen *buffers* triestados en las patas de salida para

facilitar la medida de la potencia de salida. Se pueden ver los detalles de la metodología de medición en el Apéndice A.

4.2.2. Resultados Experimentales

En la Tabla 4.2 se puede observar área, retardo y consumo dinámico obtenidos para cada uno de los circuitos de prueba. El área es expresada en CLBs, aunque también se informa la cantidad de registros *flip-flop* (FF) que se utilizan. El retardo, expresado en *ns*, corresponde al camino crítico. Finalmente la potencia dinámica es expresada en mW/MHz.

Ahorro de consumo: La Figura 4.3 muestra la comparación de ahorro de consumo de OH (*One Hot*) vs. codificación binaria, en tanto la Figura 4.4 hace lo propio para OH vs. "*Out-oriented*". Los valores positivos indican reducciones obtenidas por la codificación OH. El eje de las abscisas representa el número de estados de la FSMs. Las figuras pueden ser, conceptualmente, divididas en tres claras zonas. Para máquinas hasta ocho estados, la codificación binaria debe ser utilizada para reducir consumo. Para máquinas con más de 16 estados siempre OH es la mejor opción y finalmente si la máquina posee entre 8 y 16 estados no es claro la opción a elegir, pero claramente "*Out-oriented*" es mejor que binario puro.

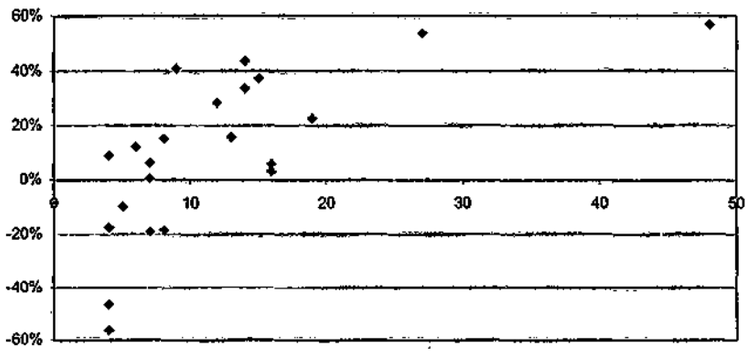


Figura 4.3. Ahorro de consumo en función de la cantidad de estados. One-Hot respecto de Binario.

Por otra parte la codificación TH (*Two-Hot*) consume más que OH en prácticamente todos los casos, no obstante, sigue siendo mejor opción que "*Out-oriented*" y binario puro para máquinas de estado grandes. Cabe destacar que el ahorro de consumo por la elección de la codificación correcta puede llegar al 57%.

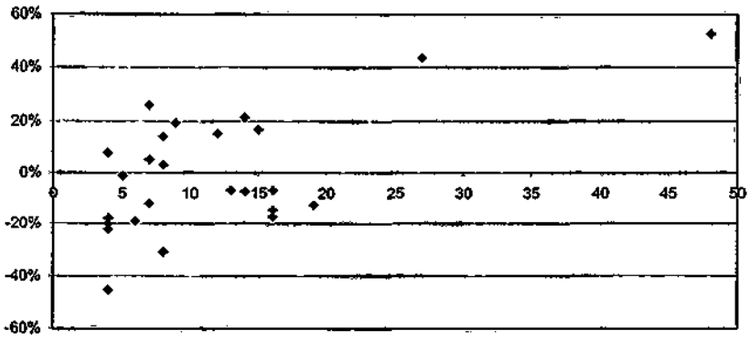


Figura 4.4. Ahorro de consumo en función de la cantidad de estados. One-Hot respecto de "Out-oriented"

Relación estados-consumo: Se comprueba que para todas las codificaciones, el consumo es lineal con la cantidad de estados. El coeficiente de determinación *r2* para los diferentes análisis de regresión esta siempre sobre 0,85 (Figura 4.8). El consumo está aún más relacionado ($R^2 \cong 0{,}87$) respecto de *n+i* (cantidad de estados más el número de entradas).

Relación estados- área: En este caso, la correlación es similar al análisis previo con un coeficiente de determinación $r2 \cong 0{,}80$. Como es de esperar, este relación aumenta cuando se relaciona el área con de *n+i* (estados más cantidad de entradas) obteniendo un $r2 \cong 0{,}86$. La relación área respecto de *n+i+o* (estados más cantidad de entradas y salidas), *r2* supera el 0,88.

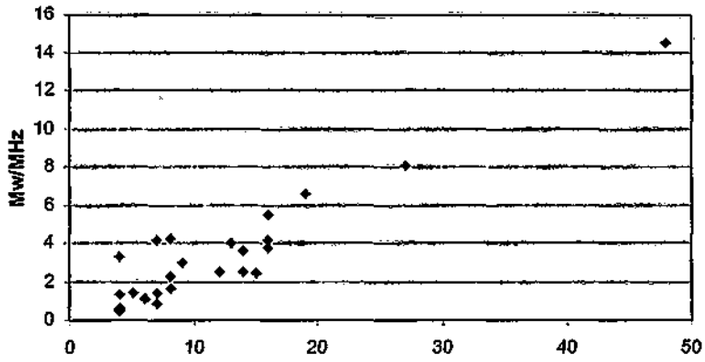


Figura 4.5. Consumo por cantidad de estados. Codificación binaria.

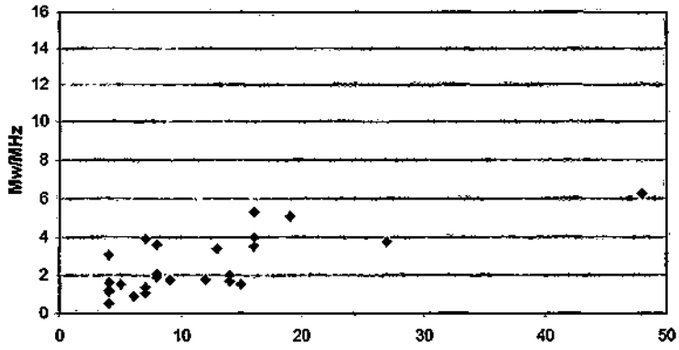


Figura 4.6. Consumo por cantidad de estados. Codificación One-Hot.

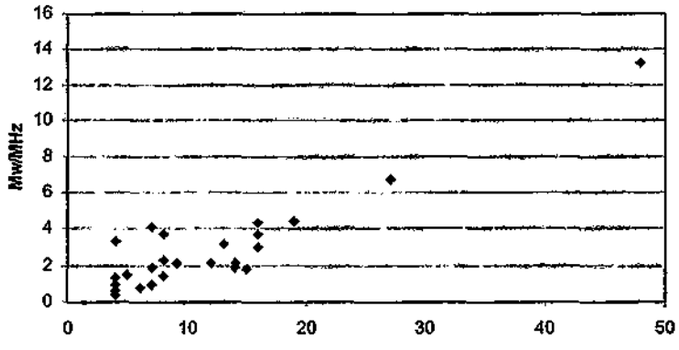


Figura 4.7. Consumo por cantidad de estados. Codificación Out-oriented.

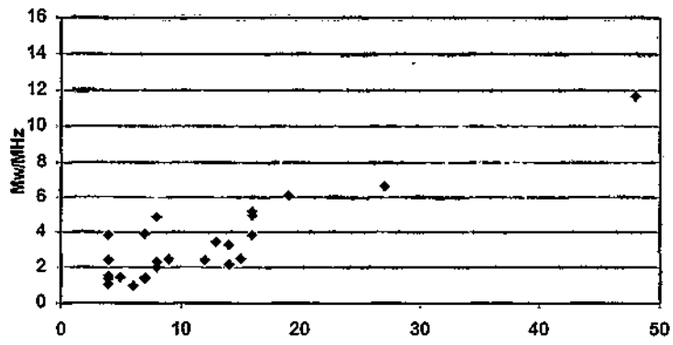


Figura 4.8. Consumo por cantidad de estados. Codificación Two-Hot.

Velocidad-Consumo: La relación se puede ver gráficamente en la Figura 4.9, el coeficiente de correlación es $r2 \cong 0.7$. El experimento no sigue tan claramente la regla general dentro del diseño de bajo consumo en FPGAs, que indica que, los circuitos más rápidos consumen menos potencia. La afirmación que para una máquina de estados determinada la codificación que produce el circuito más veloz es la que menos consume solo se cumple 36 %.

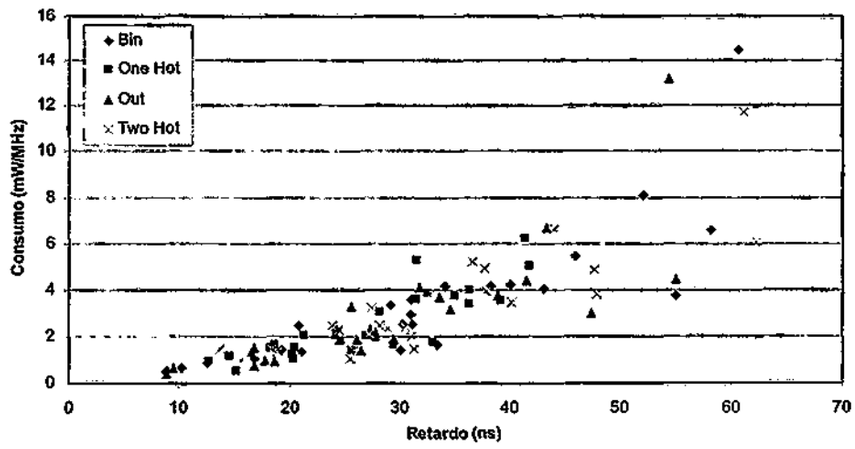


Figura 4.9. Relación retardo- consumo para las FSMs

Área-Consumo: La correlación es realmente importante ($r2 \cong 0.91$) y puede ser utilizada como una primera aproximación para decidir por un sistema de codificación. La figura Figura 4.10 muestra esta distribución. Si se compara área y consumo para una misma FSM, se puede observa que en este experimento, el 77 % de las veces se cumple que el circuito que menos ocupa es el que menos consume.

Otras correlaciones: Se han explorado otras correlaciones, como cantidad de estados-velocidad sin resultados visibles ($r2$ menor que 0,6). Otras correlaciones de área, velocidad o consumo respecto de los diferentes paramentos de la máquina de estados (cantidad de entradas, salidas, estados, reglas) y combinaciones de estos, tampoco producen resultados significativos.

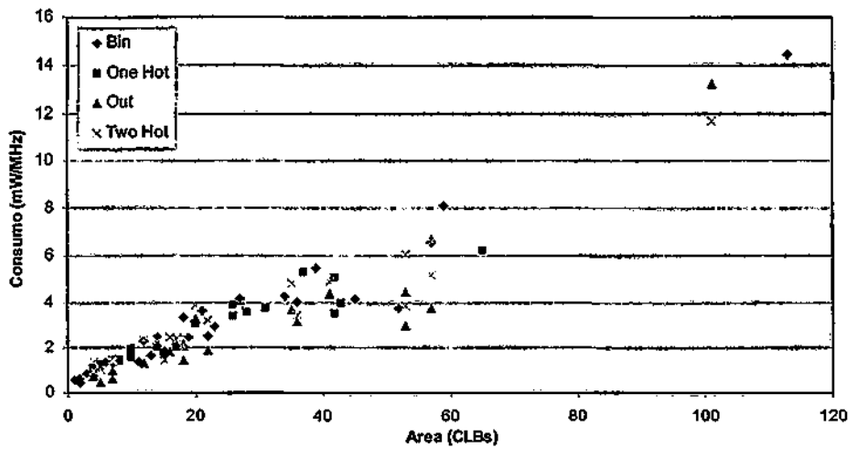


Figura 4.10. Relación área – consumo en las máquinas de estados.

4.2.3 Conclusiones sobre codificación de máquinas de estados en FPGAs

Esta sección ha presentado un análisis de las alternativas de codificación para maquinas de estados desde el punto de vista del bajo consumo. Las conclusiones más importantes son que, para máquinas de estados pequeñas (hasta ocho estados), el área, retardo y consumo es minimizado con codificaciones binarias (codificaciones densas).

Por otra parte las codificaciones poco densas, pero más fáciles de decodificar como *One-Hot* o *Two-Hot* muestran mejores resultados para máquinas de estados grandes (más de dieciséis estados).

La comparación entre los 26 circuitos de prueba muestra una gran diferencia de consumo. Dependiendo del sistema de codificación utilizado, se puede alcanzar una ahorro de hasta el 57%. La aproximación *Two-Hot* si bien es mejor que binario y "*out-oriented*" para máquinas grandes siempre es de inferior calidad que *One-Hot* por lo que no parece ser muy interesante. Dentro de las codificaciones densas "*out-oriented*" se comporta en promedio mejor que el binario puro.

Por último se puede observar una clara relación área-consumo. Esta puede ser utilizada durante el ciclo de diseño para estimar el consumo a través de la información provista por la herramienta de síntesis.

Estos resultados han sido publicados en [Sut02a] y [Sut02b], posteriormente [Men03] propuso un enfoque híbrido utilizando *cero-one-hot* para la parte de la FSM con mayor actividad y una codificación mínima (ó densa) para las partes con menor actividad. Sus resultados por simulación utilizando Xpower y dispositivos Spartan-2 llegan al 60 % de reducción.

4.3 Partición de máquinas de estado en FPGAs

En esta sección, se estudia la optimización de consumo en máquinas de estado finitos implementadas en FPGA por la técnica de descomposición (*decomposition*) o particionado (*partitioning*). Cuando el tamaño de las máquinas de estado crece su complejidad hace que las estrategias conocidas de asignación de estados no sean óptimas. Se pueden lograr mejoras en área, velocidad y consumo en circuitos secuenciales realizando la interconexión de dos o más circuitos. Las técnicas heurísticas para la asignación de estados y optimización lógica trabajan mejor sobre problemas pequeños que sobre grandes.

Desde el punto de vista del consumo, si se realiza una división correcta, la mayor parte del tiempo solo una de las submáquinas de estados ha de estar funcionado pudiéndose desactivar las demás.

En la solución que se presenta en esta sección, la máquina de estados es dividida en dos submáquinas usando un criterio probabilístico. Solo una submáquina está activa a la vez, mientras la otra esta inactiva con el objeto de ahorrar consumo. Se han probado diferentes alternativas para llevar a cabo la desactivación en dispositivos Xilinx XC4K.

Para probar éstas técnicas se han utilizado 14 máquinas de estados de los bancos de prueba (*benchmarks*) MCNC [Lis88] y PREP [Pre00] y construido más de 200 circuitos. Con la técnica propuesta en esta sección, que se adapta a grandes máquina de estados, se han logrado ahorros de consumo de hasta el 42,5 % respecto de la máquina tradicional codificada en *one-hot* y de hasta 54 % respecto de la misma máquina con codificación binaria.

4.3.1 Alternativas de bajo consumo en FSMs

Indudablemente la técnica más popular para reducir consumo es modificando la manera en que se codifican los estados [Tsu94b][Ben95a][Not99][Wu00][Mar00] como se ha explicado y estudiado en la sección 4.2. No obstante, otras ideas muy utilizadas son lo que se conoce como técnicas de manejo de energía (*power management*). Esto es "apagar" los bloques de hardware en aquellos periodos de tiempo en que no están produciendo datos útiles. El apagado de un circuito puede ser llevado a cabo de

diferentes maneras: Apagando la fuente de alimentación; deshabilitando la señal de reloj; o "congelando" (*freezing or blocking*) los datos de entrada.

Dentro de esta categoría de técnicas, caen métodos como precomputación (*precomputation*), bloqueo de reloj (*gated clock*), sistemas con reloj selectivo (*selectively clocked systems*). En las técnicas de "*gated-clock*" [Ben96][Ben95b], el reloj de la FSM se para cuando la máquina esta en un auto-ciclo (queda en el mismo estado) y las salidas no cambian. En precomputación [Ali94], un bloque de lógica combinacional es agregado al circuito original. Bajo ciertas condiciones de entrada, la lógica de precomputación deshabilita la carga de los registros de entrada. Esta sección se centra en las alternativas de descomposición que se detallan a continuación.

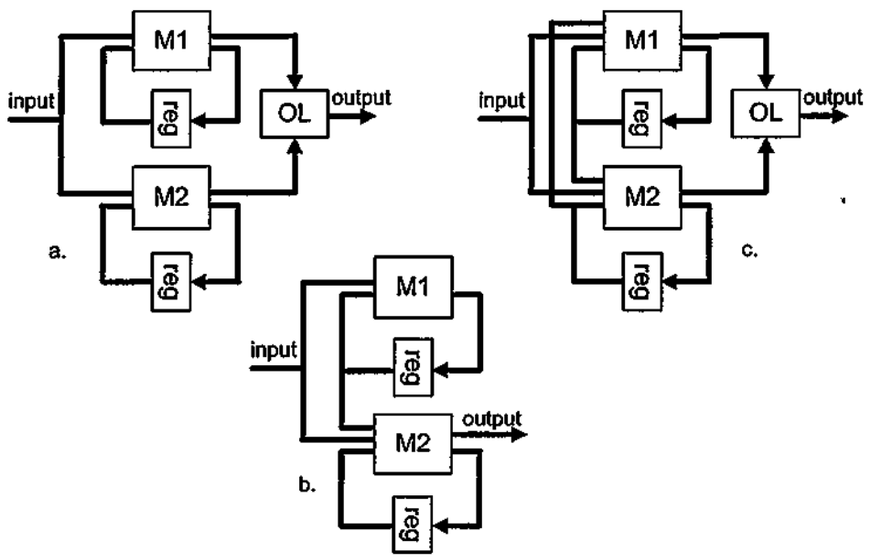


Figura 4.11. Descomposición de máquinas de estados. a. Paralela. b. Cascada. c. General.

4.3.2 Decomposición o partcionado de FSMs

Los primeros trabajos sobre partición de máquinas de estados datan de los 60 [Har60]. El primer objetivo fue reducir la complejidad del bloque combinacional los que se implementaban en arquitecturas que poseían una cantidad limitada de lógica (PLAs, etc) [Ash91][Gei91]. En lo subsiguiente, por simplicidad, la partición se realizará en

dos submáquinas aunque el concepto es aplicable a *n* submáquinas. Conceptualmente se pueden dividir en máquinas paralelas, en cascada o generales (Figura 4.11), siendo estas últimas las más utilizadas en la práctica.

4.3.2.1 Modelo general para la partición de máquinas de estado

En el modelo general, la FSM se divide en dos (o más) máquinas de estados relacionadas, donde cada submáquina conoce en que estado se encuentra la (las) otras (Figura 4.12). Esta estrategia agrega un estado ocioso en cada submáquina.

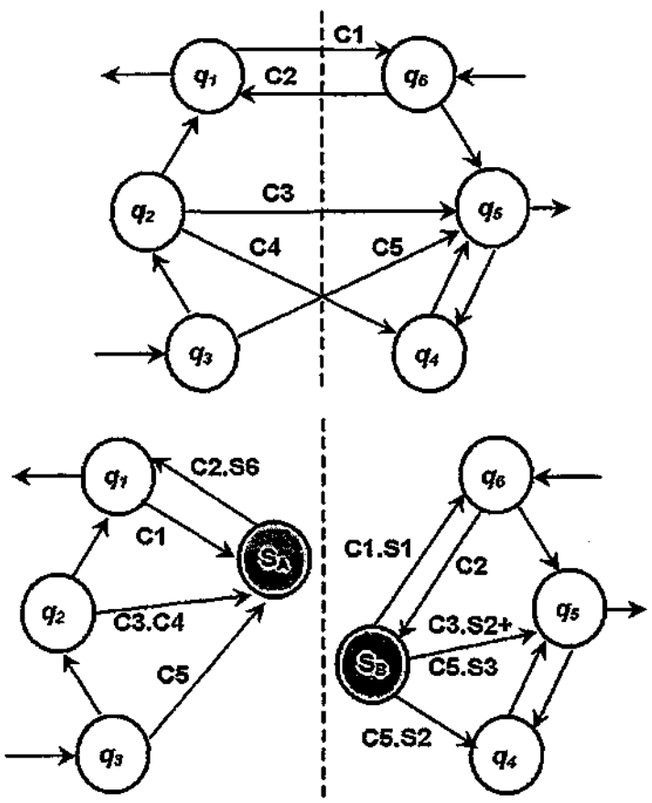
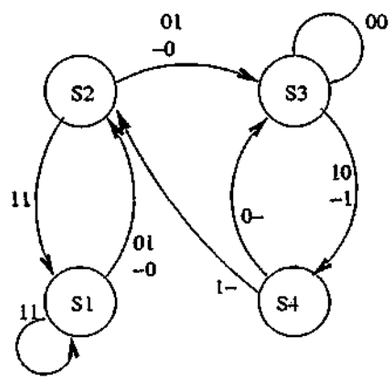


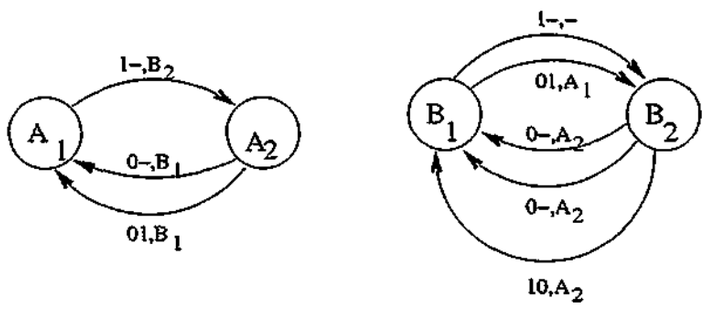
Figura 4.12. Diagrama de estados de una máquina de estados y diagramas de las particiones en la aproximación tradicional.

4.3.2.2 Partición ortogonal de máquinas de estado

Otro esquema de partición de máquinas de estados es la partición ortogonal [she99]. En este caso, la cantidad de estados no es $n_1+n_2 = n$, pero es aproximadamente $\sqrt{n}$ en cada partición. Si se considera Q = $\{q_1,q_2,\ldots,q_n\}$ el conjunto original de estados, luego dos particiones $\prod_A = \{A_1, A_2, \ldots, A_m\}$ y $\prod_B = \{B_1, B_2, \ldots, B_k\}$ del conjunto Q son *ortogonales* si, para todo i, j con $i \leq m, j \leq k$, se cumple que $A_i \cap B_j = \varnothing$ o bien $A_i \cap B_j = \{q_l\}$. De este modo, para representar un estado q_l de la máquina original se utiliza la combinación de un estado A_i y otro B_j



Grafo de transición de estados



STG para la partición A

STG para la partición B

Figura 4.13. Diagrama de estados y partición ortogonal de una FSM [She92]

La función de próximo estado para cada máquina se describe en función de la entrada y el estado de ambas submáquinas δ_a: $\prod_A \times \prod_B \times \Sigma \rightarrow \prod_A$, análogamente δ_b: $\prod_A \times \prod_B \times \Sigma \rightarrow \prod_A$. Lo mismo sucede con la función de salida en que de ser una máquina de Moore dependerá del estado de ambas submáquinas (λ: $\prod_A \times \prod_B \rightarrow A$) y en caso de ser una máquina de Mealy además dependerá de la entrada (λ: $\prod_A \times \prod_B \times \Sigma \rightarrow A$).

En la Figura 4.13 se muestra el siguiente ejemplo. La máquina original posee los siguientes cuatro estados Q = { S_1, S_2, S_3, S_4}. Las siguientes podrían ser dos particiones $\prod_A$ = {(s1,s2),(s3,s4)} y $\prod_B$ = {(s1,s3),(s2,s4)}. Es decir, A_1 = {s1,s2}, A_2 = {s3,s4}, B_1 = {s1,s3}, B_2 = {s2,s4}. Expresado en otros términos, las dos submáquinas están en el estado original s1 cuando la submáquinas estén en A_1 y B_1 respectivamente. Con este último razonamiento se obtiene: s1 = (A_1 , B_1), s2 = (A_1 , B_2), s3 = (A_2 , B_1), s4 = (A_2 , B_2). Obsérvese que las transiciones de estados de las particiones, no solo, dependen de la entrada, si no también, del estado de la otra submáquina.

4.3.3 Técnicas de descomposición para bajo consumo

La idea principal para reducir consumo por descomposición de máquinas de estados es desactivar la parte que no está operando de la FSM. La desactivación puede ser alcanzada o bien bloqueando las entradas, usando *latches*, puertas ANDs, *buffers* tri-estados, ó bien apagando la parte del circuito que no es utilizada (deshabilitando el reloj por ejemplo).

Tanto en [Mon98] como en [Ben98], la máquina de estados es particionada en varias piezas, que son implementadas en máquinas separadas con un estado ocioso extra (*idle state*). En éste caso solo una máquina es activa a la vez, mientras las otras se encuentran en estado ocioso. Por ende se puede desactivar el reloj para las submáquinas ociosas, así como, desactivar sus respectivas entradas. Con esto se logra reducir la actividad de conmutación y por tanto la potencia disipada.

En [Mon98] el diagrama de transición de estados (STG) es dividido en dos maquinas de estados no balanceadas: una es pequeña y se encuentra activa la mayor parte del tiempo, en tanto que la mayor que usualmente esta inactiva. La clave consiste en encontrar un subconjunto pequeño de estados donde la FSM ha de estar mucho

tiempo y que existan pocas transiciones hacia el resto de los estados. El algoritmo se basa en rotular el STG con las probabilidades de transición entre estados.

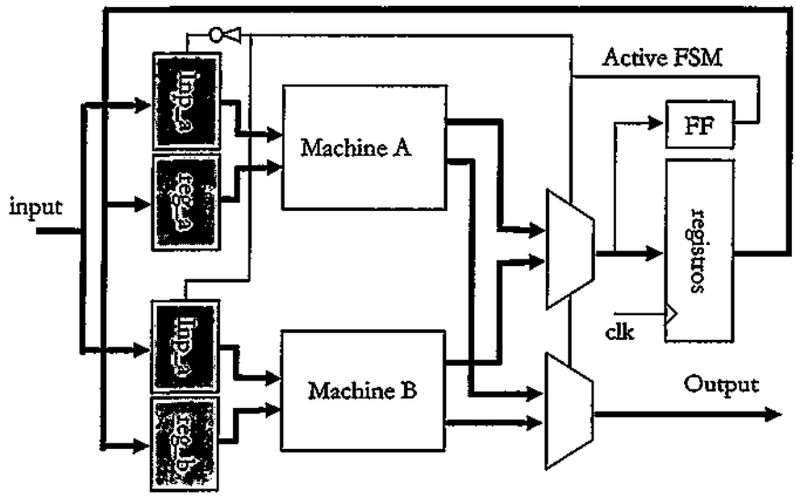


Figura 4.14. Arquitectura I para descomposición de máquinas de estado en FPGAs.

Un esquema muy interesante basado en codificación disjunta es propuesto en [Cho96]. La partición resultante no sigue exactamente la estructura estándar de la descomposición de las máquinas de estados finitos. En este método, el diagrama de transición de estados es particionado en dos (pueden ser más) conjuntos de estados. Todos los estados de un subconjunto de estados es codificado con el bit más significativo (MSB - *most significant bit*) a 0, mientras que el otro conjunto es codificado con su MSB a uno. Por tanto la lógica combinacional puede ser partida en dos bloques separados: uno que es activa cuando el MSB de la codificación esta en cero, y otro que será activo cuando el bit más significativo esté a 1. De esta forma, el consumo puede ser potencialmente reducido.

Finalmente, otra técnica utilizada para reducir consumo es utilizar la partición ortogonal descrita en la sección anterior, utilizando mecanismos de *gated clock* y precomputación para la máquina inactiva [She00]. En cada submáquina de estados, el programa de partición trata de maximizar el número de arcos a si mismo, es decir que

la máquina se quede en el mismo estado, tras el flanco de reloj. Para cada condición de arco a si mismo (*self-edge*), las señales de entrada y de reloj son deshabilitadas.

4.3.4 Una arquitectura de descomposición de FSMs para FPGAs

En esta sección se describe, una arquitectura de descomposición orientada a la implementación en FPGAs basadas en LUTs. Los mismos códigos son utilizados en ambas submáquinas, pero solo una de ellas estará activa a la vez. Para discernir la máquina activa, se utiliza un bit extra llamado *ActiveFSM* (FSM activa). La primera opción arquitectural se puede ver en la Figura 4.14. La máquina de estados es dividida en dos circuitos combinacionales (machines A y B), ambos computan las salidas y próximo estado. La transferencia de control entre las submáquinas se basa en que la que tiene el control, cambia el valor del bit *activeFSM*. En función del valor *activeFSM* se controla cual de las submáquinas es activa activando los multiplexores y bloques de entrada correspondientes. Los bloques sombreados indican circuitos que "congelan" las entradas.

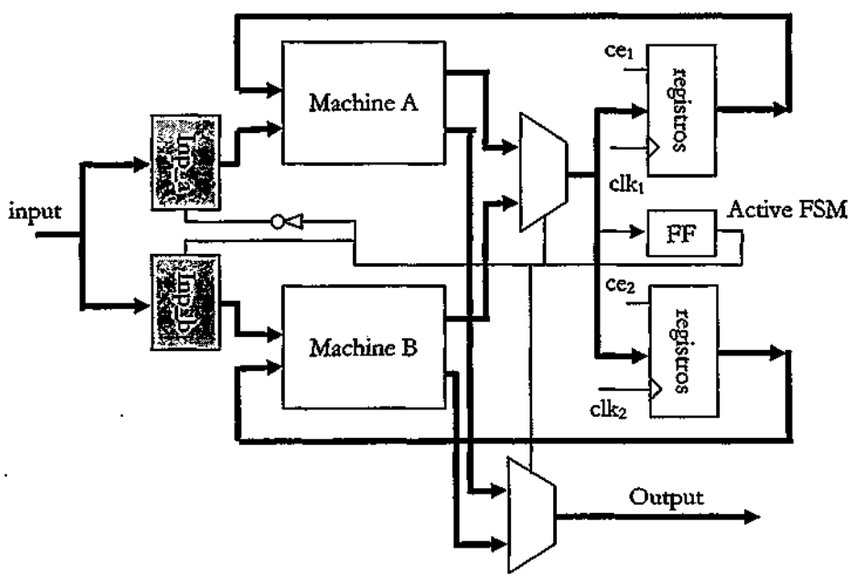


Figura 4.15. Arquitectura II para descomposición de máquinas de estado en FPGAs.

Una segunda arquitectura se puede ver el la Figura 4.15. En este caso, dos bancos de registros son utilizados para parar la evolución de la máquina de estados. Existen dos posibilidades para controlar los registros. Vía la señal de habilitación (*enable*) ó utilizando la técnica de *gated-clock* desaconsejado para el caso de FPGAs por los fabricantes [Xil03].

Para ambas arquitecturas, se lleva a cabo el mismo ciclo de diseño. En primer lugar se debe seleccionar un algoritmo para descomponer la máquina de estados en dos submáquinas. Más tarde, se debe implementar un método para bloquear los datos en la submáquina deshabilitada. Finalmente, se debe escribir el código VHDL para proceder a la síntesis e implementación.

4.3.5. Cálculo de probabilidades de transición en un STG

Para llevar a cabo la partición de las máquinas de estado se realiza un modelo probabilístico [Tsu94a] del grafo de transición de estados de la FSM. Para calcular la probabilidad de transición de estados sobre un grafo de transición de estados (STG), se debe conocer ante todo la distribución de probabilidad para las entradas. Estos valores pueden ser obtenidos por una simulación de alto nivel en el contexto donde interactúa la máquina de estados o bien asignar equiprobabilidad.

Luego la probabilidad de transición para cada arco dentro del STG puede ser determinado, modelando al grafo de transición de estados como una cadena de Markov. Una cadena de Markov es un proceso estocástico cuyo funcionamiento dinámico es tal que su comportamiento depende solo del estado presente, sin tener en cuenta como el proceso ha arribado al estado actual.

La probabilidad estática (*steady state probability*) de un estado q_i esta definida como la probabilidad de una máquina de estados de quedarse en el estado q_i. Este valor no es dependiente del tiempo. Esto es, cuando el tiempo crece, converge a un valor real. Sea **P** la matriz de probabilidades condicionales y **v** el vector de probabilidades estáticas (cuyos componentes son las probabilidades de los estados). Luego, la probabilidad estática para cada estado puede ser calculada resolviendo el sistema de n+1 ecuaciones:

$$v \cdot P = v \quad \text{y} \quad \sum_{i=0}^{n-1} P_i = 1; \quad \text{donde} \quad v = [P_0 P_1 ... P_{n-1}] \quad \text{y}$$

$$P = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ p_{n-1,0} & p_{n-1,1} & \cdots & p_{n-1,n-1} \end{bmatrix}$$

Aquí **P** es una matriz estocástica (esto es, todos los valores son positivos y la suma de cada columna es uno) cuyas entradas son las probabilidades condicionales de transición. La probabilidad total de transición $P_{i,j}$ puede ser calculada como $P_{i,j} = p_{i,j} \cdot P_i$.

Un ejemplo sencillo: La Figura 4.16 muestra una máquina de estados y las diferentes transformaciones que sufre el diagrama de transición de estados para terminar en un grafo ponderado, que puede luego ser utilizado por el algoritmo de partición de máquinas de estado.

Para obtener el grafo de probabilidades de transición se asume que las entradas son no relacionadas y equiprobables. Luego, la probabilidad condicional puede ser calculada como se explico anteriormente. La Figura 4.16.b muestra la probabilidad condicional para cada arco. Obsérvese por ejemplo, la transición del estado S1 al S2 donde los valores 00, 10 y 11 pueden generar transiciones de estados, los que se corresponden con una probabilidad condicional de 3/4. Los arcos a si mismo son eliminados.

La probabilidad estática de cada estado y la probabilidad total de transición (la que surge de multiplicar la probabilidad estática por la probabilidad de transición del estado donde parte) se puede observar en la Figura 4.16.c. Finalmente en la Figura 4.16.d se obtiene el grafo de probabilidades de transición a través de sumar los arcos paralelos (observar que en este ejemplo los valores no están normalizados). En el apéndice D se describen detalles de la implementación de éste cálculo.

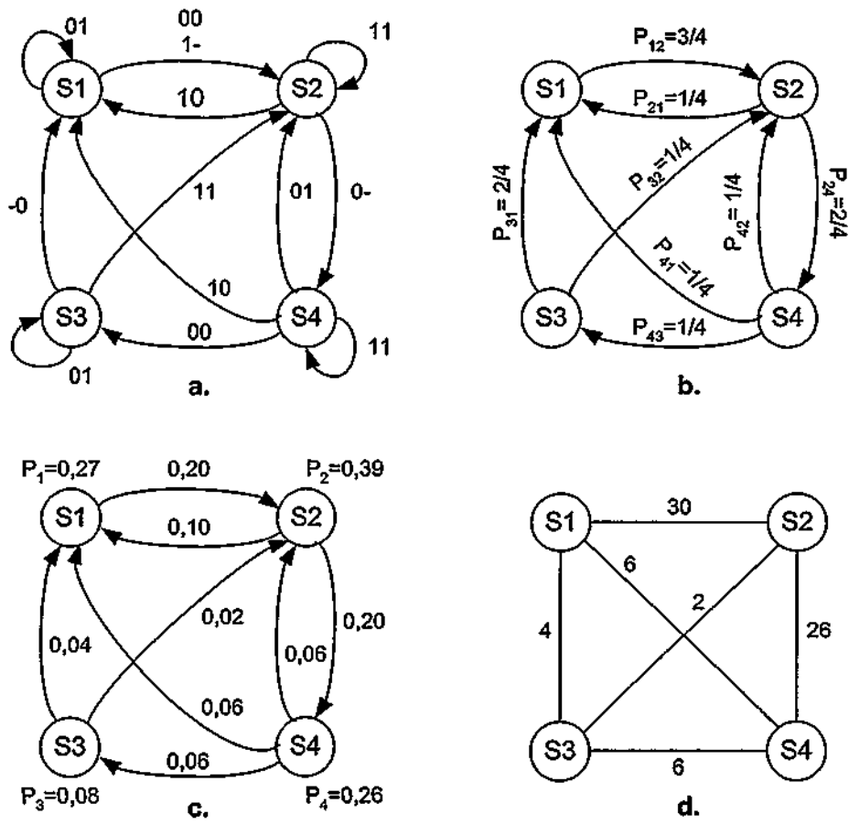


Figura 4.16. Diagrama de transición de estados y grafos de probabilidades. a. FSM de ejemplo. b. Probabilidad condicional para cada arco. c. probabilidad estática de cada estado y la probabilidad total de transición. d. grafo de probabilidades de transición.

4.3.6. Particionando la FSM en submáquinas

La técnica separa la máquina de estados original en dos submáquinas tal que la probabilidad de transición de estados dentro de una submáquina es maximizada, mientras que la interacción entre la otra submáquina es minimizada.

Primero se calcula la probabilidad de transición de la máquina de estado (Figura 4.17). Luego una partición con igual cardinalidad en cada submáquina es llevada a cabo. Por

ejemplo, considérese dos particiones $\Pi_A = \{S_{a1}, S_{a2}, \ldots S_{an}\}$ y $\Pi_B = \{S_{b1}, S_{b2}, \ldots S_{bn}\}$, con probabilidad de transición *p(i,j)* entre los estados S_i y S_j. En este caso, el algoritmo minimiza la suma de las probabilidades de transición entre submáquinas, esto es:

$$\min(\sum p(i,j)), \quad \forall i \in \Pi_A, j \in \Pi_B .$$

No hace falta un algoritmo ávido (*greedy*) para resolver este problema, ya que la búsqueda exhaustiva por un algoritmo de *backtracking* suficientemente acotado logra resolver los peores casos presentados en pocos segundos. En el Apéndice D se brindan mayores detalles de los algoritmos utilizados.

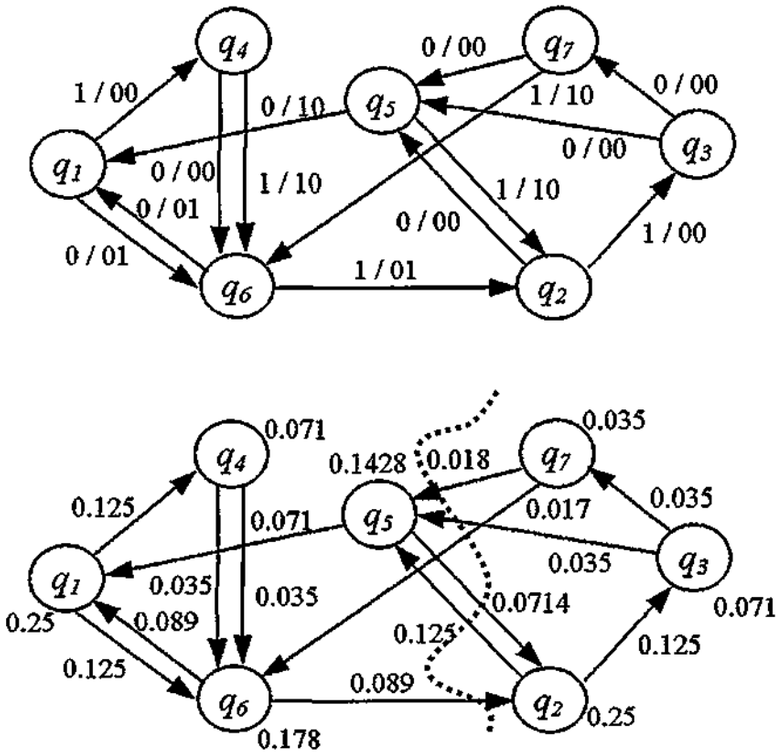


Figura 4.17. a) Diagrama de transición de estados (State Transition Graph - STG), b) Probabilidad estática y probabilidad total de transición.

4.3.7 Métodos para bloquear los datos

Para eliminar la actividad en la máquina de estado inactiva (bloques sombreados en Figura 4.14 y Figura 4.15), la mejor alternativa es utilizar *latches* para capturar los datos (*Blocking Latches*). Otras posibilidades son el uso de puertas ANDs o *buffers* de tercer estado. Ambas fueron probadas y descartadas por su peor comportamiento en área, velocidad y consumo.

Para la arquitectura de partición de FSMs de la Figura 4.15, que utiliza dos bancos de registros, existe la alternativa de usar la inhabilitación por la señal de habilitado (*enable*) o bloquear el reloj (*gate-clock*). La segunda alternativa presenta la ventaja que no existe movimientos en la señal de reloj y consecuentemente no hay consumo. La desventaja pasa por la menor fiabilidad de este método (pueden aparecer movimientos espureos - *glitches*) y el no menos importante hecho que los recursos de reloj no sobran en las FPGAs y puede ser necesario utilizar líneas comunes para rutar el reloj con una importante perdida de calidad en el rutado.

4.3.8 Síntesis de la máquina de estados

Se desarrolló una herramienta denominada *part_FSM* (Apéndice D) que genera la partición de máquinas de estado especificadas en formato KISS2 [Sen92] (Apéndice C). La herramienta calcula la probabilidad estática y divide la máquina original en dos submáquinas como se explica en la sección 4.3.7 para finalmente generar código VHDL sintetizable. Otros parámetros necesarios para el programa son: tipo de arquitectura, método de bloqueo (*blocking method*) y tipo de codificación de las submáquinas.

El código VHDL generado contiene la entidad de la máquina de estados y tres procesos. El primeto para la lógica combinacional, otro para la circuitería de bloqueo de los datos, y por último uno para incorporar *buffers* tri-estado en las patas de salida para medir por separado la potencia a la salida (*off-chip power*).

4.3.9 Experimentos con partición de máquinas de estado

Los circuitos de prueba fueron implementados de diferentes formas: en primer lugar de manera original con codificaciones binarias y *One Hot*. Luego, cada máquina fue particionada de dos formas: una correspondiente a la *arquitectura I* (Figura 4.14) y otra

a la *arquitectura II* (Figura 4.15). Nuevamente para cada submáquina se aplico codificación binaria y *One Hot*. Adicionalmente sobre la *arquitectura I* se probaron diferentes técnicas de bloqueos de datos. Se implementaron y midieron un total de 168 circuitos.

Por otra parte, se procedió a implementar la partición ortogonal (sección 4.3.2) en las máquinas de estado. Esta técnica de descomposición descrita en [She99][She00] reducen el consumo basados en el hecho de la disminución de estados que implica la partición ortogonal. En la implementación en FPGAs realizada, los resultados fueron muy deficientes tanto en velocidad como consumo.

Todos los experimentos utilizan máquinas de estado del banco de pruebas (*benchmark*) MCNC91 [Lis88][Yan91] y del consorcio PREP [Pre00]. Se seleccionaron aquellas máquinas con doce estados o más. Cada máquina de estados fue minimizada en cantidad de estados con STAMINA [Hac91]. La cantidad de entradas, salidas, reglas de próximo estado (arcos del grafo de transición de estados) para cada máquina de prueba se puede ver en la Tabla 4.3. Adicionalmente la probabilidad de transición entre submáquinas y la cantidad de arcos que las comunican tras aplicar el programa *part_FSM* se muestra en la tabla.

Circuitos	Parámetros de las FSM originales				Partición	
	Entradas $\|\Sigma\|$	Salidas $\|\sigma\|$	Estados $\|Q\|$	Reglas $\|\delta\|$	Prob	Arcos
Bbsse	7	7	13	208	0,024	52
Cse	7	7	16	91	0,017	36
Dk16	2	3	27	108	0,247	28
Dk512	1	3	15	30	0,175	7
Ex1	9	19	18	233	0,022	53
Ex2	2	2	14	56	0,218	25
Keyb	7	2	19	170	0,004	63
Kirkman	12	6	16	370	0,002	46
Mark1	5	16	12	180	0,037	79
Planet	7	19	48	115	0,052	14
Prep4	8	8	16	78	0,041	9
S386	7	7	13	69	0,024	27
S820	18	19	24	254	0,006	138
S832c	18	19	24	243	0,006	118

Tabla 4.3. Datos originales de las máquinas de estados, cantidad de entradas, salidas, estados y arcos. Además información de la partición (probabilidad y número de arcos entre particiones)

El código VHDL generado por el programa *part_FSM* (Apéndice E) fue implementado en una FPGA XC4010EPC84-4 usando como herramienta de síntesis FPGA Express [Syn99] y las Xilinx Foundation tools [Xil00b] para su implementación. Este modelo de FPGA no posee *latches*, con lo que éstos fueron construidos con *look up tables* (LUTs) actuando como memoria RAM.

Todos los circuitos fueron implementados y medidos en idénticas condiciones al igual que en los experimentos de la sección 4.2. Se ha utilizando el arreglo experimental del Apéndice A y los circuitos fueron medidos a 100 Hz, 1 MHz, 2 MHz y 4 MHz.

4.3.10 Resultados experimentales de la partición de máquinas de estado

El consumo de corriente expresado en mW/MHz, se muestra en la Tabla 4.4. Las primeras columnas muestran los valores para la máquina de estados original codificada en *One Hot* (OH) y binario (bin). Luego se exhiben los resultados para el circuito particionado y codificado tanto en *One Hot* como en binario. Se presentan los resultados para cuatro formas diferentes *Architectura 1* (*Arch1*), *Architectura 2* (*Arch2*), *Architectura 1* sin métodos de bloqueo (*No Blk*), y finalmente, *Architectura I* con bloqueos implementados con puertas ANDs (*Blk and*). En la última columna se observa el factor de mejora en el consumo, el que se obtiene como relación entre el consumo de la mejor máquina de estados original respecto de la mejor implementación con técnica de partición.

Posteriormente en la Tabla 4.5 se muestra el área de los circuitos de la Tabla 4.4 expresada en CLBs de la serie XC4000, adicionalmente se presenta la utilización de registros *flip-flop*.

Mejora de consumo: Para la mayor parte de las máquinas de estado se obtiene una reducción considerable de consumo llegando hasta un 42,5 % respecto de la máquina tradicional codificada en *one-hot* y de hasta 54 % respecto de la misma máquina con codificación binaria. No obstante, en cinco circuitos no se logro mejora o incluso hubo resultados negativos. Una observación más detallada muestra que, este efecto es causado por la alta probabilidad de transición entre submáquinas (Tabla 4.6). Cuando la partición produce que la probabilidad de transición entre submaquínas supere el 5% los resultados son de esta técnica son negativos.

	FSM Original		Particionado codificación One Hot				Particionado codificación binaria				Mejora del Consumo
Circuito	OH	Bin	Arch1	Arch2	No Blk	Blk and	Arch1	Arch2	No Blk	Blk and	
Bbsse	3,90	4,70	3,80	3,95	4,04	4,34	3,55	3,76	4,23	3,91	9,0%
Cse	3,85	4,10	3,24	3,46	4,29	5,30	3,00	2,88	3,83	3,59	25,3%
Dk16	3,88	10,00	5,80	5,76	5,81	6,34	7,50	7,01	9,09	9,96	-32,8%
Dk512	1,84	2,80	2,46	2,79	2,44	2,14	2,24	2,51	2,16	1,94	-5,2%
Ex1	7,09	8,56	6,73	6,53	8,11	8,16	6,53	6,11	7,90	7,79	13,8%
Ex2	2,51	4,10	3,40	3,09	2,69	3,26	3,09	2,88	3,58	3,46	-6,5%
Keyb	5,50	7,06	4,73	4,31	7,88	7,69	3,66	4,65	5,25	6,81	33,4%
Kirkman	4,50	4,61	4,90	4,66	4,49	4,50	4,80	4,49	4,83	4,80	0,3%
Mark1	2,70	3,30	3,01	3,01	3,31	3,09	2,66	2,78	2,63	2,88	2,8%
Planet	8,04	16,80	9,18	9,29	10,23	10,01	10,88	11,81	15,18	16,99	-12,4%
Prep4	4,66	5,71	5,44	5,38	6,86	7,55	5,11	4,66	6,86	6,44	0,0%
S386	4,23	4,84	4,08	4,45	4,98	4,98	4,21	4,21	5,55	4,59	3,6%
S820	7,84	9,28	5,81	5,44	8,43	7,98	4,51	4,65	8,83	7,30	42,4%
S832c	7,01	10,21	5,08	5,00	7,64	6,60	4,73	5,04	7,55	6,75	32,6%

Tabla 4.4. Consumo de potencia expresado en mW/MHz para la partición de máquinas de estado.

Codificación Binario vs. One Hot en las submáquinas: De acuerdo con los resultados relacionados con el consumo en máquinas de estados sin particionar (sección 4.2), *One Hot* provee mejores resultados para máquinas de estados con más de 16 estados. Por oposición para máquinas con ocho o menos estados la codificación binaria produce mejores resultados.

Métodos de Bloqueo: Los latches son la mejor forma en la mayoría de los casos. La mejora respecto de utilizar puertas AND (que ocupa un CLB) puede llegar a un 30 %. Solo en dos circuitos del banco de pruebas, el bloqueo con puertas AND fue mejor. Se probaron además como alternativas de bloqueo puertas OR y *buffers* triestado con resultados claramente inferiores y fueron desechados. Los resultados concuerdan con los expuestos en la sección 3.6 donde se manifiesta la superioridad de las alternativas de bloqueo por puertas AND y latches.

Circuito	FSM Original				Particionado codificación One Hot								Particionado codificación binaria							
	OH		Bin		Arch1		Arch2		No Blk		Blk and		Arch1		Arch2		No Blk		Blk and	
	CLBs	FF	CLBs	FF	CLBs	FF	CLBs	FF	CLBs	FF	CLBs	FF	CLBs	FF	CLBs	FF	CLBs	FF	CLBs	FF
Bbsse	26	13	36	4	48	8	46	15	37	8	40	8	45	4	38	7	31	4	32	4
Cse	65	128	150	7	145	65	99	129	33	65	33	65	36	7	26	13	5	7	5	7
Dk16	42	16	52	4	64	9	66	17	50	9	56	9	64	4	62	7	46	4	45	4
Dk512	32	27	61	5	68	15	54	29	44	15	49	15	69	5	58	9	53	5	56	5
Ex1	10	14	14	4	26	9	18	17	13	9	10	9	17	4	14	7	10	4	8	4
Ex2	51	18	79	5	90	10	82	19	66	10	82	10	82	5	78	9	71	5	71	5
Keyb	17	11	21	4	39	8	32	15	20	8	26	8	30	4	26	7	23	4	26	4
Kirkman	12	13	21	4	30	8	25	15	17	8	16	8	33	4	31	7	21	4	19	4
Mark1	42	19	57	5	78	11	60	21	57	11	61	11	62	5	64	9	51	5	59	5
Planet	43	16	45	4	80	9	70	17	45	9	46	9	80	4	67	7	39	4	50	4
Prep4	15	12	19	4	34	7	30	13	18	7	19	7	29	4	27	7	15	4	18	4
S386	65	48	113	6	117	25	100	49	83	25	83	25	129	6	127	11	102	6	120	6
S820	34	16	40	4	71	9	60	17	53	9	57	9	50	4	46	7	35	4	43	4
S832c	25	13	36	4	46	8	41	15	36	8	37	8	50	4	45	7	38	4	36	4

Tabla 4.5. Área expresada en CLBs para la partición de máquinas de estado.

Penalidad en Área: Tanto la sincronización como la circuitería de la partición agregan lógica extra a la máquina de estados. Esta sobrecarga depende de la cantidad de entradas, salidas y estados. Cada señal de entrada requiere dos LUTs para implementar los *latches*, y cada salida requiere una LUT extra para implementar el multiplexor de salida. Finalmente, cada estado agrega dos LUTs extra para implementar los latches en *architecture I*, en tanto que la *architecture II* no necesita lógica extra para implementar el bloqueo de los estados. Dado que la XC4000E no posee *latches* y estos se implementan con LUTs configuradas como RAM [Xil99], en términos de área la *arquitectura I* posee peores características que la *arquitectura II* (lo contrario que en el consumo).

Sample	|Σ|	|σ|	|Q|	|δ|	Arcs bet. part	% arcs bet. part.	Prob	Power Improv.
Dk16	2	3	27	108	28	26 %	0,247	-32,8 %
Ex2	2	2	14	56	25	45 %	0,218	-6,5 %
Dk512	1	3	15	30	7	23 %	0,175	-5,2 %
Planet	7	19	48	115	14	12 %	0,052	-12,4 %
Prep4	8	8	16	78	9	12 %	0,041	0,0 %

Tabla 4.6. Circuitos donde no se logra mejora en el consumo debido a la alta probabilidad de transición entre submáquinas de estados

Penalidad del periodo de reloj: El esquema de sincronización produce una degradación importante en la velocidad. La Figura 4.18 muestra la máxima frecuencia en MHz para cada circuito implementado con codificación *One Hot*. La influencia de los *latches* es remarcable, en tanto que la arquitectura que utiliza el bloqueo de los datos con puertas ANDs muestra mejores resultados.

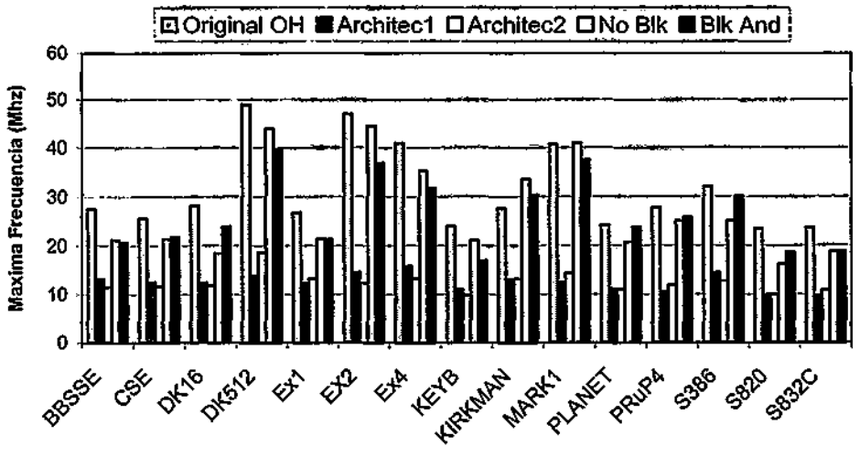


Figura 4.18. Gráfica de frecuencia, donde se observa el impacto negativo de los registros de bloqueo.

Particionado ortogonal: Se implemento el esquema de partición ortogonal como se explica en la sección 4.4.2. Los resultados fueron muy deficientes tanto en área, velocidad así como en consumo. Si bien con está técnica se reduce el número de estados total, el

aumento de lógica necesaria para la decodificación de siguiente estado y salida aumenta el área y actividad del circuito. En la Figura 4.19 se muestra el área en CLBs y en la Figura 4.20 el retardo en *ns* de las particiones de los circuitos con *arquitectura 1* y *arquitectura 2* respecto de la partición *ortogonal*. Por otra parte, en la Figura 4.21, se compara el consumo expresado en mW/MHz para los mismos circuitos. Claramente y sin excepción la partición ortogonal muestra peores resultados.

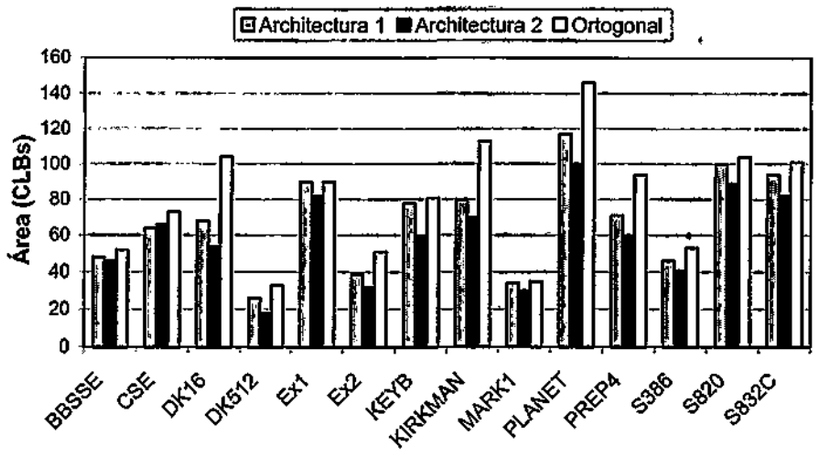


Figura 4.19. Área de la partición ortogonal respecto de las arquitectura 1 y 2

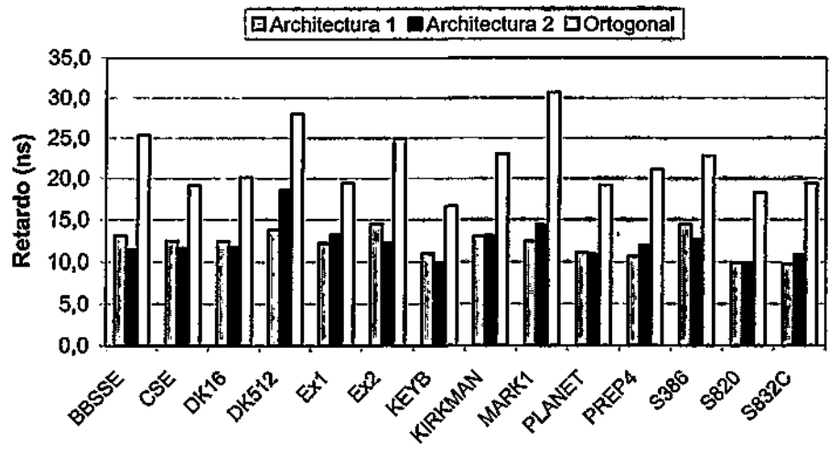


Figura 4.20. Retardo de la partición ortogonal respecto de las arquitectura 1 y 2

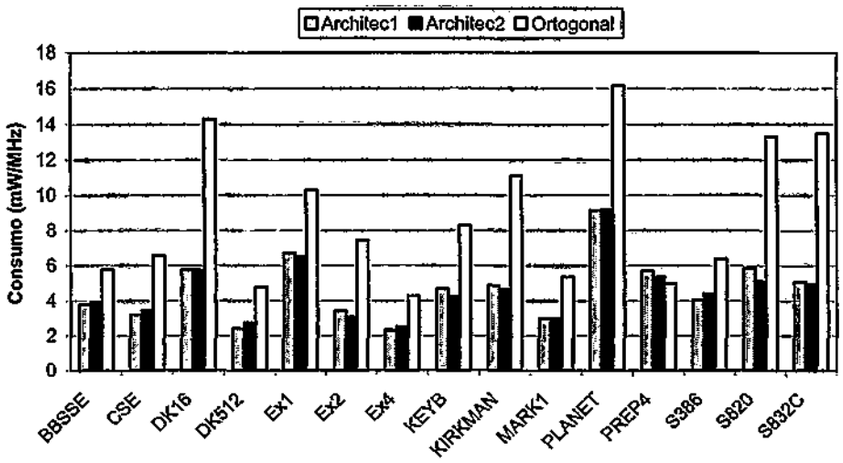


Figura 4.21. Consumo de la partición ortogonal respecto de las particiones según la arquitectura 1 y 2.

4.3.11 Conclusiones de la partición de máquina de estado en FPGAs

Esta sección explora los métodos de partición de máquinas de estados para reducir consumo en FSMs implementadas en FPGAs. La principal conclusión es la utilidad de esta técnica, desarrollada para circuitos basados en células, y que puede ser adaptado con éxito en FPGAs. Se pueden obtener importantes ahorros en el consumo de energía con esta técnica, llegando hasta un 54 %. Estos resultados pueden mejorar significativamente en dispositivos que disponen de *latches* nativos (Xilinx XC4000EX, Spartan2/3, Virtex/-II).

El esquema de codificación en las submáquinas juega un importante papel: los esquemas basados en codificación binaria funcionan bien por submáquinas pequeñas (hasta ocho estados); en tanto que para máquinas más grandes la mejor opción es *One Hot*.

Por otra, cabe destacar que la alternativa de partición ortogonal, que si bien disminuye la cantidad de estados de las submáquinas, siempre ha dado peores resultados en área, velocidad y consumo que la técnica aquí propuesta.

Por ultimo, cabe señalar que los buenos resultados en cuanto al bajo consumo dependen de la actividad entre las submáquinas de estados. Ésta técnica resulta ineficaz en aquellas máquinas de estado donde no se puede generar particiones donde la actividad entre las submáquinas sea pequeña.

4.4 Recomendaciones para la reducción de consumo en máquinas de estado

Los experimentos realizados en este capítulo, permiten generar la siguiente heurística para la minimización del consumo en máquinas de estados. La primera recomendación, aunque trivial, es llevar a cabo un diseño minimal de la máquina de estados. Existen innumerables programas, mayoritariamente de libre distribución, para la minimización de estados, pero un diseño cuidadoso los hace prescindibles.

Otra discusión siempre vigente en el diseño de FSM es la utilización de máquina de Mealy o Moore. Las máquinas de Moore aunque son más grandes su sincronismo con el reloj las hace más adecuada para los diseños síncronos y al no producir *glitches*, adecuadas para el bajo consumo. Existe no obstante, para el caso que la representación de Mealy es extremadamente más pequeña que su equivalente de Moore, la posibilidad de generar máquinas de Mealy síncronas como alternativa viable al bajo consumo.

Una vez, definida la máquina de estados y en función de la cantidad de estados existen diferentes alternativas: Si la máquina de estados es pequeña no superando los ocho estados, las codificaciones binarias son la mejor alternativa. La codificación óptima depende de las transiciones más probables dentro de la FSM. No obstante, con transiciones con la misma probabilidad se demostró una clara correlación área-consumo, pudiéndose utilizar el área como métrica para la mejor codificación.

Para máquina de estado entre ocho y dieciséis estados no existe una regla clara respecto al tipo de codificación a utilizar, no obstante para más de dieciséis estados *One-Hot* es mejor alternativa que las codificaciones binarias. La reducción de consumo debido a la correcta elección de la codificación puede llegar al 57 %.

Respecto de las máquinas grandes (más de dieciséis estados) las arquitecturas de particionamiento de máquinas de estado son una alternativa viable. La condición para que este método logre disminución en el consumo, es lograr realizar una partición de las máquinas de estados tal que la probabilidad de pasar de una máquina a otra sea pequeña. Para ejemplos concretos se han logrado disminuciones de hasta el 54 %.

4.5 Referencias del Capítulo

[Alc00] J. Alcalde, J. Rius and J. Figueras, "Experimental techniques to measure current, power and energy in CMOS integrated circuits", *Proceedings of XV Conference on Design of Circuit and Integrated Systems (DCIS'2000).* Montpellier, France, November 2000.

[Ald99] Aldec State Editor, a graphical FSM entry tool for Xilinx Foundations 2.x y 3.x, 1999. www.xilinx.com

[Ali94] M. Alidina, J. Monteiro, S. Devadas, A. Gosh, M. Papaefthymiou, "Precomputation-Based sequential logic optimization for Low-Power", *IEEE Very Large Scale Integration (VLSI) Systems Journal* , Vol. 2, num.4, pp.426-435, December 1994.

[Ash91] P.Ashar, S.Devadas, and A.Newton, "Optimum and heuristic Algorithms for an Approach to FSM Decomposition," *IEEE Transaction on Computer-Aided Design*, 10(3):296-310, March 1991.

[Ben95a] Lucca Benini and Giovanni De Micheli, "State Assignment for Low Power Dissipation," *IEEE Journal of Solid State Circuits*, Vol. 30, No. 3, pp. 258-268, March 1995.

[Ben95b] L. Benini, and G. De Micheli, "Transformation and Synthesis of FSMs for Low Power Gated Clock Implementation", *Proceedings of ISLP'95 International Symposium on Low Power Design, ACM-SIGDA and IEEE-CAS*, April, 1995.

[Ben96] L.Benini P.Siegel and G.De Micheli. "Automatic synthesis of low-power gated-clock finite-state machines," *IEEE Transaction on Computer Aided Design of Integrated Circuit*, vol.15, Issue 6, pp. 630– 643, June 1996.

[Ben98] L. Benini, G. De Micheli, and F. Vermeulen. "Finite-State Machine Partitioning for Low Power," *In Proceeding of IEEE International Symposium on Circuits and Systems (ISCAS '98)*, volume 2, pages 5-8, Monterey, California, May-June, 1998.

[Cho96] S-H.Chow, Y-C.Ho, and T.Hwang. "Low Power Realization of Finite State Machines Decomposition Approach," *ACM Transaction on Design Automation in Electronic Systems*, 315-340, July 1996.

[Dem88] G. De Michelli, R.K. Briton y A. San Giovanni-Vincentelli. "Optimal State Assignment for Finite State Machines," *IEEE transaction on CAD*, vol. CAD-4, pages 269-284, July 1985.

[Dev88] Devadas, S., Ma, H., Newton, A., and Sangiovanni-Vincentelli, A. "MUSTANG: State assignment of finite state machines targeting multilevel logic implementations," *IEEE Transaction on Computer-Aided Design* 7, 12 December), 1988.

[Dun97] J.Dunoyer, F.Pétrot, L.Jacomme. "Intrinsic Limitations of Logarithmic Encodings for Low Power Finite State Machines," *Mixed Design of VLSI Circuit Conference*, pp. 613-618, Pologne, 1997.

[Gei91] M.Geiger T.Müller-Wipperfürth, "FSM Decomposition Revisited: Algebraic Structure Theory Applied to MCNC Benchmark FSMs," *28th ACM/IEEE Design of Automation Conference (DAC '91)*, Amsterdam, The Netherlands, February 1991.

[Hac91] G.D. Hachtel, J.-K. Rho, F. Somenzi, and R. Jacoby. "Exact and Heuristic Algorithms for the Minimization of Incompletely Specified State Machines," *In Proceedings of the European Conference on Design Automation*, pages 184–191, Amsterdam, The Netherlands, February 1991.

[Hac94] G. Hachtel, J.K. Rho, F. Somenzi, and R. Jacoby, "Exact and heuristic algorithms for the minimization of incompletely specified state machines," *IEEE Transaction on Computer Aided Design (CAD)* 13 (2) pp167-177, February 1994.

[Har60] Juris Hartmanis, "Symbolic Analysis of a decomposition of information processing," *Information Control*, 3: 154-178, June 1960.

[Lin89] B. Lin and A.R. Newton. "Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages," *In Proceedings of International Conference on VLSI*, pages 187–196, August 1989.

[Lis88] Bob Lisanke. "Logic synthesis and optimization benchmarks," *Technical report, MCNC*, Research Triangle Park, North Carolina, December 1988.

[Mar00] M. Martínez, M. J. Avedillo, J. M. Quintana, M. Koegst, ST. Rulke, and H. Susse: "Low Power State Assignment Algorithm," *Proceedings of XV Conference on Design of Circuit and Integrated Systems (DCIS'2000)*, pp. 181-187. Montpellier, France, November 2000.

[Men03] Luis Mengibar, Luis Entrena, Michael G. Lorenz, and Raúl Sánchez-Reillo, "State Encoding for Low-Power FSMs in FPGA," PATMOS 2003, LNCS 2799, Springer-Verlag, Berlin, pp. 31–40, 2003.

[Men99] L. Mengíbar, M. García, D. Martín, and L. Entrena, "Experiments in FPGA Characterization for Low-power Design", *Proceedings of XIV Conference on Design of Circuit and Integrated Systems (DCIS'99)*, Palma de Mallorca, November 1999.

[Mon98] J. Monteiro, A. Oliviera, "Finite State Machine Decomposition for Low Power", *Proceedings 35th Design Automation Conference (DAC'98)*, San Francisco, pp. 758-763, 1998.

[New88] A. R. Newton, S. Devadas, Hi-Keung Ma, A. San Giovanni-Vincentelli. "MUSTANG: State Asignament of Finite State Machine Targeting Multilevel Logic Implementation," *IEEE transaction on Computer Aided Design (CAD)*. December 1988.

[Not99] Winfried Nöth and Reiner Kolla, "Spanning Tree Based State Encoding for Low Power Dissipation," *In Proceedings of Design, Automation and Test in Europe (DATE '99)*, pp 168-174, Munich, Germany, March 1999.

[Pau59] M.C. Paull and S.H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions," *IRE Transactions on Electronic Computers*, EC-8:356–367, September 1959.

[Pre00] Programmable Electronics Performance Corporation (PREP) Benchmarks (Programmable Electronics Performance Company), Avilable from: http://www.prep.org.

[San90] A. San Giovanni-Vincentelli, T. Villa. "NOVA: State Asignament of Finite State Machines for Optimal Two Level Logic Implementation," *IEEE transaction on Computer Aided Design (CAD)*, September 1990.

[Sen92] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Seq. Circuit Synthesis". *Technical Report Mem. No. UCB/ ERL M92/41, Universidad of California, Berkeley*, 1992.

[She00] R.Shelar, H.Narayanan, M.Desai, "Orthogonal Partitioning and Gated Clock Architecture for Low Power Realization of FSMs", *IEEE Int ASIC/SOC conference.* Sep 2000, pp. 266-270.

[she99] R.Shelar, M.P. Desai, H.Narayanan, "Decomposition of Finite State Machines for Area, Delay Minimization," *IEEE Conference on Computer Design (ICCD99)* Austin, 10-13, pp. 620-625 Oct. 1999.

[Sta02] StateCAD 5.03, a graphical entry tool for FSM. distributed with Xilinx ISE foundations tolls 4.x, www.xilinx.com

[Sut02a] G. Sutter and E. Boemo. "Low Power Finite state machines in FPGA: Bynary vs One hot encoding," *VIII Workshop Iberchip*, Guadalajara, Mexico, April 2002.

[Sut02b] G. Sutter, E. Todorovich, S. Lopez-Buedo, and E. Boemo, "Low-Power FSMs in FPGA: Encoding Alternatives", Lecture Notes in Computer Science, Vol.2451, pp.363-370, Berlin: Springer-Verlag, sept 2002.

[Syn99] Synopsis, FPGA Express User Guide version 2.3, 1999. FPGA Express home page; http://www.synopsys.com/products/ fpga/fpga_express.htm

[Tek02] Tektronix inc., "TLA 700 Series Logic Analyzer User Manual", 2002, available at http://www.tektronix.com.

[Tod00] E. Todorovich, G. Sutter, N. Acosta, E. Boemo and S. López-Buedo, "End-user low-power alternatives at topological and physical levels. Some examples on FPGAs", *Proceedings of XV Conference on Design of Circuit and Integrated Systems (DCIS'2000).* Montpellier, France, November 2000.

[Tsu94a] C-Y Tsui, M.Pedram, A. Despain, "Exact and Approximate Methods for Calculating Signal and Transition Probabilities in FSMs", *31st ACM/IEEE*

Design Automation Conference (DAC'94)., pp. 18-23, San Diego, CA, USA. June 1994.

[Tsu94b] Chi-Ying Tsui, Massoud Pedram, Chih-Ang Chen, and Alvin Despain. "Low Power State Assignment Targeting Two- and Multi-level Logic Implementations", *Proceedings of ACM/IEEE International Conf. of Computer-Aided Design*, pp. 82-87, November 1994

[Vil90] T.Villa, A.Sangiovanni-Vincentelli, "NOVA: State assignment for finite state machines for optimal two-level logic implementation", *IEEE Transaction on CAD*, Vol.9-9, pp.905, Sept. 1990.

[Wu00] X. Wu, M. Pedram, and L. Wang, "Multi-code State Assignment for Low Power Design", IEEE *Proceedings-Circuits, Devices and Systems*, Vol.147, No.5, pp.271-275, Oct. 2000.

[Xil00a] Xilinx inc. "Xilinx software manual, Synthesis and Simulation Design Guide: Encoding State Machines". www.xilinx.com. 2000.

[Xil00b] Xilinx Inc, "Xilinx Foundation Tools F3.1i user Guide", 2000, available at www.xilinx.com/support/library.htm

[Xil03] Xilinx inc, "Data Feedback and Clock Enable", *Development system design guide; Chapter 2: Design Flow*, 2003.

[Xil99] Xilinx Inc. "XC4000E and XC4000X Series Field Programmable Gate Arrays", Product Specification (Version 1.6), May, 1999.

[Yan91] Saeyang Yang. "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0" *Technical report, MCNC*, Research Triangle Park, North Carolina, January 1991.

Capítulo 5:

Experimentos sobre Bloques Aritméticos

Este capítulo examina alternativas a nivel algorítmico para diferentes bloques aritméticos. Por un lado se examinan las opciones para la multiplicación modular, operación central en los cálculos criptográficos, más tarde se presenta un experimento cuyos resultados poseen un efecto inesperados desde el punto de vista del consumo como es la suma por el algoritmo de *carry-skip*, a continuación, se examinan los algoritmos y arquitecturas para la división entera. Por último, se analizan las alternativas para la división de números fraccionarios, operación típicamente llevada a cabo sobre la mantisa en las operaciones de división de números en coma flotante. Se presentan resultados para más de 250 circuitos y mediciones del consumo en más de 130 casos.

5.1 Introducción

En la sección 2.7 se reconocía el efecto directo que poseen la complejidad, la precisión y la regularidad de los algoritmos sobre el consumo. Otro aspecto importante, aunque más sutil, es reconocer cuan bien se adecua un algoritmo a una arquitectura de bajo consumo (como las explicadas en las sección 2.5 y 2.6). Para una eficiente implementación de los algoritmos orientados al bajo consumo, existen características deseables como son la concurrencia y la modularidad de los algoritmos.

Existen asimismo otras características, aún más sutiles a tener en cuenta, como son la capacidad de producir o no movimientos espurios (*glitches*) dentro de la ruta de datos, la representación de los datos utilizados por el algoritmo, así como la correlación de datos que se producen producto del algoritmo utilizado.

5.2 Multiplicación modular

Se describen en está sección tres algoritmos de multiplicación modular: multiplicación y reducción, sumas y desplazamientos, y por último multiplicación de Montgomery. Se brindan estimaciones de los costes correspondientes tanto para las versiones combinacionales como secuenciales. Por último se muestran los resultados prácticos para implementaciones combinacionales, y secuenciales utilizando dispositivos programables de la familia XC4K.

5.2.1 Introducción a la multiplicación modular

Gran parte de los algoritmos criptográficos de clave pública utilizan, como primitivas de cálculo, operaciones en un anillo finito (Z_m) o en un cuerpo finito ($GF(p^k)$). Es el caso del RSA [Riv78] y de los algoritmos basados en curvas elípticas [Bla99]. Por tanto la generación de modelos VHDL de multiplicadores modulares es un punto de partida importante. Por lo general los multiplicadores modulares que, a su vez, sirven para el cálculo de la función exponencial $y^x \ mod \ m$, se sintetizan en base al algoritmo de Montgomery [Mon85]. Es un método eficiente para la generación de procedimientos que ejecutan la exponenciación modular [Men96], y ha sido utilizado frecuentemente para la realización en hardware de parte del cifrado y descifrado, así como de la generación de las claves, del RSA [Fis01], [Man01], [Blu99]. Sin embargo es un método poco eficiente cuando se trata de multiplicaciones modulares propiamente dichas [Men96]. En esta sección se proponen y comparan los tres algoritmos antes mencionados: multiplicación y reducción, sumas y desplazamientos, multiplicación de Montgomery.

La sección 5.2.2 describe los algoritmos de multiplicación modular utilizados. La sección 5.2.3 brinda detalles de la síntesis, y en la siguiente sección hace lo propio para las implementaciones secuenciales. Por último, la sección 5.2.5 presenta los resultados en consumo.

5.2.2 Algoritmos para la multiplicación modular

Dados tres números naturales x, y y m, y una cantidad de dígitos n, tales que:

$$x < m, y < m \text{ y } m < 2^n,$$

Se proponen tres algoritmos – con los circuitos correspondientes – para calcular:

$z = x.y \bmod m$.

5.2.2.1 Multiplicación y reducción (*Multiply and Reduce*)

El primer algoritmo consiste en (1) multiplicar x por y, generando así un resultado intermedio p de $2n$ bits, y (2) reducir p módulo m.

La multiplicación de dos números naturales se descompone en una serie de desplazamientos hacia la izquierda y de sumas condicionales. Es el clásico algoritmo de sumas y desplazamientos (*shift and add*).

Algoritmo 5.2.1 – multiplicación por sumas y desplazamientos

```
p := 0;
for i in 0 .. n-1 loop
        p :=( p + x(i)*y)/2;
end loop;
p := p*(2**n);
```

La reducción módulo m se descompone en una secuencia de desplazamientos hacia la izquierda, restas y bifurcaciones. El algoritmo con restauración (*restoring*) es simular a la división a mano:

Algoritmo 5.2.2 – reducción con restauración.

```
module := m*(2**n);
r(0) := p;
for i in 1 .. n loop
        remainder := (2*r(i-1))-module;
        if remainder < 0 then
                r(i) := 2*r(i-1);
        else r
                (i) := remainder;
        end if;
end loop;
z := r(n) / 2**n;
```

Un algoritmo diferente sin restauración (*non-restoring*) utiliza como primitiva de cálculo un sumador / restador cuya operación depende de una condición binaria previamente calculada:

Algoritmo 5.2.3 – reducción sin restauración

```
module := m*(2**n);
r(0) := (2*n) - module;
for i in 1 .. n-1 loop
        if r(i-1)<0 then
                r(i) := (2*r(i-1))+ module;
        else
                r(i) := (2*r(i-1))- module;
        end if;
end loop;
z := r(n-1) / (2**n);
if z < 0 then z := z + m; end if;
```

Obsérvese que el algoritmo con restauración incluye una bifurcación basada en el bit de signo de un resultado (*remainder*) calculado durante la misma etapa de la iteración, en tanto que la bifurcación incluida en el algoritmo sin restauración se basa en el bit de signo de un resultado (*r(i-1)*) calculado durante la etapa anterior.

5.2.2.1 Suma y desplazamiento (*shift and add*)

En lugar de multiplicar, y luego reducir módulo *m* el número de 2.*n* bits así obtenido, otra opción consiste en ejecutar la multiplicación empezando con el bit más significativo de *x* y reducir en cada etapa:

Algoritmo 5.2.4 – reducción en cada etapa

```
z := 0;
for i in 1 .. n loop
   z := (z*2 + x(n-i)*y) mod m;
end loop;
```

El valor máximo de *z.2+x(n-i).*y es

$2.(m\text{-}1) + (m\text{-}1) = 3.(m\text{-}1)$.

Por tanto

$$2.z + x(n\text{-}i).y = m.q + r$$

Donde $q \in \{0,1,2\}$, y el cálculo de $z.2 + x(n\text{-}i).y$ módulo m puede realizarse de la forma siguiente:

Algoritmo 5.2.5

```
p1 := z*2;
p2 := p1 + x(n-i)*y - m;
if p2 < 0 then
        p3 := p2 + m;
        z := p3;
else
        p3 := p2 - m;
        if p3 < 0 then
                z := p2;
        else
                z := p3;
        end if;
end if;
```

Se puede simplificar el algoritmo 5.2.5. Por un lado $p2$ y $p3$ no pueden ser negativos al mismo tiempo:

$$p2 = 2.z + x(n\text{-}i).y - m$$

Con lo cual $-m \leq p2 < 2.m$; si $p2 < 0$ entonces

$$p3 = p2 + m \geq -m + m = 0.$$

Por otra parte, en lugar de calcular

$$p2 = p1 + x(n\text{-}i).y - m,$$

Se puede evaluar $k = m.y$ de antemano (fuera de la iteración) de tal manera que $p2 = p1\text{-}m$ ó $p2 = p1\text{-}k$. El algoritmo así obtenido es el siguiente:

Algoritmo 5.2.6 – suma y desplazamiento

```
z := 0; k := m-y;
for i in 1 .. n loop
        if x(n-i) = 0 then  w := m;
        else  w := k;
        end if;
        p1 := z*2;   p2 := p1 - w;
        if p2<0 then p3 := p2+m;
        else  p3 := p2-m;
        end if;
        if p3<0 then z := p2;
        else  z := p3;
        end if;
end loop;
```

5.2.2.3 Multiplicación de Montgomery

Si m es impar, el mayor común divisor de 2^n y m es igual a 1, con lo cual existe un número natural, representado como 2^{-n}, tal que $2^{-n}.2^n = 1 \bmod m$. Montgomery [Mon85] propuso un algoritmo que calcula:

$$z = x.y.\ 2^{-n} \bmod m,$$

el así llamado producto de Montgomery. Cada etapa de la iteración consta de dos sumas condicionales:

Algoritmo 5.2.7 – algoritmo de Montgomery

```
r(0) := 0;
for i in 1 .. n loop
        a := r(i-1) + x(i-1) * y;
        r(i) := (a + a(0) * m) / 2;
end loop;
if r(n) < m then  z := r(n);
else  z := r(n) - m;
end if;
```

El algoritmo no calcula $x.y \bmod m$. Sin embargo obsérvese que si x, y y $z = x.y \bmod m$, son sustituidos por $x' = x.2^n \bmod m, y' = y.2^n \bmod m$ y $z' = z.2^n \bmod m$, entonces

$$z' = x'.y'.2^{-n} \bmod m.$$

Ello significa que z' es el producto de Montgomery de x' y y'. Dicho de otra forma se puede definir una transformación basada en la aplicación de Z_m en Z_m definida por $a \rightarrow a.2^n \bmod m$, de tal manera que dentro del dominio transformado el producto módulo m sea sustituido por el producto de Montgomery. Obsérvese que la transformación directa $a \rightarrow a.2^n \bmod m$ es equivalente a la multiplicación de Montgomery por $2^{2n} \bmod m$, y la transformación inversa $a' \rightarrow a'.2^{-n} \bmod m$ a la multiplicación de Montgomery por 1.

Considérese ahora el clásico algoritmo de exponenciación, basado en una secuencia de multiplicaciones, que calcula $e = y^x$:

Algoritmo 5.2.8

```
e := 1;
for i in 1 .. n loop
   e := e*e;
   if x(n-i) = 1 then
      e := e*y;
   end if;
end loop;
```

Para calcular $e = y^x \bmod m$, se modifica el algoritmo anterior de la forma siguiente: 1 e y son sustituidos por $1.2^n \bmod m$ e $y.2^n \bmod m$, el producto de los enteros por el producto de Montgomery, y el resultado final e por $e.2^{-n} \bmod m$. Supóngase que los valores de $one\_m = 2^n \bmod m$ y de $two\_m = 2^{2n} \bmod m$ hayan sido previamente calculados (para todos los valores útiles de m).

Entonces el siguiente algoritmo, en el cual *MM* es un procedimiento que ejecuta la multiplicación de Montgomery, calcula $e = y^x \bmod m$:

Algoritmo 5.2.9 - exponenciación por Montgomery

```
e := one_m;
y := MM(y, two_m);
for i in 1 .. n loop
        e := MM(e,e);
        if x(n-i) = 1 then  e := MM(e,y);
        end if;
end loop;
e := MM(e,1);
```

5.2.3 Detalle de la síntesis

Los tres tipos de algoritmos de multiplicación modular, han sido sintetizados en forma de circuitos combinacionales e integrados en FPGA de la familia XC4K. Para la síntesis de los algoritmos (5.2.1 y 5.2.2 o 5.2.3, 5.2.6 y 5.2.7) se necesitan las primitivas de cálculo siguientes:

suma: $r : a+b$
resta: $r = a + (2n - b)$
suma / resta: $r = a + (1-x).b + x.\ (2n - b)$
suma condicional: $r = a + x.b$
resta condicional: $r = a + x.\ (2n - b)$
selección: $r = (1-x).a + x.b$

Donde a y b son números de n bits y x un número de un solo bit. Todas se sintetizan con $n/2 + 1$ bloques lógicos configurables (CLBs) de la familia XC4000, excepto la selección (multiplexor 2-1 de n bits) que se sintetiza con $n/2$ CLBs.

Obsérvese que el coste de un multiplexor 2-1 es prácticamente el mismo que el de un sumador / restador ($n/2$ vs. $n/2 + 1$), es decir, el de n LUTs. La conclusión sería bastante diferente en el caso de la versión *Standard Cell* de las mismas primitivas.

5.2.3.1 Multiplicación y reducción

La multiplicación (algoritmo 5.2.1) incluye n sumas condicionales. El coste correspondiente es igual a $n.(n/2 + 1)$ CLBs. Para ejecutar la reducción con el algoritmo 5.2.2 (con restauración) se necesitan n restadores condicionales y n multiplexores, mientras que el algoritmo 5.2.3 (sin restauración) sólo incluye $n-1$

sumador-restador, una resta inicial y una suma condicional final. El coste correspondiente es $(n+1).(n/2 + 1)$ CLBs. La Figura 5.1 muestra las celdas elementales y un esquema del reductor modulo m con el algoritmo 5.2.3. El coste total es igual a:

$C_{\text{multiplicación y reducción}} = n^2 + 2{,}5.n + 1$ CLBs.

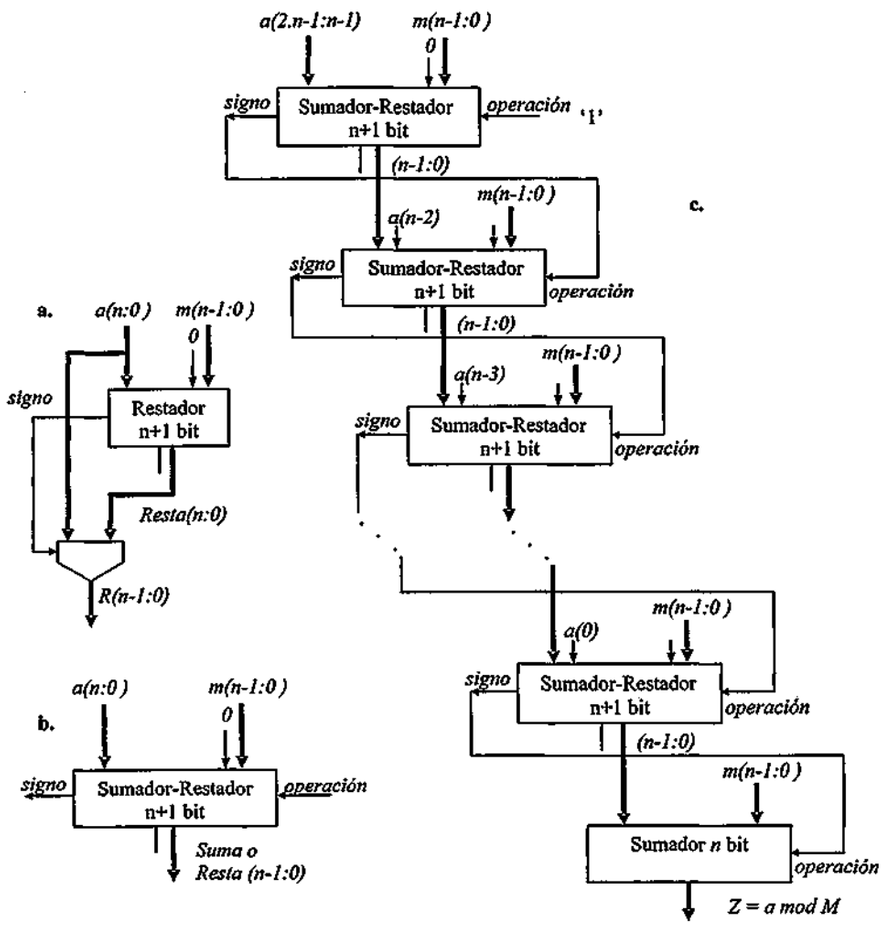


Figura 5.1. Reductor módulo *M*. a. Celda elemental del algoritmo con restauración. b. Celda elemental algoritmo sin restauración c. Implementación combinacional algoritmo sin restauración.

5.2.3.2 Sumas y desplazamientos

Para la ejecución del algoritmo 5.2.6 se necesitan:

un primer restador (cálculo de *k*),
n multiplexores (selección de *m*),
n restadores de *n*+1 bits (cálculo de *p2*; es necesario añadir un bit para poder detectar el signo de *p2*),
n sumadores / restadores de *n*+1 bits (cálculo de *p3*; se necesita un bit adicional para detectar el signo de *p2*),
n multiplexores (selección de *z*).

El coste correspondiente es igual a $n/2 + 1 + n.(n/2 + (n+1)/2 + 1 + (n+1)/2 + 1 + n/2)$, es decir:

$$C_{\text{sumas y desplazmientos}} = 2.n^2 + 3{,}5.n + 1 \text{ CLBs.}$$

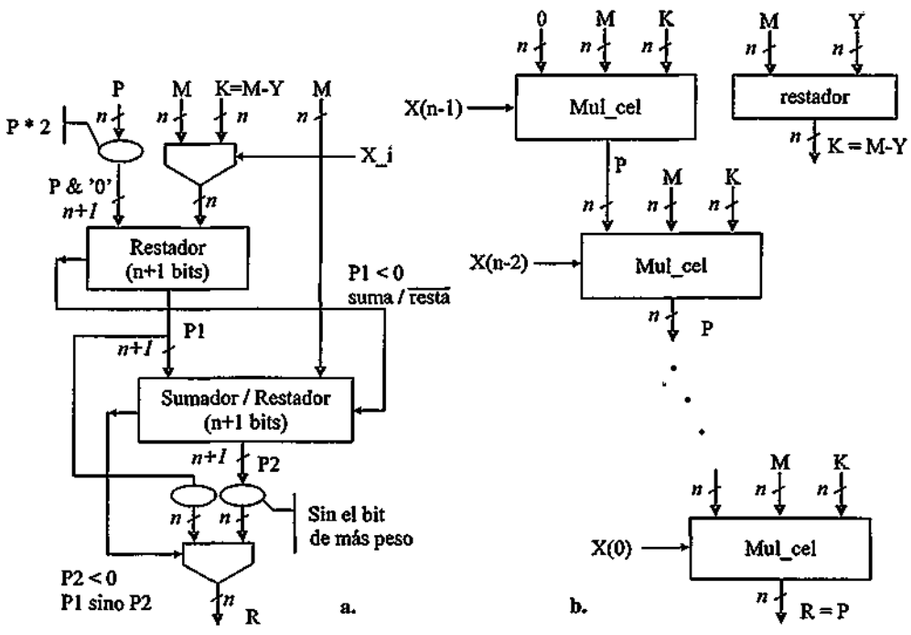


Figura 5.2. Multiplicador modular *shift and add*. a. Celda de cálculo. b. arreglo combinacional.

5.2.3.3 Multiplicación de Montgomery

El algoritmo 5.2.7 incluye n sumas condicionales de n+1 bits (*a*), n sumas condicionales de n+2-bits (*r(i)*), una resta de n+1 bits (*r(n)* - *m*), un multiplexor de n bits (selección de *z*). El coste correspondiente es igual a $n.((n+1)/2 + 1 + (n+2)/2 + 1) + (n+1)/2 + 1 + n/2$, es decir:

$C_{Montgomery} = n^2 + 4{,}5.n + 1{,}5$ CLBs.

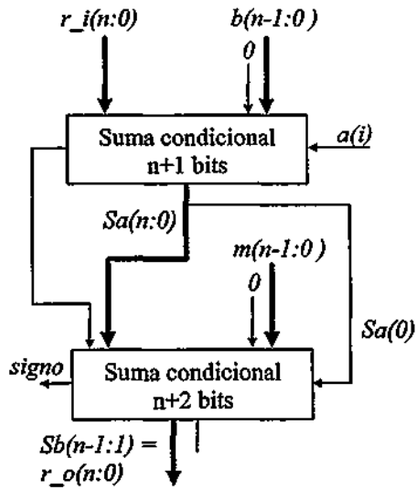


Figura 5.3. Celda de cálculo del multiplicador de Montgomery.

5.2.3.4 Comparación en área y velocidad

Los tres tipos de multiplicadores modulares (*m_r*: multiplicación y reducción, *s_d*: sumas y desplazamientos, *mont.*: multiplicador de Montgomery) han sido integrados en dispositivo programable de la serie XC4000. La descripción inicial es un modelo VHDL sintetizable utilizando los paquetes de funciones aritméticas del IEEE. Los sumadores-restadores, sumadores condicionales, restadores condicionales y multiplexores se modelaron con sentencias *if then else*:

```
if x='0' then r <= a + b; else r <=a - b; end if;
if x='0' then r <= a; else r <= a + b; end if;
if x='0' then r <= a; else r <= a - b; end if;
if x='0' then r <= a; else r <=  b; end if;
```

En la tabla 5.1 se presenta el número de CLBs y el retardo máximo (en *ns*), sólo para 8 y 16 bits dado que los multiplicadores de 24 y 32 bits no caben dentro de la matriz elegida.

	Área (CLBs)			Retardo (ns)		
bits	*m_r*	*s_d*	*mont.*	*m_r*	*s_d*	*mont.*
8	85	157	102	186	201	167
16	297	563	334	454	724	325
24	637	1232	694	-	-	-
32	1104	2160	1166	-	-	-

Tabla 5.1. Número de CLBs y Retardo Máximo (ns) para los multiplicadores modulares secuenciales

Obsérvese que los costes reales son muy parecidos a los que se habían calculado. Las conclusiones prácticas son las siguientes:

1. Los costes de los multiplicadores *m_r* (multiplicación y reducción) y *mont.* (producto de Montgomey) son casi idénticos ($n^2 + 2{,}5.n + 1$ vs. $n^2 + 4{,}5.n + 1{,}5$). Sin embargo el multiplicador de Montgomery es más rápido.
2. El coste del multiplicador *s_d* (sumas y desplazamientos) es casi el doble del de los anteriores ($2.n^2 + 3{,}5.n + 1$). Ello se debe al alto coste relativo de los multiplexores.

El algoritmo por sumas y desplazamientos debe ser descartado, por lo menos en el caso de un circuito combinacional integrado dispositivos de la familia XC4K. A la hora de elegir entre "Multiplicación y Reducción" y "Multiplicación de Montgomery" se deben tener en cuenta los comentarios siguientes:

1. El producto de Montgomery (algoritmo 5.2.7) es más rápido.
2. El cálculo de la función exponencial con la multiplicación de Montgomery (algoritmo 5.2.9) necesita el cálculo previo de $2^n \bmod m$ y de $2^{2.n} \bmod m$ para todos los valores de m susceptibles de ser utilizados. Dichos valores podrían ser almacenados en la memoria del sistema. Una solución alternativa consiste en reconfigurar la matriz de forma específica para cada valor de m.
3. El multiplicador de Montgomery no calcula $z = x.y \bmod m$ sino $z'' = x.y.2^{-n} \bmod m$. Para obtener z a partir de z'' es necesario haber calculado previamente

el valor de $2^{2n} \bmod m$ y se debe ejecutar un segundo producto de Montgomery dado que $z = z''.2^{2n}.2^{n} \bmod m$. Por tanto el método de Montgomery es ineficiente para realizar una multiplicación modular aislada (cuando no sirve para calcular la función exponencial modular).

5.2.4 Realización secuencial

Para grandes valores de n el circuito debe ser (por lo menos parcialmente) secuencializado. Los tres tipos de multiplicadores modulares han sido sintetizados y comparados en base a las hipótesis siguientes: el circuito, completamente secuencializado, ejecuta el cuerpo de la iteración principal de los algoritmos 5.2.1 y 5.2.3 (*m_r*), 5.2.6 (*s_d*) y 5.2.7 (*mont.*), respectivamente; además contiene registros para almacenar las variables del algoritmo, un contador y la lógica de control. El circuito se integra en la misma FPGA que antes.

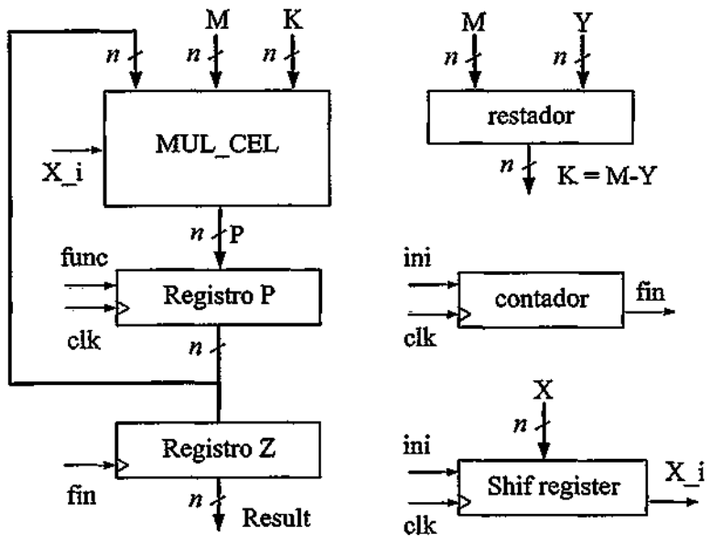


Figura 5.4. Esquema de la implementación secuencial del algoritmo de sumas y desplazamientos.

Para la ejecución secuencial de los algoritmos, los recursos adicionales que se necesitan son: registros, registros de desplazamiento y contadores. Cada CLB de la familia XC4K contiene dos biestables con lo cual los registros de n bits se sintetizan con $n/2$

CLBs. Para los contadores compuestos de n biestables (2^n estados) el número mínimo de CLBs es igual a $n/2$. En el caso de los contadores más complejos (bidireccionales, programables, con habilitación del reloj) se necesitan CLBs adicionales. Como regla práctica se supone que el coste de un contador de n bits es del orden de n CLBs. En la Figura 5.4 se puede ver un esquema de la implementación secuencial del algoritmo de sumas y desplazamientos.

Para la síntesis secuencial de los algoritmos 5.2.1 y 5.2.2 (multiplicación y reducción) se usan:

un sumador condicional de n bits,
un sumador / restador de n bits,
un sumador condicional de n bits (etapa final),
un contador de $2.n$ estados,
dos registros de desplazamiento de n bits,
un registro de $2.n$ bits,
una máquina de cuatro estados.

El coste correspondiente es del orden de $n/2 + 1 + n/2 + 1 + n/2 + 1 + \log_2(2.n) + 2.(n/2) + (2.n)/2 + 4$, es decir:

$$C_{\text{multiplicación y reducción}} = 3{,}5.n + \log_2 n + 8.$$

Para la síntesis secuencial del algoritmo 5.2.6 (sumas y desplazamientos) se utilizan:

un restador de n bits,
dos multiplexores de n bits,
un restador de $n+1$ bits,
un sumador / restador de $n+1$ bits,
un contador de n estados,
un registro de desplazamiento de n bits,
un registro de n bits.

El coste correspondiente es del orden de $n/2 + 1 + 2.n/2 + (n+1)/2 + 1 + (n+1)/2 + 1 + \log_2 n + n/2 + n/2$, es decir,

$$C_{\text{sumas y desplazamiento}} = 3{,}5.n + \log_2 n + 4.$$

La síntesis del algoritmo 5.2.7 (Montgomery) se hace con:

un sumador condicional de n+1 bits,
un sumador condicional de n+2 bits,
un restador de n+1 bits,
un multiplexor de n bits,
un contador de n estados,

un registro de desplazamiento de n bits,
un registro de n+1 bits.

El coste correspondiente es del orden de $(n+1)/2 + 1 + (n+2)/2 + 1 + (n+1)/2 + 1 + n/2 + \log_2 n + n/2 + (n+1)/2$, es decir

$$C_{Montgomery} = 3.n + \log_2 n + 5{,}5.$$

Los resultados de la implementación en FPGAs se recogen en las Tabla 5.2 (número de CLBs y frecuencia máxima en MHz). El número total de ciclos es igual a n para los multiplicadores *s_d* (sumas y desplazamientos) y *mont.* (Montgomery), y a $2.n$ para los multiplicadores *m_r* (multiplicación y reducción). Obsérvese que los costes reales son muy parecidos a los calculados.

	Área (CLBs)			Retardo (ns)		
bits	*m_r*	*s_d*	*Mont.*	*m_r*	*s_d*	*Mont.*
8	57	33	34	25	17,2	32,1
16	72	63	59	22,4	12,7	25,8
32	126	119	108	16,9	7,1	24,4
64	240	232	204	-	-	-
128	465	457	398	-	-	-
256	915	905	783	-	-	-

Tabla 5.2. Número de CLBs y Frecuencia máxima (MHz) para multiplicadores modulares secuenciales.

5.2.5 Consumo de potencia en la multiplicación modular

Para medir el consumo de los diferentes multiplicadores se utilizo el arreglo experimental y la metodología descrita en el Apéndice A. Adicionalmente cada circuito fue medido a 100 Hz, 2, 3, 4 y 5 MHz. El código VHDL fue sintetizado con FPGA express [Fpg99][Fpg01] y el entorno de desarrollo de Xilinx [Xil00b] en un dispositivo XC4010EPC84-4 [Xil99].

5.2.5.1 Consumo en implementaciones combinacionales

La entrada/salida de los circuitos secuenciales fue registrada, y un ancho de ocho bits en la ruta de datos fue elegido. En la tabla 5.3 se puede ver un resumen del consumo, área y retardo de los circuitos.

	M_r	*s_a*	*mont.*
Energía (nJoules)	96,0	186,4	92,7
Área (CLBs)	85	157	102
Retardo (ns)	186	201	167

Tabla 5.3. Área, retardo y consumo (Area-Time-Energy) de los multiplicadores modulares combinacionales

Se observa que la implementación de Montgomery (*mont*) consume menos potencia que multiplicar y reducir (*M_r*) a pesar de utilizar mayor área. El algoritmo de Montgomery posee aproximadamente un 4% menos de transiciones a la salida para el patrón de pruebas utilizado. Esto es a causa de que el algoritmo de Montgomery no computa $z = x.y \bmod m$ sino en realidad $z'' = x.y.2^{-n} \bmod m$.

Las medidas muestran que el algoritmo de multiplicar y reducir (*M_r*) y el algoritmo de Montgomery (*mont*), tienen no solo prácticamente la misma figura de área y retardo, sino que también similar consumo. No obstante el multiplicador de Montgomery es levemente más rápido y consume menos potencia. El consumo del algoritmo de desplazar y reducir (*s_a*), así como el área es alrededor del doble que los algoritmos anteriores.

5.2.5.2 Consumo en implementaciones secuenciales

En las implementaciones secuenciales se separó la potencia dinámica en dos componentes. Por un lado la potencia de sincronización (debida al reloj y los flip-flops) y por otro, la lógica combinacional (debida a la ruta de datos). Como se puede ver en los resultados de la tabla 5.4 la potencia de sincronización es lineal con la cantidad de registros utilizados.

	m_r	*s_a*	*mont.*
Consumo dinamico (nJoules)	71,5	52,4	38,6
Consumo de sincronización (nJoules)	46,8	26,2	27,2
Consumo ruta de datos (nJoules)	24,7	26,2	11,1
Área (CLBs)	57	33	34
Flip - Flops (Cantidad)	67	37	31
Máxima Frecuencia (Mhz)	25	17,2	32,1
Retardo total (ns)	320	465	249

Tabla 5.4. Área, retardo y consumo de los multiplicadores modulares secuenciales

La implementación secuencial del algoritmo de Montgomery consume menos potencia que las otras alternativas. El circuito de multiplicar y reducir tiene la peor figura de consumo y utiliza el doble de ciclos de reloj para computar el resultado.

Cabe destacar que la energía consumida medida en nanoJoules es menor en las implementaciones secuenciales que en las puramente combinacionales. Esto se justifica en el hecho de que las etapas de registros disminuyen el efecto avalancha de *glitches*. La potencia de sincronización en estos casos es mayor que la potencia combinacional.

Otro punto a destacar, es que en estas mediciones, tanto en circuitos secuenciales como combinacionales, se cumple la ley empírica de que el circuito más rápido es el que menos consume.

Por último en la Figura 5.5 se muestra la relación de área-retardo-consumo (ATP: *Area-Time-Power*) para los diferentes multiplicadores modulares. Donde el postfijo "*comb*" y "*sec*" en los circuitos se refiere a las implementaciones combinacionales y secuenciales respectivamente.

De la figura se puede extraer la mayor velocidad de los circuitos combinacionales a expensas de área y consumo. Por ejemplo el circuito Montgomery combinacional (c) el retardo disminuye en un factor 1,5 respecto de la versión secuencial (f), en tanto el área aumenta en un factor de 3 y el consumo en 2,5.

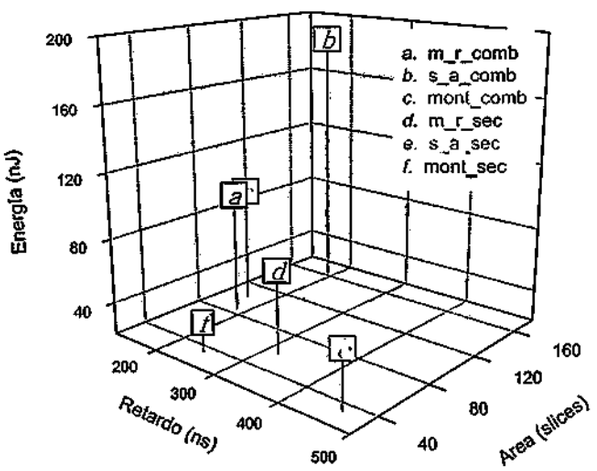


Figura 5.5. Área – retardo – consumo para los multiplicadores modulares

5.2.6 Sugerencias para la multiplicación modular

Para calcular $e = y^x \bmod m$, donde m pertenece a un conjunto conocido de valores (de tal manera que los valores de 2^n y $2^{2.n}$ módulo m puedan ser calculados de antemano), el algoritmo de Montgomery muestra la mejor figura de consumo (también área y retardo), independientemente del tipo de circuito (combinacional o secuencial).

Para calcular $z = x.y \bmod m$, la versión combinacional del algoritmo *m_r* (multiplicación y reducción) es mejor que la del algoritmo *s_d* (sumas y desplazamientos). En el caso de las versiones secuenciales ambos métodos dan resultados similares en cuanto al área y velocidad (teniendo en cuenta el hecho de que para multiplicar y reducir se necesitan $2.n$ ciclos en lugar de n), pero la potencia del desplazar y sumar (*s_d*) es claramente menor.

Desde el punto de vista del consumo, se prueba que la elección del algoritmo correcto puede dar reducciones de consumo del orden del 50% en los casos combinacionales (Montgomery vs. desplazar y sumar), del 54% en el caso secuencial (Montgomery vs. Multiplicar y reducir). El retardo de los los circuitos puede ser utilizado como métrica indirecta para determinar el algoritmo que menos consume.

Por otra parte la energía consumida para realizar la misma operación en las versiones secuenciales es menor que en las versiones combinacionales (hasta superar el 58% de reducción en el algoritmo de Montgomery), debido a la disminución de los *glitches* al reducir la profundidad lógica.

5.3. Sumadores de alta velocidad

En esta sección se analiza el algoritmo de adición de alta velocidad conocido como *carry skip*. De los algoritmos alternativos del sumador *ripple-carry*, el *carry skip*, parece por su estructura que ha de tener menos *glitches* producto de una menor propagación del acarreo y consecuentemente menos consumo. Basándose en esta idea y en el hecho, analizado y descrito en la sección 3.2, que los circuitos más veloces consumen menos, se llevaron a cabo varios experimentos. Sin embargo, las mediciones negaron la suposición inicial de la reducción de consumo. En la sección 5.3.3.3 se explica la causa.

Aunque las técnicas de *carry-look-ahead* (CLA) parecen ser más veloces que las de *carry skip* [Kor02] [Par00] [Obe01], en algunas tecnologías especificas esto no puede ser realmente explotado. De hecho la implementación del CLA en FPGAs no tiene buenos resultados [Hau00]. Por el contrario el uso de los canales de acarreo rápido (*fast carry propagation channels*) de las FPGAs son particularmente atractivos para implementar el *carry-skip*. Aquí se analizaran circuitos sumadores con operandos de hasta 1024 bits. La suma con grandes operandos es de gran utilidad en las operaciones criptográficas.

5.3.1 Sumador *ripple-carry*

Las FPGAs en general poseen recursos de computación para generar sumadores rápidos [Xil01] [Xil03a]. Por ejemplo, las familias actuales (Spartan 2/3 y Virtex /II) incluyen puertas lógicas y multiplexores que junto a las tablas de *look-up* de propósito general permiten construir sumadores de tipo *ripple-carry* de forma eficiente.

En la Figura 5.6 se puede ver una celda básica de un sumador. La cadena de acarreo conecta dos celdas de sumador dentro de un *slice* y luego con el *slice* adyacente vertical. Es decir, las cadenas de acarreo recorren verticalmente la FPGA de abajo a arriba, existiendo una cadena por columna de *slices*. En la celda de un sumador *ripple-carry* la tabla de *look-up* es utilizada para computar la función propagación del acarreo *p* (*carry-propagate*).

$$p(i) = x(i) \; xor \; y(i)$$

Con lo que el acarreo siguiente es

$$q(i+1) = g(i) + p(i).q(i),$$

Donde $g(i) = x(i).y(i)$ es la función de generación de acarreo (*carry-generate function*). Luego se tiene que el acarreo siguiente es:

$$q(i+1) = g(i)+p(i).q(i) = x(i).y(i)+(x(i) \text{ xor } y(i)).q(i) =$$
$$not(x(i) \text{ xor } y(i)).y(i) + (x(i) \text{ xor } y(i)).\ q(i) = not(p(i)).y(i) + p(i).q(i)$$

Con el uso de los multiplexores de acarreo y las conexiones dedicadas, el tiempo de computo del sumador de n bits ($T_{adder}(n)$) es aproximadamente:

$$T_{adder}(n) = t_{lut} + n.t_{muxcy}$$

Donde t_{lut} es el tiempo de cómputo de una LUT de propósito general y t_{muxcy} el retardo de los multiplexores de acarreo dedicados junto al retardo de la conexión al bloque adyacente. El valor de t_{muxcy} es mucho menor que la suma de los tiempos de un multiplexor generado con tablas de *look-up* y conexiones de propósito general. En cada slice se incluyen dos celdas de un sumador con lo que el costo en área de un sumador de n bits es:

$C_{adder}(n) = n/2$ slices.

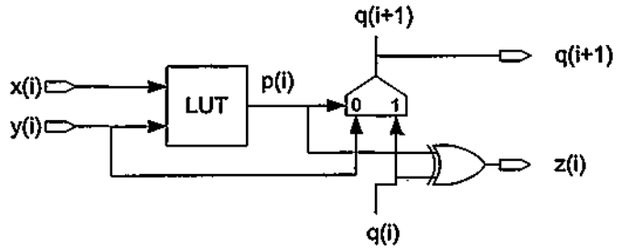


Figura 5.6. Celda básica de un sumador, que utiliza la cadena de acarreo en las FPGAs de Xilinx.

5.3.2 Sumador *carry-skip*

Dado el hecho de que $t_{lut} >> t_{muxcy}$, utilizar la lógica dedicada de acarreo es esencial para poder generar sumadores veloces. Como consecuencia las técnicas del tipo *carry-look-ahead* no pueden ser implementadas fácil y efectivamente. No obstante, las técnicas de *carry-skip* si pueden tener implementaciones eficientes. Para lograr éstas

implementaciones eficaces se utiliza en el cálculo de los productos *P(i)*, así como, para seleccionar la salida de cada grupo de *s*-bits la lógica dedicada de acarreo.

Un grupo sumador del *carry-skip* para *s*-bits se muestra en la Figura 5.7. El tiempo de computación y área utilizados son los tradicionales para un sumador *ripple-carry* normal.

$T_{adder\_group} = t_{lut} + s.t_{muxcy}$ y $C_{adder\_group} = s/2$ *slices*.

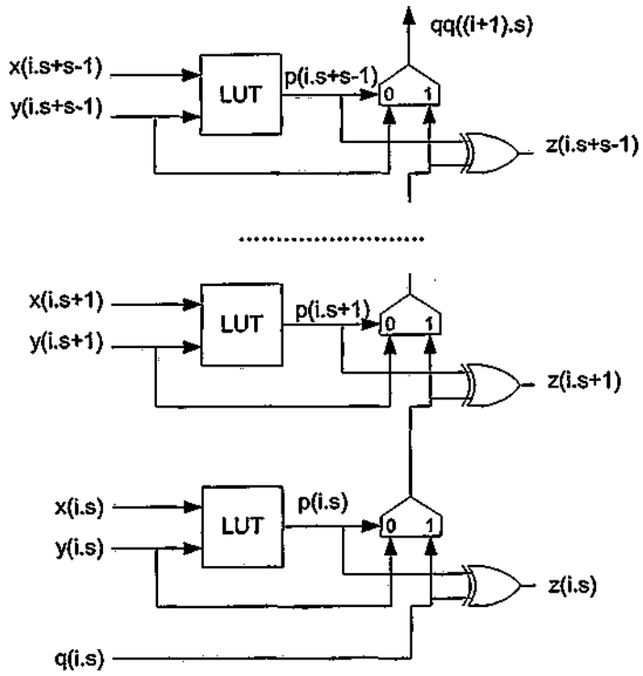


Figura 5.7. Grupo sumador de *s* bits para el carry-skip

Los multiplexores que seleccionan la salida de cada grupo de sumadores pertenece al camino crítico del circuito (Figura 5.11). Por ello, se implementan utilizando multiplexores de acarreo como se puede ver en la Figura 5.8. Obsérvese que para conectar las entradas *P(i)* a los multiplexores internos, se debe obligatoriamente hacerlo a través de tablas de *look-up*. Los tiempos de propagación y área son:

$t_{lut} + (n/s - 2).t_{muxcy}$ y $(n/s - 2)/2$ *slices*

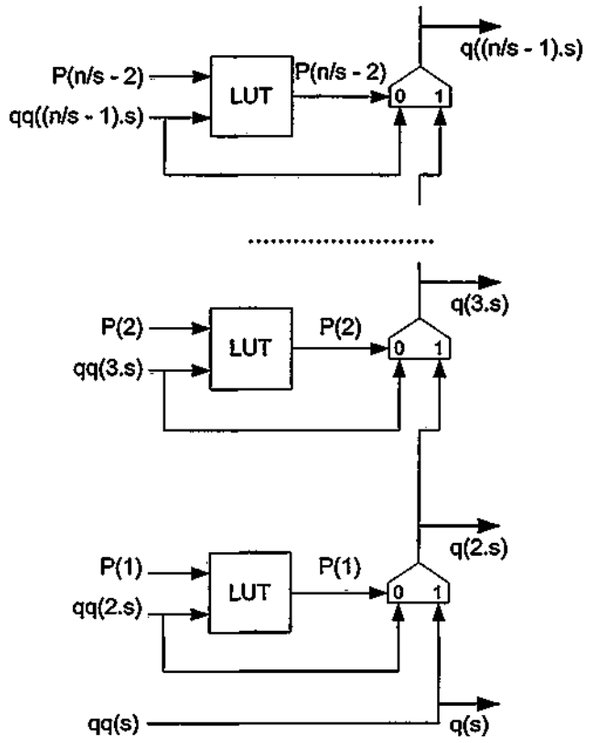


Figura 5.8. Multiplexores de salto de acarreo (*carry-skip multiplexers*)

El circuito que genera el producto $P(i) = p(i.s).p(i.s+1). \ldots p(i.s+s-1)$ se puede observar en la Figura 5.9, cada tabla de *look-up* calcula:

$p(i.s).p(i.s+1) = (x(i.s)\ xor\ y(i.s)).\ (x(i.s+1)\ xor\ y(i.s+1))$,

$p(i.s+2).p(i.s+3) = (x(i.s+2)\ xor\ y(i.s+2)).\ (x(i.s+3)\ xor\ y(i.s+3))$,

etc.,

En tanto que los multiplexores de acarreo implementan la función AND entre ellos. El tiempo de cómputo y costo son:

$t_{lut} + (s/2).t_{muxcy}$ and $s/4$ *slices*.

La estructura completa de un sumador *carry-skip* (con $n/s = 4$) se puede ver en la Figura 5.11. El tiempo de cálculo (parte sombreada del gráfico) es igual a:

$$T_{adder} = t_{lut} + max\{(2.s + n/s - 3).t_{muxcy}, t_{lut} + (1{,}5.s + n/s - 3).t_{muxcy}\} + 2.t_{connection} + t_{xor2}$$

Donde $t_{connection}$ se refiere al tiempo de una conexión de propósito general. El área utiliza es aproximadamente igual a:

$$C_{adder} \cong 0{,}75.n + 0{,}5.n/s$$

En función de la ecuación anterior, el menor retardo teórico se logrará cuando $2.s + n/s$ ó $1{,}5.s + n/s$ sea mínimo. Esto es decir:

$$s \cong (n/2)^{1/2} \text{ and } n/s \cong (2.n)^{1/2}, \text{ or } s \cong (n/1{,}5)^{1/2} \text{ and } n/s \cong (1{,}5.n)^{1/2}.$$

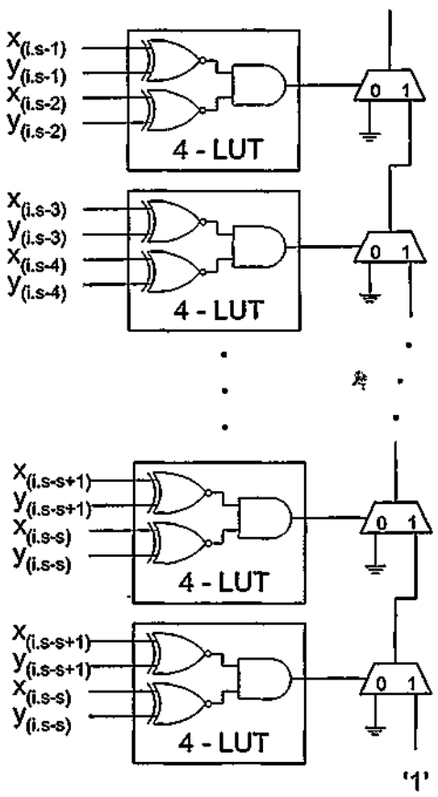


Figura 5.9. Generación de la condición de propagación del acarreo.

5.3.3 Resultados experimentales del sumador *carry-skip*

Se han implementado diferentes sumadores dentro de diferentes modelos de FPGAs de la familia Spartan2 y Virtex, se presentan a continuación por un lado los resultados de área y velocidad y por otro los resultados del consumo.

5.3.3.1 Resultados en área - velocidad

Varios sumadores fueron implementados dentro en una FPGA de la familia Spartan2 (más exactamente una XC2s200e-6pq208, una matriz 28x42-*slices*). La síntesis se llevo a cabo utilizando la herramienta XST (*Xilinx Synthesis Technology*) [Xil02a] y la implementación usando Xilinx ISE (*Integrated System Environment*) [Xil03b]. Para poder utilizar todos los recursos disponibles, el diseño instancia componentes de bajo nivel de la (FPGA LUTs, MUXCY, XORCY, etc). El diseño utiliza además emplazamiento relativo (*relative placement & routing* - RPR) [Xil99] [Xil02b] para mantener bajo control los detalles del rutado.

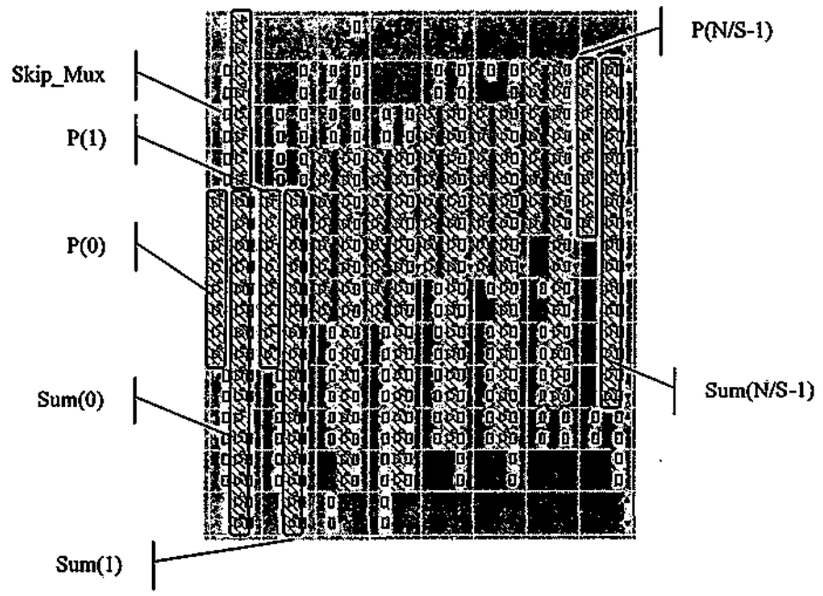


Figura 5.10. Emplazamiento relativo para un Carry-Skip con N=128, S=16.

Cada vez que es posible, el primer sumador de *s*-bits, el conjunto de multiplexores *carry skip* ($n/s - 2$ etapas) y el último sumador de *s*-bits son colocados sobre la misma columna de modo que las conexiones asociadas con las señales $qq(s)$ y $q(3.s)$ de la figura 5.10 no utilicen recursos de rutado de propósito general, sino los propios de la cadena de acarreo. La condición para poder utilizar esto es que $s + (n/s - 2)/2 \leq$ alto en *slices* de la FPGA. En la Figura 5.10 se muestra un ejemplo del emplazado relativo para N=128, S=16.

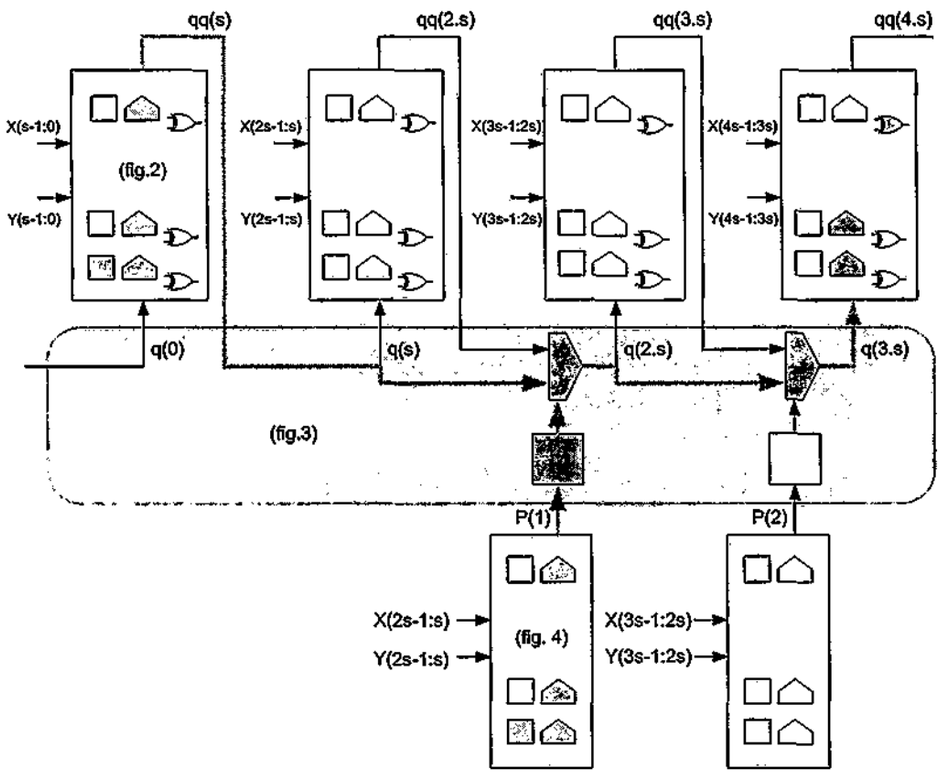


Figura 5.11. Sumador carry-skip ($n = 4.s$)

Todos los circuitos son implementados con la estructura que se sugiere en la Figura 5.12. Los retardos medidos son desde el pulso de reloj externo *clk* hasta las entradas *d* de los registros de salida, de modo que se tienen en cuenta los retardos de las conexiones de propósito general *conn.*1(1') y *conn.*2. Dado que el dispositivo utilizado posee menos patas que las utilizadas en el circuito se le agregan multiplexores a la entrada y salida los que no se grafican en la figura.

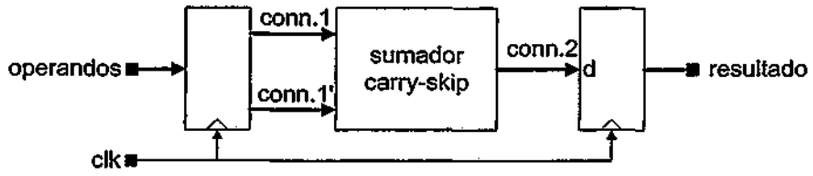


Figura 5.12. Estructura de los circuitos a medir.

Los resultados para el retardo se resumen en la Tabla 5.5. La primera columna ($s = n$) indica el retardo para un sumador tradicional. La última columna muestra el incremento de frecuencia del más rápido respecto del tradicional. La Tabla 5.6 presenta el área en *slices* de las implementaciones, así como la penalización ó aumento de área producto del uso de la técnica.

n	s=n	s=8	s=16	s=32	Aumento frec.
64	14 ns	13 ns	12 ns	-	13 %
96	16 ns	14 ns	13 ns	-	21 %
128	23 ns	14 ns	14 ns	-	63 %
256	38 ns	-	16 ns	17 ns	141 %
512	77 ns	-	20 ns	20 ns	296 %
1024	159 ns	-	28 ns	25 ns	531 %

Tabla 5.5. Resultados de la implementación de circuitos carry skip: Retardos en ns.

	Área en *Slices*				Aumento de área		
n	*s*=*n*	*s*=8	*s*=16	*s*=32	*s*=8	*s*=16	*s*=32
64	32	47	41	-	47 %	28 %	-
96	48	73	66	-	52 %	38 %	-
128	64	99	91	-	55 %	42 %	-
256	128	-	191	179	-	49 %	40 %
512	256	-	391	375	-	53 %	46 %
1024	512	-	791	767	-	54 %	50 %

Tabla 5.6. Resultados de la implementación de circuitos carry skip: Penalidad en área.

Gracias al uso de la circuitería dedicada a la lógica de acarreo para todos los bloques incluidos en el camino crítico, el incremento de frecuencia para sumadores operandos grandes son apreciables: más del 500 % para un sumador de 1024-bits.

5.3.3.2 Resultados en consumo

Dada la cantidad de datos de entrada-salida cuando se utilizan operandos largos. Se optó por utilizar los sumadores dentro de un acumulador para medir el consumo en funcionamiento. El acumulador suma tiene una entrada de 64 bits que se registran $n/64$ veces y compone el operando 1, el operando 2 surge de la recirculación del resultado (Figura 5.13). Para controlar el funcionamiento se sacan como salida los 32 bits más altos y los 32 más bajos del acumulador.

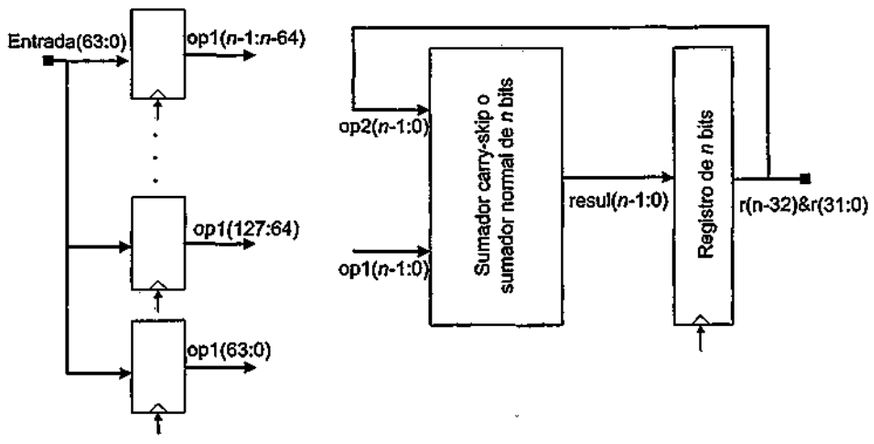


Figura 5.13. Estructura de los acumuladores utilizados para medir el consumo

Se han construido y medido el consumo para acumuladores basados en sumadores normales y sumadores *carry-skip* para operandos de 256, 512 y 1024 bits. Los resultados a nivel simulación con Xpower [Xpo02] dieron valores favorables al *carry skip* pero la posterior medición contradijo las estimaciones iniciales. En la Tabla 5.7 se puede ver el resultado del consumo para diferentes patrones de entrada.

Para la mayoría de los casos el sumador normal da mejores resultados que la implementación *carry skip*, solo en el caso de la suma de -1 (FFFF) y 1 la implementación normal da peores resultados. Se han probado una gran cantidad de valores de entrada para los acumuladores implementados con sumadores de 256 bits siendo siempre la implementación normal superior. A continuación se describen las posibles causas.

Circuito	Consumo según patrones de entrada (mW/MHz)				
	5555	6565	FFFF	0001	8001
Normal_256	6,06	5,75	2,55	2,68	4,52
Skip_256	9,94	10,09	2,08	2,32	7,73
Normal_512	8,82	8,43	3,38	3,34	6,53
Skip_512	17,74	17,69	2,69	3,14	13,89
Normal_1024	13,86	13,33	4,12	5,09	10,13
Skip_1024	31,91	31,99	4,04	4,48	24,21

Tabla 5.7. Consumo sumadores Carry-Skip.

5.3.3.3 Justificación de los resultados en consumo

La hipótesis para suponer un menor consumo del sumador *carry-skip* se basa en dos hechos. En primer lugar, la arquitectura del sumador *carry-skip* ha de tener una menor actividad en la cadena de acarreos, ya que esta se divide en tramos de n/s bits. Y por otra parte, la afirmación empírica en el diseño de FPGAs, que dice que los circuitos más rápidos consumen menos.

El error en el razonamiento parte del siguiente hecho: El cálculo del camino crítico en el sumador *ripple-carry* incluye toda la cadena de acarreo. En el peor escenario el acarreo debe propagarse desde la primera a la última etapa. En ese caso, es cierto que el acarreo producirá gran actividad en la salida. No obstante, probabilísticamente esto

ocurre en muy pocos casos, y la mayoría de las veces el acarreo se propaga por unas pocas etapas.

En general si se considera a un sumador compuesto por las funciones G-P (*generate - propagate*) donde:

g(i) = 1 sii x(i) + y(i) > B-1, en base B = 2 será: g(i) = x(i) . y(i)

p(i) = 1 sii x(i) + y(i) = B-1; en base B = 2 será: p(i) = x(i) ⊕ y(i)

Luego se pude calcular el acarreo siguiente de esta manera:

if p(i) = 1 **then** q(i+1) := q(i); **else** q(i+1) := g(i); **end if**;

De lo antedicho se deduce que importa el acarreo anterior si y solo si *p(i)* =1, es decir si x(i) e y(i) son distintos. Por tanto la probabilidad de que se propague el acarreo es de ½ por etapa del sumador. Luego la probabilidad α de que el acarreó se propague durante *s* etapas será:

$$\alpha = (1/2)^s$$

Es decir que el acarreó se propague por 5 bits tiene una probabilidad de 0,031; que se propague por 10 bits 0,00097; y que se propague por 100 alrededor de $7{,}8 \times 10^{-31}$. Por ello, en realidad los *glitches* que se producen como resultado de la propagación del acarreo no son desde el punto de vista del consumo relevantes.

5.3.4 Conclusiones sobre algoritmos sumadores

La técnica de sumador *carry skip* resulta muy interesante para ser aplicada en FPGAs para operandos largos. Aunque las técnicas de *carry-look-ahead* (CLA) parecen ser más veloces que las de *carry skip* o *ripple carry*, en el marco tecnológico específico esto no es verdad. Una implementación cuidadosa usando la lógica de acarreo convenientemente logra un incremento de frecuencia, para sumadores con operandos de 1024 bits, que superan el 500%, con un incremento del área menor del 50% respecto a la implementación por defecto en FPGA (*ripple carry*).

En adición *carry-skip* el mecanismo de cálculo del acarreo genera menos *glitches* producto de una menor propagación del acarreo y consecuentemente se preveía menos consumo. Basándose en esta idea y en el hecho estudiado en el capítulo 3, que

los circuitos más veloces consumen menos, se construyeron circuitos cuyas mediciones negaron la suposición inicial.

El error en el razonamiento parte del siguiente hecho: El cálculo del peor tiempo (camino crítico) incluye la cadena de acarreo y el peor caso donde el acarreo debe propagarse desde la primera a la última etapa. En ese caso es cierto que el acarreo producirá gran actividad en la salida. Pero, probabilísticamente esto ocurre en muy pocos casos y la mayoría de las veces el acarreo se propaga por unas pocas etapas, por tanto los *glitches* por propagación del acarreo no son tan importantes como se creía.

5.4 Algoritmos y arquitecturas para la división entera

Naturales, enteros y fraccionarios pueden ser multiplicados exactamente mientras existan suficiente cantidad de dígitos para el resultado. La operación de división no comparte esta característica, de hecho la división generalmente no provee un resultado de longitud fija. La precisión debe ser definida de antemano, seleccionando el máximo tamaño del resultado. La cantidad de ciclos del algoritmo dependerá pues de la precisión pretendida y no del largo de los operandos.

El estudio se divide en forma temprana en algoritmos para división de enteros y algoritmos de división con números fraccionarios normalizados. El primer grupo de algoritmos analizados corresponde con las típicas operaciones de división de números naturales o enteros (esta sección), en tanto la segunda como parte esencial en sistemas de punto flotante (sección 5.5). La nomenclatura y descripción de los algoritmos corresponde a [Par00][Des02].

5.4.1 Algoritmos de división para números naturales y enteros

Sean X e Y dos naturales en base *B*, de *n* y *m* bits respectivamente:

$$X = x_{n-1}.B^{n-1} + x_{n-2}.B^{n-2} + ... + x^0.B^0,$$

$$Y = y_{m-1}.B^{m-1} + y_{n-2}.B^{n-2} + ... + y_0.B^0,$$

$$x_i, y_i \in \{0, 1, ..., B-1\},$$

Con la condición que $Y > 0$. Se define Q y R como el cociente y resto de la división X por Y, con una precisión de *p* digitos:

$$B^p.X = Q.Y + R$$

Donde Q y R son números naturales y $R < Y$ (resto menor que la división). De otra forma se puede escribir:

$$X/Y = Q.B^{-p} + (R.B^{-p}/Y) \qquad \text{(ec. 5.1)}$$

Con

$$R.B^{-p}/Y < B^{-p}. \qquad \text{(ec. 5.2)}$$

El algoritmo básico se aplica a operandos X e Y tales que X < Y. Un paso previo consiste en sustituir Y por $Y.B^n$ tal que $Y.B^n \geq 1.B^n > X$, con lo que queda:

$$B^p.X = (Q.B^n).Y + R$$

La implementación de este ajuste es trivial, ya que consiste en agregar simplemente más dígitos ceros en la operación y realizar un ajuste similar con el resultado.

Lema 5.4.1

El siguiente lema justifica el algoritmo básico de la división así como los algoritmos de división por recurrencia de dígitos (*Digit recurrence division algorithms*). En general dados dos números naturales a y b tal que $a < b$, luego existen dos únicos números q y r tal que satisfacen $B.a = q.b + r$, con $q \in \{0, 1, \ldots, B\text{-}1\}$ y además $r < b$.

La aplicación recursiva del lema anterior, es decir

$$B.r(0) = q(1).Y + r(1),$$
$$B.r(1) = q(2).Y + r(2),$$
$$\ldots$$
$$B.r(p\text{-}1) = q(p).Y + r(p), \qquad \text{(Ec 5.3)}$$

Con r(0) = X, genera la siguiente relación

$$X.B^p = (q(1).B^{p-1} + q(2).B^{p-2} + \ldots + q(p).B^0).Y + r(p), \qquad \text{(Ec 5.4)}$$

Con lo que se tiene

$$Q = q(1).B^{p-1} + q(2).B^{p-2} + \ldots + q(p).B^0 \text{ ; y } R = r(p).$$

Si se asume un procedimiento *division_step* definido como

```
procedure division_step (a, b: in natural; q, r: out natural);
```

Que calcula q y r tal que $B.a = q.b + r$, con $q \in \{0, 1, \ldots, B\text{-}1\}$; y también $r < b$. Luego el siguiente algoritmo básico de división es consecuencia directa de las ecuaciones (5.3) y (5.4).

5.4.1.1 Algoritmo con restauración (*restoring division algorithm*)

En función de lo antes descrito, se puede definir el algoritmo de división con restauración por medio del algoritmo 5.3.1.

Algoritmo 5.4.1 - de división con restauración

```
r(0) := X;
for i in 1 .. p loop
        division_step (r(i-1), Y, q(i), r(i));
end loop;
```

Esto generará la representación en base-B de Q formada por q(1) q(2) ... q(p) y el resto R = r(p). La Figura 5.14.a muestra un diagrama de bloques de este algoritmo. Para la base-B = 2 el procedimiento *division_step* resulta muy simple, se puede ver un esquema en la Figura 5.14.b y se resume a continuación:

Algorithm 5.4.2

```
z := 2*a - b;
if z < 0 then q := 0; r := 2*a; else q := 1; r := z; end if;
```

Para una base B mayor que dos el paso de división es más complejo, por ejemplo:

Algorithm 5.4.3

```
if B*a < b then q := 0; r := B*a;
elsif B*a < 2*b then q := 1; r := B*a – b;
elsif B*a < 3*b then q := 2; r := B*a – (2*b);
...
elsif B*a < (B-1)*b then q := B-2; r := B*a – ((B-2)*b);
else q := B-1; r := B*a – ((B-1)*b);
```

A cada paso del algoritmo 5.3.1, el nuevo resto r(i) es igual a B.r(i-1) - q(i).Y. Si fuese q(i) = 0 entonces r(i) = B.r(i-1) y el resto precedente r(i-1) se dice que es restaurado (en realidad restaurado y deplazado). Por esta razón al algoritmo básico de división se lo conoce como algoritmo de restauración (*restoring division algorithm.*)

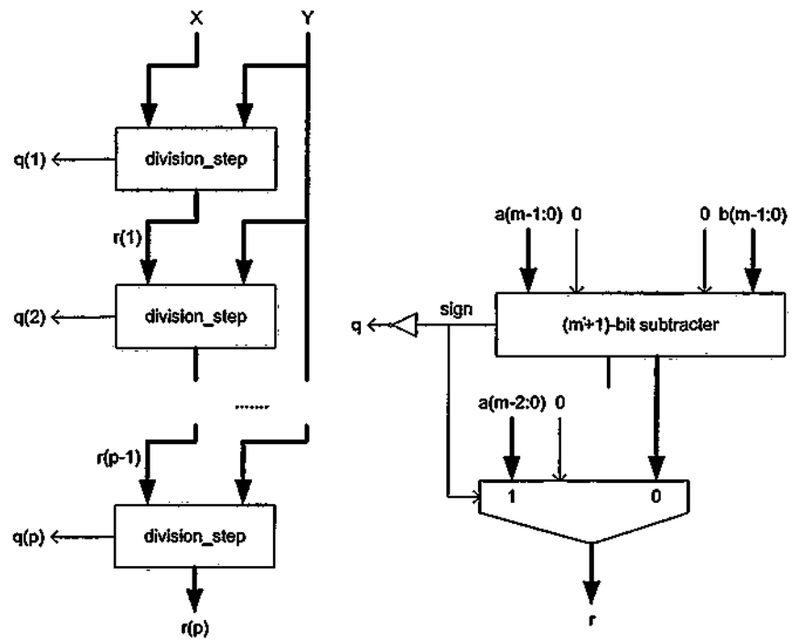


Figura 5.14. Divisor con restauración. a. Estructura general. b. Celda básica

5.4.1.2 Algoritmo sin restauración (*non-restoring division algorithm*)

Sea X un entero de n bits en complemento a la base con base-B dos (*2's complement*) y un número positivo de m bits. Se definen el cociente Q y resto R de la división entre X e Y con una presición de p bits a:

$$2^p X = Q.Y + R \qquad \text{(Ec 5.6)}$$

Donde, tanto Q como R son enteros tales que signo(R) = signo(X) y abs(R) < Y. En otras palabras:

$$X/Y = Q.2^{-p} + (R.2^{-p}/Y) \qquad \text{(Ec 5.7)}$$

Con

$$\text{signo}(R.2^{-p}/Y) = \text{signo}(X) \text{ y } \text{abs}(R.2^{-p}/Y) < 2^{-p}. \qquad \text{(Ec 5.8)}$$

El algoritmo básico se aplica a operandos X e Y con tal que $-Y \leq X < Y$. Como sucede en el algoritmo de división con restauración, un paso previo puede consistir en

reemplazar Y por $Y.2^{n-1}$ tal que $Y.2^{n-1} \geq 1.2^{n-1} > X$, $-Y.2^{n-1} \leq -1.2^{n-1} \leq X$; por tanto: $2^p.X = (Q.2^{n-1}).Y + R$.

El siguiente lema contituye la justificación del algoritmo de división sin restauración (*non-restoring división algorithm*).

Lema 5.4.3

Dados un intero a y un numero positivo b, tal que $-b \leq a < b$, luego $2.a$ puede ser descompuesto de la forma $2.a = q.b + r$ donde $q \in \{-1, 1\}$ y además $-b \leq r < b$.

Una aplicación recursiva de este lema nos da que:

$2.r(0) = q(0).Y + r(1)$,
$2.r(1) = q(1).Y + r(2)$,
...
$2.r(p-1) = q(p-1).Y + r(p)$, (Ec 5.9)

Si se comienza con $r(0) = X$, se genera la siguiente relación:

$X.2^p = (q(0).2^{p-1} + q(1).2^{p-2} + ... + q(p-1).2^0).Y + r(p)$. (Ec 5.10)

Se define $q'(i) = (q(i)+1)/2 \in \{0, 1\}$ y se subtituye $q(i) \in \{-1, 1\}$ por $2.q'(i)-1$:

$X.2^p = ((q'(0)-1).2^p + q'(1).2^{p-1} + ... + q'(p-1).2^1 + 1.2^0).Y + r(p)$. (Ec 5.11)

Por consiguiente:

$X.2^p = Q'.Y + r(p)$ (Ec 5.12)

Donde

$Q' = (q'(0)-1).2^p + q'(1).2^{p-1} + ... + q'(p-1).2^1 + 1.2^0$; $-Y \leq r(p) < Y$. (Ec 5..13)

Si coinciden los signos del resto y el dividendo; signo(r(p)) = signo(X) entonces:

$Q = Q'$ and $R = r(p)$.

Si los signos son diferentes se debe llevar a cabo una corrección final:

if $r(p) < 0$ and $X \geq 0$ then $Q = Q' - 1$ and $R = r(p) + Y$;
if $r(p) \geq 0$ and $X < 0$ then $Q = Q' - 1$ and $R = r(p) - Y$;

Si se supone además un procedimiento *division_step*, definido como

procedure division_step (a: **in** integer; b: **in** natural; q: **out** binary; r: **out** integer);

Este procedimiento computará q and r tal que $2.a = (2.q - 1).b + r$, con $q \in \{0, 1\}$ y $-b \leq r < b$.

De acuerdo con (5.9), (5.12) y (5.13) el siguiente algoritmo computa la representación en complemento a dos $q(0)\ q(1)\ q(2) \ldots q(p)$ de Q' y el resto $R' = r(p)$. Un esquema del algoritmo se puede observar en la Figura 5.15.a.

Algoritmo 5.4.7 – Algoritmo sin restauración

```
r(0) := X;
for i in 0 .. p-1 loop
    division_step (r(i), Y, q(i), r(i+1));
end loop;
q(p) := 1; q(0) := 1 - q(0);
```

En caso de ser necesario se debe anadir el siguiente paso de corrección:

```
if X < 0 and r(p) >= 0 then R := r(p) - Y; Q := Q' + 1;
elsif X >= 0 and r(p) < 0 then R := r(p) + Y; Q := Q' - 1;
else R := r(p); Q := Q'; end if;
```

El proceso de división *division_step* es muy simple, el esquema correspondiente en la Figura 5.15.b. En tanto el esquema del paso de corrección se puede ver en Figura 5.16.

Algoritmo 5.4.8

```
if a < 0 then q := 0; r := 2*a + b; else q := 1; r := 2*a - b; end if;
```

Obsérvese que a cada paso del algoritmo 5.3.7, el nuevo resto $r(i+1)$ es igual o bien a $2.r(i) - Y$ o $2.r(i) + Y$, de este modo nunca es restaurado el valor anterior.

5.4.1.3 Otros algoritmos para números enteros

Otros algoritmos de división suponen dividendo y divisor en cierto rango, el costo en área, velocidad y consumo asociado a la normalización los hacen poco atractivos. Para dividir enteros, tambien se pueden utilizar algoritmos en bases mayores que dos,

aunque esta opción no posee ninguna ventaja desde el punto de vista del consumo (ni en área y velocidad). En la siguiente sección, donde se estudia la división para numeros fraccionarios, se analizan algoritmos tipo SRT para bases mayores de dos, los cuales sí poseen interes desde el punto de vista de la velocidad y el consumo.

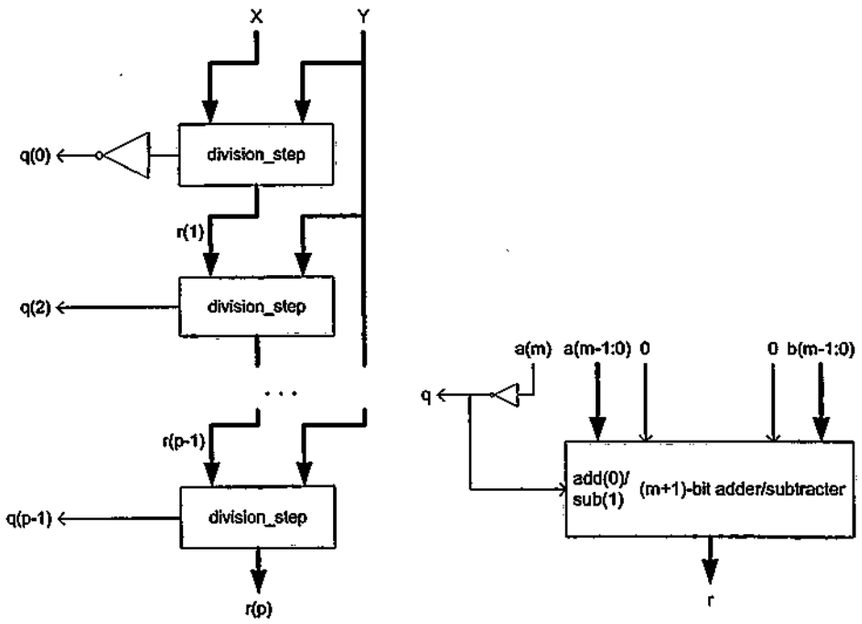


Figura 5.15. Divisor sin restauración. Estructura general. Celda básica del algortimo.

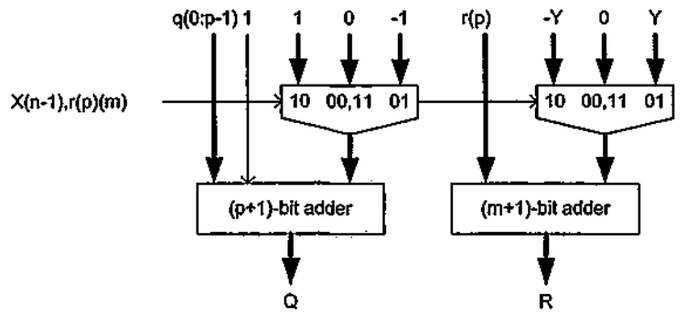


Figura 5.16. Celda de corrección del cociente y resto en el algoritmo sin restauración.

5.4.2 Implementaciones en FPGA para números naturales y enteros

Para poder sacar provecho de las características de los dispositivos programables es imprescindible utilizar los recursos específicos, fundamentalmente la lógica de acarreo (*carry-logic*) en la construcción de sumadores y restadores. Los diseños presentados en esta sección son optimizados para las familias Virtex/Spartan 2 y Virtex II/Spartan 3 en sus diferentes versiones. En estos dispositivos, las primitivas de cálculo:

suma: $r : a+b$
resta: $r = a + (2n - b)$
suma / resta: $r = a + (1-x).b + x.(2n - b)$
suma condicional: $r = a + x.b$
resta condicional: $r = a + x.(2n - b)$
selección: $r = (1-x).a + x.b$

Donde a y b son números de n bits y x una señal de un solo bit, se sintetizan con $n/2$ *slices*. Si las operaciones de suma y/o resta deben retornar una señal de acarreo o arrastre final (*carry* ó *borrow*), éstas necesitarán realizar la operación con un bit más, es decir, se han de sintetizar en $n/2 + 1$ *slices*. Esta última consideración no tiene sentido para la selección (multiplexor 2-1 de n bits) que se sintetiza con $n/2$ *slices*. Obsérvese que el coste de un multiplexor 2-1 es prácticamente el mismo que el de un sumador / restador ($n/2$ vs. $n/2 + 1$), es decir, el de n tablas de *look-up*. La conclusión sería bastante diferente en el caso de la versión *standard cell* de las mismas primitivas.

5.4.2.1 Algoritmo con restauración en base dos

Como se mencionó anteriormente, el algoritmo básico se aplica a operandos enteros X e Y tales que $X < Y$. Para el caso de enteros de n y m bits se procede a sustituir Y por $Y.B^m$ (es decir se agregan m ceros al final del operando Y). Dado que, los últimos m bits del operando Y son ceros, el algoritmo 5.2 de la celda elemental puede ser simplificado en cuanto al tamaño de bits a operar. Obsérvese que la multiplicación por dos (desplazamiento a izquierda) del operando X, como el desplazamiento y la resta de Y pueden ser realizadas con m bits.

La implementación de la Figura 5.17.a consta de n instancias de la celda *nr_division_step* (Figura 5.17.b). La celda de calculo está compuesta por un restador de (m+1)-bits y un multiplexor 2 a 1 de m-bits, con lo que necesitará m+1 *slices*. El retardo será Tlut + m.Tmux, para el restador, más Tconex + TLut producto del multiplexor y su

conexión, con lo que el tiempo total será: $2.T_{lut} + m.T_{mux} + T_{conex}$. Luego para el divisor entero de n sobre m bits

$$C(n,m) = n.(C_{división\_step}(m)) = n.(m+1) = n.m + n \text{ slices}$$

$$T(n, m) = n.(T_{división\_step}(m) + T_{conex}) = 2.n.Tlut + m.n.T_{mux} + 2.n.T_{conex}$$

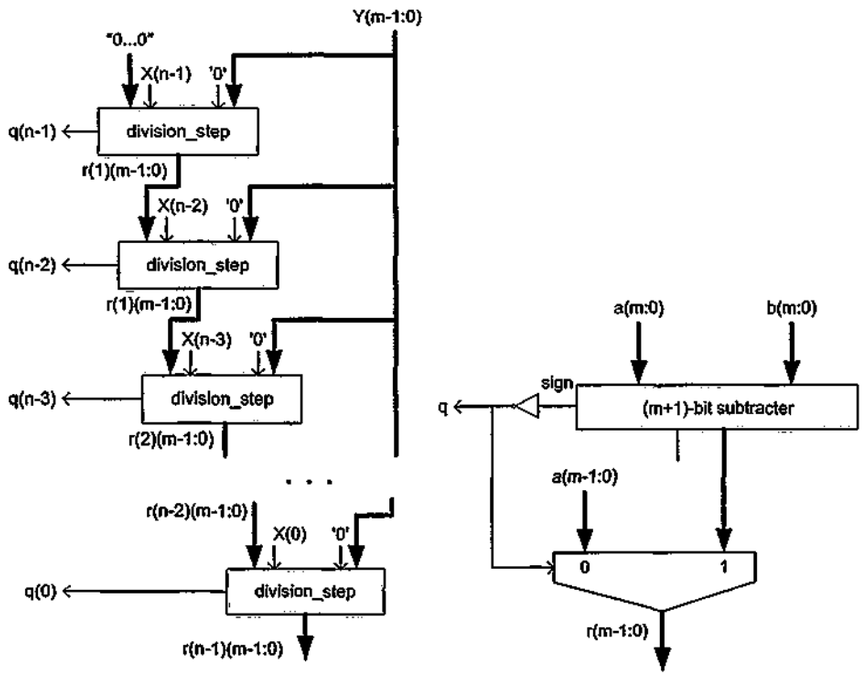


Figura 5.17. Circuito de división de naturales binarios con restauración. a. Estructura general. b. Celda elemental de cálculo.

5.4.2.2 Algoritmo sin restauración en base dos

Al igual que en el caso del algoritmo con restauración, X debe ser menor que Y, más aun $-Y \leq X < Y$. Se opta por la misma solución, es decir se sustituye Y por $Y.B^m$ (es decir se agregan m ceros al final del operando Y). La consideración aplicada al algoritmo 5.2 es extensible al algoritmo 5.8, es decir la operación *division_step* (de ahora en adelante *adder_substracter*) se puede realizar con m bits.

Si en el algoritmo de división se considera a X como natural, los bits indicados con s en la primera celda de cálculo en la Figura 5.18, han de ser ceros. En caso de X entero, los bits s serán reemplazados por $X(n-1)$ (se extiende el signo). La celda *correction_cell* descrita en la Figura 5.16 puede ser simplificada en el caso de X natural, ya que, el ajuste del resto se limitará a una suma condicional y la del cociente a la resta de uno. Más aun se ha de ajustar (sumar Y al resto parcial y restar uno al cociente) si el resto parcial r(n) < 0, con ello r(n)(m) = 1, solo se deberá sumar Y. La resta del cociente no se realiza realmente.

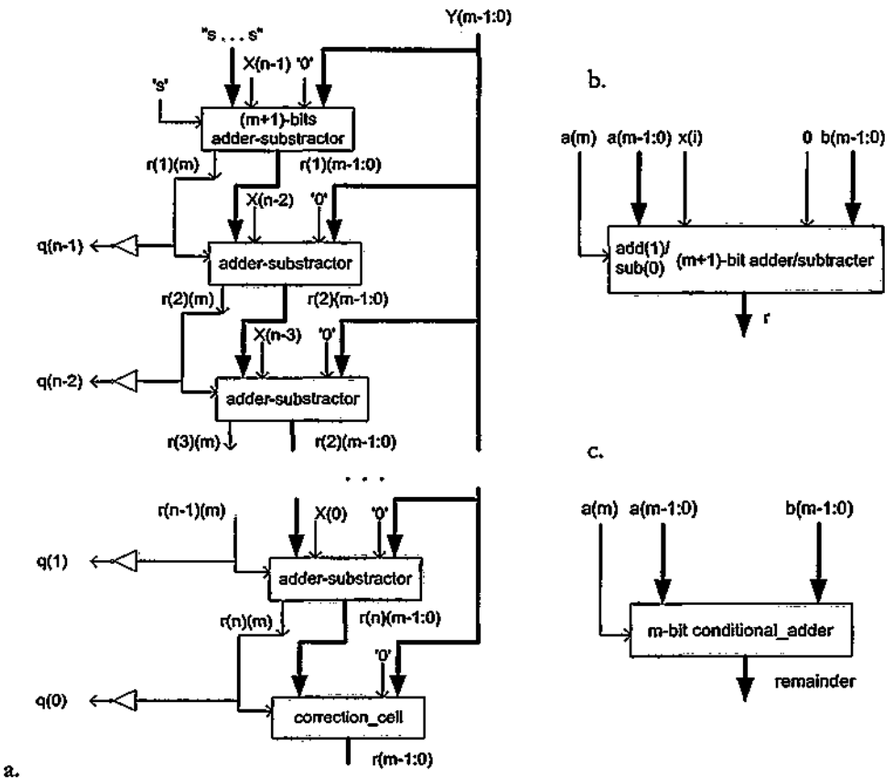


Figura 5.18. Circuito de división de enteros sin restauración. a. Estructura general. b. Celda elemental de cálculo. c. Celda de corrección para naturales.

La Figura 5.18.a se implementa con *n* instancias de la celda *adder_substracter*, más un sumador final. La celda sumador o restador y la celda final (sumador condicional) de *m* bits necesitan por igual *m/2*+1 *slices*, con un retardo de Tlut + *m*.Tmux, Luego para el divisor sin restauración de *n* sobre *m* bits tendremos:

$$C(n,m)=n.(C_{adder\_sustractor}(m))+C_{adder}(m) = (n+1).(m/2+1) = n.m/2+m/2+n+1 \text{ slices}$$

$$T(n, p) = n.(T_{adder\_substracter}(m) + T_{conex}) + T_{adder}(m) + T_{conex} =$$

$$= (n+1).T_{lut} + m.(n+1).T_{mux} + (n+1).T_{conex}$$

5.4.3 Implementaciones para divisores de números enteros

Los circuitos fueron implementados en una Virtex XCV800hq240-6, utilizando el entorno de desarrollo ISE 6.1 [Xil03d] y XST (*Xilinx Synthesis Technologies*) [Xil03e] para la síntesis. Los resultados en área y velocidad se pueden ver en la Tabla 5.8. En la tabla se puede observar que la división con restauración ocupa al menos el doble de área y necesita alrededor del 50% más de tiempo que el algoritmo sin restauración. Los circuitos "*div_X_Y*" se refiere a dividendo de *X* bits, y divisor de *Y* bits, con lo que el resultado de la división será de X bits y el resto de Y bits

Circuitos	División con restauración			División sin restauración		
	Área		Retardo	Área		Retardo
	Slices	4-LUTs	(ns)	Slices	4-LUTs	(ns)
div_32_32	1589	2048	291,7	641	1119	198,3
div_32_24	1201	1536	253,6	493	847	171,8
div_32_16	812	1024	218,4	345	575	148,1
div_32_8	424	512	177,5	197	303	122,4
div_16_16	396	512	110,5	177	303	80,0
div_16_8	207	256	91,1	101	159	66,8
div_8_8	103	128	45,8	57	87	36,6

Tabla 5.8. Área y retardo para la división con y sin restauración.

5.4.3.1 Resultados del consumo para divisores enteros

En función de los resultados en área y velocidad se puede pensar que el consumo debería favorecer claramente a la división sin restauración. Sin embargo los resultados contradicen ésta suposición.

El consumo fue separado en consumo estático, dinámico (debido a la ruta de datos y sincronización) y el consumo debido a las patas (*off-chip*). La medición de los circuitos fue realizada utilizando dos secuencias diferentes: a) Valores aleatorios (*aleatTog*); b) una secuencia con una alta probabilidad de transición en los nodos (*maxTog*). Los vectores de prueba fueron ingresados a los circuitos con un generador de patrones [Tek04b] y las salidas conectadas a un analizador lógico [Tek04a] como se describe en el Apéndice B.

La Tabla 5.9 muestra los resultados del consumo dinámico medio expresados en mW/MHz para los circuitos presentados anteriormente. Adicionalmente se muestran los resultados para la estimación con XPOWER [Xpo04] (*estim*) con los vectores de prueba aleatorios (*aleatTog*). Detalles del uso y aplicaciones de la herramienta Xpower se describen en el apéndice F.

Circuito	División con restauración			División sin restauración		
	Estim	AleatTog	MaxTog	Estim	AleatTog	MaxTog
div_32_32	48	50	47	332	414	454
div_32_24	54	141	153	254	354	432
div_32_16	69	265	151	176	304	314
div_32_8	106	234	168	94	236	259
div_16_16	21	23	19	57	95	99
div_16_8	17	48	45	34	75	86
div_8_8	14	15	18	18	27	28

Tabla 5.9. Consumo dinámico para los divisores de números enteros expresados en mW/MHz.

La Figura 5.19 muestra el consumo en mW/MHz para los patrones aleatorios. Se observa la disminución del consumo respecto del tamaño de los datos en el divisor sin restauración, aunque se ve un claro crecimiento del consumo promedio en la división con restauración a medida que decrece el tamaño del dividendo.

Esta figura presenta dos hechos, los que a priori, son poco intuitivos. El primero es el mayor consumo que presenta el divisor sin restauración a pesar de ser más veloz y ocupar menos de la mitad que el algoritmo de división con restauración. En segundo lugar el aumento del consumo medio que sufre el divisor *restoring* a medida que se disminuye el tamaño del divisor. Ambos hechos se explican por la actividad espuria

(*glitches*) generados por los algoritmos y por los ajustes necesarios para poder dividir números enteros.

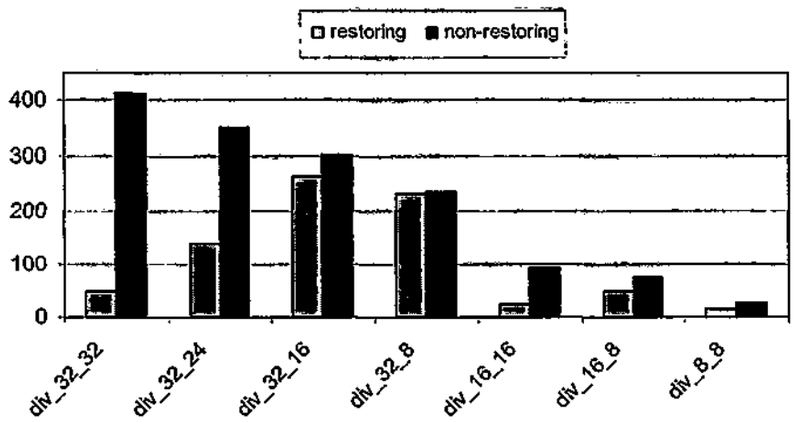


Figura 5.19. Consumo para los divisores con y sin restauración para la secuencia aleatoria (aleatTog) en mW/MHz,

5.4.3.2 Resultados en la actividad interna en los divisores enteros

Como se mencionó en los apartados anteriores (5.4.1 y 5.4.1.2) los algoritmos básicos de división se aplican a operandos X e Y tales que X < Y. El paso previo consiste en sustituir Y por $Y.B^n$ tal que $Y.B^n \geq 1.B^n > X$, con lo que el cociente debe ser multiplicado por B^n. Esto se realiza agregando *m* ceros delante del operando X y suponiendo desplazado el operando Y en *n* posiciones a izquierda (Figura 5.17 y Figura 5.18).

En la Figura 5.20 se muestra la actividad que se genera al dividir cuatro valores en un divisor con restauración de 8 por 8 bits, en tanto la Figura 5.21 hace lo propio con el algoritmo sin restauración. Las figuras muestran los restos intermedios, así como los valores del cociente.

Los ejemplos de las figuras muestran como, en el algoritmo sin restauración, el resto pasa de positivo a negativo prácticamente en todos los restos intermedios, en tanto el algoritmo con restauración al tener los restos siempre positivos genera menos actividad.

En el algoritmo *non-restoring*, como el operando Y ha sido desplazado *n* posiciones a izquierda el resto en las primeras etapas pasará de un número pequeño positivo a un numero pequeño negativo generando gran actividad al estar representado en complemento a dos (ver el ejemplo de la sección 2.6.4).

Este efecto se potencia cuando el divisor tiene muchos bits respecto del dividendo (más de $n/2$). Para los casos más típicos donde el divisor tiene exactamente $n/2$ bits esta diferencia en el consumo se reduce aunque sigue favoreciendo al algoritmo con restauración.

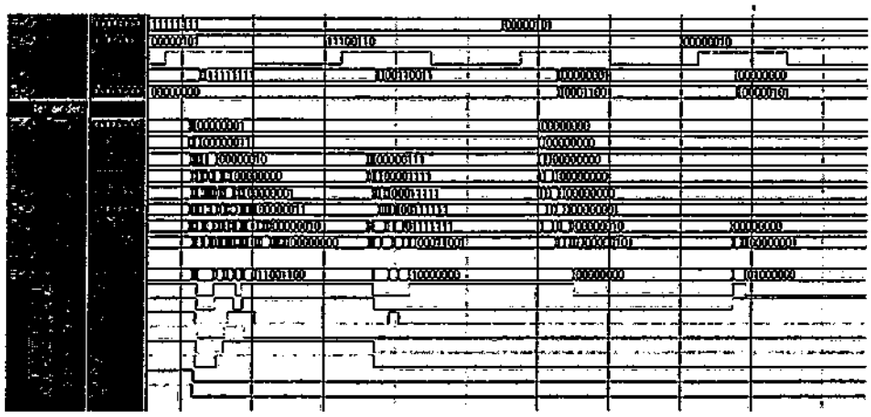


Figura 5.20. Actividad en un divisor con restauración de 8 por 8 bits

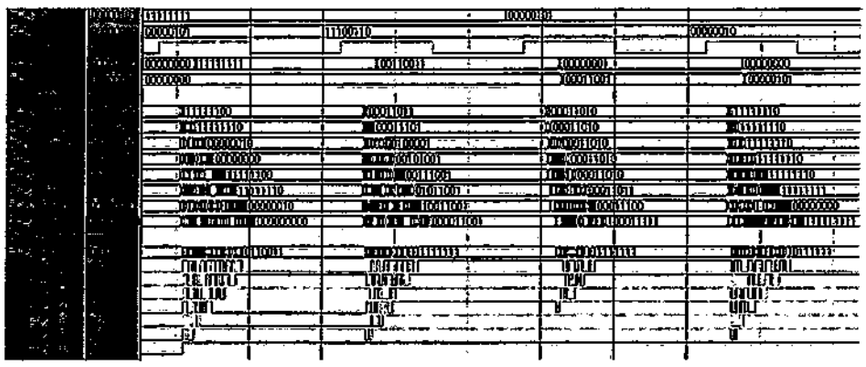


Figura 5.21. Actividad en un divisor sin restauración de 8 por 8 bits.

Como conclusión, el algoritmo *non-restoring* produce en los pasos intermedios del algoritmo mucha más actividad que su contraparte con restauración. Se puede observar en la Tabla 5.10 los resultados de la actividad a nivel simulación para una secuencia aleatoria de entrada.

Por último cabe destacar que estos resultados en el consumo son completamente diferentes cuando se utilizan operando normalizados (por tanto no hay que desplazar el dividendo) como es el caso de la división de números fraccionarios que se estudia en la siguiente sección.

Origen de la actividad	División con restauración	División sin restauración
Reloj de entrada	10	10
Actividad media de cada entrada	2,5	2,5
Máxima actividad en señal interna	13,0	102,5
Máxima actividad en lógica interna	16,7	65,5
Actividad media de las 10 señales que más consumen	8,1	52,6
Activ. media de los 10 elem. de lógica que más consumen	12,7	46,3

Tabla 5.10. Actividad de señales y lógica para los algoritmos divisores de enteros de 8 bits. Los valores expresan frecuencias de las señales en MHz.

5.4.4 Conclusiones para la división de números enteros

Los resultados en área y velocidad favorecen claramente a la división sin restauración (*non-restoring*). El algoritmo con restauración (*restoring*) ocupa el doble de área y necesita alrededor del 50% más de tiempo que el algoritmo sin restauración.

Sin embargo los resultados en el consumo son claramente favorables al algoritmo de división con restauración. El motivo de la diferencia de consumo son las transiciones internas generadas en cada algoritmo. Para poder dividir cualquier par de enteros el dividendo debe ser desplazado n posiciones a izquierda, lo que genera en el algoritmo *non-restoring*, de un número pequeño positivo a un número pequeño negativo generando gran actividad ya que el resto es representado en complemento a dos. Este efecto se refuerza cuando el tamaño del divisor es mayor a $n/2$ bits, siendo n el tamaño del dividendo.

Para la división de números de 32 bits el algoritmo sin restauración consume más de ocho veces que el algoritmo con restauración, en tanto el consumo medio para la división de 32 bits por 16 bits en *non-restoring* es solo alrededor del 20 % superior al algoritmo de división con restauración.

La división de enteros es un excelente ejemplo que demuestra que a nivel algorítmico ni el área, ni la velocidad son buenas métricas para evaluar un algoritmo desde el punto de vista del consumo. Por último cabe destacar que estos resultados en el consumo son completamente diferentes cuando se utilizan operando normalizados.

5.5 Algoritmos y arquitecturas para la división de números fraccionarios

La división de números fraccionarios y normalizados tiene fundamental interés en la división de números en coma (o punto) flotante, ya que es la operación fundamental a realizar en la división de las mantisas. Esta sección se especializa en los divisores en el rango [1,2).

Para dividir números normalizados se pueden utilizar los algoritmos con restauración y sin restauración descritos previamente, aunque el algoritmo de recurrencia de dígitos más utilizado en los CPU's modernos es SRT, que debe su nombre a Sweeney [Swe61], Robertson [Rob58] y Tocher [Toc58], quienes desarrollaron los trabajos a aproximadamente el mismo tiempo.

Una descripción más amplia de los algoritmos de recurrencia de dígitos y en especial de los algoritmos SRT se puede encontrar en [Obe97][Erc94][Par00][Erc04][Sod96].

En esta sección se propone una implementación eficiente del algoritmo SRT en base 2 que posee interesantes características desde el punto de vista del consumo. Adicionalmente se estudian implementaciones en diferentes bases. Por último se cuantifica la influencia de la segmentación (*pipeline*) y de las implementaciones secuenciales.

5.5.1 Algoritmos de división fraccionaria

Dados dos números reales no negativos, el dividendo X y el divisor D ($D \neq 0$), el cociente q y el resto r son números reales no negativos definidos por la siguiente expresión: $X = q.D + r$ con $r < D.ulp$, donde *ulp* son las unidades en las posiciones menos significativas (*unit in the least significant position*). Si X y D son las mantisas de dos números en punto flotante según el estándar IEEE-754, los números pertenecerán al rango [1,2), y q estará en el rango [0.5, 1). Este resultado puede ser normalizado desplazando el cociente un bit a la derecha y ajustando el exponente oportunamente.

La división no provee resultados de una longitud finita. La precisión debe ser definida de antemano, definiendo el máximo tamaño para el resultado (p). La cantidad de ciclos

del algoritmo por tanto no depende del tamaño de los operandos (n), sino de la precisión requerida en el resultado (p).

5.5.2 Algoritmos con y sin-restauración (*restoring and non-restoring algorithm*)

Los circuitos con y sin restauración descritos anteriormente (5.3.2) son fácilmente adaptables a operandos fraccionarios [Par00][Erc04]. Las implementaciones en FPGAs son fáciles de realizar y los resultados en área y velocidad favorecen siempre al algoritmo sin restauración (*non-restoring*). La

Figura 5.22 resume los algoritmos con y sin restauración para operandos fraccionarios. En el último algoritmo se debe agregar un paso de corrección, en caso que el resto sea negativo.

Restoring division algorithm	*Non-restoring division algorithm*
r(0) := X; *for* i *in* 1 .. p *loop* rest_step(r(i-1), D,q(i), r(i)); *end loop;* rest_step (a, b, q, r) z := 2*a - b; *if* z < 0 *then* q := 0; r := 2*a; *else* q := 1; r := z; *end if;*	r(0) := X; *for* i *in* 0 .. p-1 *loop* nonr_step(r(i), D,q(i), r(i+1)); *end loop;* q(p):=1; q(0):=1-q(0); nonr_step (a,b,q,r) *if* a < 0 *then* q := 0; r := 2*a+b; *else* q := 1; r := 2*a-b; *end if;*

Figura 5.22. Algoritmos de división con y sin restauración para números fraccionarios.

5.5.3 Algoritmos SRT

Como todos los algoritmos de recurrencia de dígitos, en cada paso de la iteración se obtienen un número fijo de bits. Los algoritmos SRT utilizan una cantidad limitada de comparaciones para acelerar el cálculo.

Un diagrama de bloques general de una iteración de un algoritmo SRT en base r ($r = 2^k$, K entero) se puede ver en la Figura 5.23. Un divisor necesitará $t = \lceil n/k \rceil$ iteraciones para la división de operandos de n bits. Además se ha de agregar el

hardware necesario para transformar el cociente de dígitos con signo en una notación estándar en base 2. El divisor debería además tener una etapa previa para la detección de la división por cero y que x y d estén en el rango necesario.

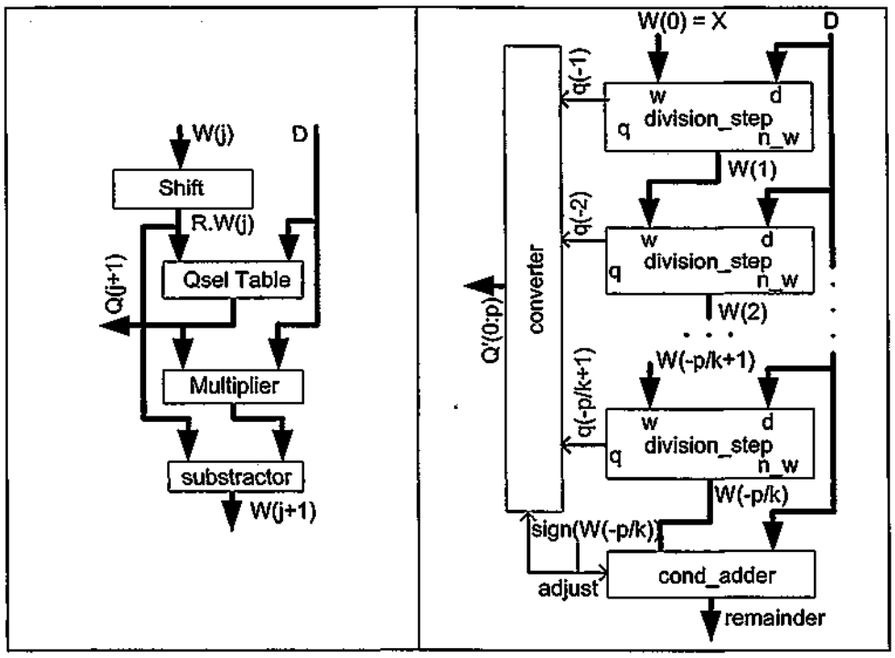


Figura 5.23. Arquitectura para el algoritmo SRT. a) Paso de división (*division_step*) y circuito combinacional.

La división de x/d produce k bits del cociente q en cada iteración. El dígito del cociente q(i) se representa con una notación en base r. El primer residuo w(0) es inicializado con x. En la iteración j el residuo $w(j)$ es desplazado a izquierda k bits (produce $r.w(j)$). Basado en algunos de los dígitos superiores de $r.w(j)$ y d (nr y nd bits respectivamente) se puede deducir el siguiente digito del cociente. Finalmente se realiza la substracción a $r.w(j)$ el producto $q(j+1)*d$ para formar el próximo residuo $w(j+1)$.

En una implementación hardware, existen varios parámetros importantes en un divisor SRT:

- **Base r:** El rol más importante es la base $r=2^k$ en que se realiza la operación. Para mayores valores de k se necesitan menos iteraciones pero cada iteración es más

compleja (tablas Qsel mucho mayores y productos *q(i+1)*d* más complejos). Valores de base grandes (mayores que 16, K=4) son impracticas en las FPGAs actuales.

- **Representación del cociente:** La representación en base 2^k usa un sistema representación redundante de dígitos con signo, para asegurarse que la determinación del digito del cociente solo se basa en los algunos de los dígitos más significativos *nr* y *nd* del residuo y el divisor respectivamente. En un sistema de representación no redundante, los dígitos en base *r* se representan con exactamente *r* dígitos cociente {0, 1, . . . , *r*}. En un sistema redundante se utilizan más de *r* valores. La representación de los dígitos tienen la forma $\{-\alpha, -\alpha+1, \ldots, -1, 0, 1, \ldots, \alpha-1, \alpha\}$, es decir, un conjunto simétrico y consecutivo de valores enteros con un dígito máximo α. Para ser un sistema redundante, *a* debe cumplir la relación $a \geq \lceil r/2 \rceil$. El grado de redundancia se mide por el factor de redundancia (*redundancy factor*), definido como $\rho = a/(r-1)$. En sistema para $a = \lceil r/2 \rceil$ es conocido como redundante mínimo (*minimally redundant*), mientras que con $a = r-1$ (y por tanto $\rho = 1$) es llamado redundante máximo *(maximally redundant).* Si $a > r-1$ y $\rho > 1$, el conjunto de dígitos es conocido como sobre redundante (*over-redundant*). Por ejemplo, para la representación en base-4, el conjunto de dígitos mínimo es {-2,-1, 0, 1, 2} ($a = 2$, $\rho = 2/3$), el redundante máximo es {-3,-2,..., 2, 3} ($a = 3$, $\rho = 3/3$) y el sobre redundante es {-4,-3,..., 3, 4} ($a = 4$, $\rho = 4/3$). Mayores valores de *a* conducen a selecciones más simples de cociente (menores valores de *nr* + *nd* para direccionar la tabla *Qsel*). En esta sección solo se utiliza RAM distribuida (basada en LUTs, dentro de los *slices*) para la implementación de las tablas de selección. Los bloques específicos de RAM (llamados BRAM en Virtex and Virtex II) pueden ser utilizados para generar las tablas *Qsel.*

- **Representación del residuo w(j):** Tradicionalmente, en tecnologías VLSI se utilizan representaciones redundantes tipo *carry-save*, con el objetivo de acelerar la resta $r.w_{(j)}-q_{(j+1)}.D$. La utilidad de esta técnica en FPGAs es relativa, donde la lógica dedicada de acarreo hace que los sumadores tradicionales tipo *ripple-carry* sean más eficientes para pequeños y medianos tamaños de datos, tanto para la suma como para la resta. En esta sección se analizan sistemas no-redundantes (complemento a dos) y redundantes (*carry-save*).

5.5.4 Implementaciones combinacionales en FPGAs para números fraccionarios

En ésta sección se describen los detalles en la implementación de divisores para números fraccionarios. Primero los algoritmos con y sin restauración, y luego los algoritmos SRT base-2, -4, -8, -16 con resto en complemento a dos. Finalmente, se examina una implementación original del algoritmo SRT base 2 con resto en formato carry-save.

5.5.4.1 Algoritmo con restauración y sin restauración

La división de enteros como se mencionó trata fundamentalmente con los algoritmos con y sin restauración. Ajustar estos algoritmos para operandos fraccionarios es trivial. El arquitectura de división con restauración, implementada con el algoritmo representado en la Figura 5.17, necesita p celdas *restoring_division_step*. Cada una de ellas necesita un restador de $(n+1)$-bit y un multiplexor 2 a 1 de n-bit, esto quiere decir que necesita $(n+1)$ *slices* y un retardo de $2.T_{lut} + n.T_{muxcy} + T_{net}$. Entonces, un divisor de n-bits con p-bits de precisión tiene un área y retardo de:

$$C_{rest}(n, p) = p.(C_{rest\_div\_step}(n)) = p.(n+1) = p.n + p \text{ slices}$$

$$T_{rest}(n, p) \approx p.(T_{rest\_div\_step}(n) + T_{net}) = 2.p.T_{lut} + p.n.T_{muxc} + 2.p.T_{net}$$

El algoritmo sin restauración implementado en Virtex ó Virtex II es más eficiente. La celda *nonrestoring_division_cell* se implementa con p $(n+1)$-bits sumador-restador. Cada celda necesita $n/2+1$ *slices* y tiene un retado de $T_{lut} + n.T_{muxcy}$. Si se requiere el resto final, un sumador condicional $(n/2$ *slices*) será necesario para ajustar el resto cuando sea negativo. Área y retardo el algoritmo sin restauración.

$$C_{nr}(n, p) = p.(C_{nonr\_step}(n)) + C_{cond\_adder}(n) = p.(n/2+1) + n/2 = p.n/2 + p + n/2 \text{ slices.}$$

$$T_{nr}(n,p) \approx p.(T_{nonr\_step}(n) + T_{conec}) + T_{cond\_adder}(n) + T_{net} = (p+1).T_{lut} + p.(n+1).T_{muxc} + (n+1).T_{net}$$

5.5.4.2 Algoritmo SRT base 2 resto complemento a dos

En ésta implementación en base 2 se utiliza para la representación del residuo el complemento a la base. Esto permite que la tabla Qsel sea trivial (de hecho no existe),

ya que, se puede utilizar los dos bits de mayor peso para determinar la operación a realizar (Tabla 5.11).

W(n:n-1) = srn(1:0)	Valor Residuo	Operación a Realizar	Digito resultado Q(i)
0 0	$0 \leq r < 1/2$	Nada	0
0 1	$1/2 \leq r < 1$	Restar Divisor	1
1 0	$-1 < r \leq -1/2$	Sumar Divisor	-1
1 1	$-1/2 \leq r < 0$	Nada	0

Tabla 5.11. Operaciones a realizar por el SRT en base 2

De este modo se pueden integrar: Qsel, el multiplicador (solo multiplica por 1, 0 o -1) y el restador de la Figura 5.23 en una única celda *srt_b2_division_step*. Esta celda ocupa $(n+1)/2$ slices (n+1 ya que se opera con un bit más de precisión que el tamaño de los operandos). El contenido de un *slice* de la celda *srt_b2_division_step* se puede ver en la figura 5.4. El valor carry(-1) se carga con la señal srn(0).

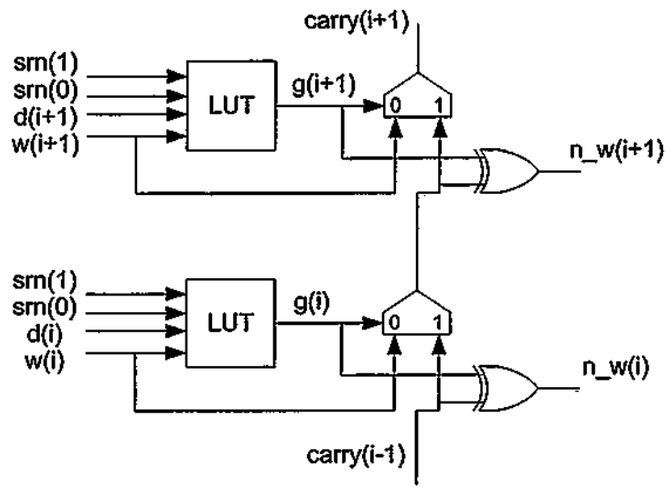


Figura 5.24. Arquitectura de la celda *srt_r2_division_step* para el algoritmo SRT en base 2.

La función lógica a realizar por la LUT (g(i)) es la descrita en la Tabla 5.12. Para el primer caso srn = 00, se pasa el operando W directo, en el segundo (srn = 01) al realizar la suma se realiza la operación W ⊕ D como en la suma normal. En el tercer caso (srn = 10) al realizar la resta con lo que se calcula not(W ⊕ D), además en el carry(-1) se ingresa 1. En el último caso (srn = 11) se debe dejar el operando W directo, pero dado que la entrada carry(1) = srn(0) = 1 la función g(i) computa la operación not(w). Se puede observar que esto ultimo funciona, por un lado la salida de la puerta XOR calculará not(w) ⊕ 1 = w, y por otro la salida carry(y) será siempre uno, dado que el multiplexor de acarreo realizará la función not(not(w)).w+not(w) * 1 = 1. El retardo de la celda *restoring_division_step* será: $T_{srt\_step\_r2}(n) = T_{lut} + (n+1).T_{muxcy}$

	s1	s0	d	W(j)	g(i)
Operando W Directo	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	0
	0	0	1	1	1
Suma W + D	0	1	0	0	1
	0	1	0	1	0
	0	1	1	0	0
	0	1	1	1	1

	s1	s0	d	W(j)	g(i)
Resta W – D	1	0	0	0	0
	1	0	0	1	1
	1	0	1	0	1
	1	0	1	1	0
Niega W	1	1	0	0	1
	1	1	0	1	0
	1	1	1	0	1
	1	1	1	1	0

Tabla 5.12. Función lógica implementada en la celda S_R_Nada del algoritmo SRT base 2

La arquitectura del divisor SRT base 2 se puede ver en la Figura 5.25. Las celdas *srt_r2_division_step* realizan la multiplicación y resta antes descrita y da como salida el nuevo resto W(i+1) y el digito q(i) del cociente (representado con dos bits y correspondientes a los bits n-1 y n-2 del resto W(i) de entrada).

La celda *cond_adder* es un sumador condicional de *n* bits que se utiliza en le caso que el último resto parcial fuese negativo. La señal *adjust* es simplemente el bit de signo del último resto. El área y retardos son los ya descritos.

La celda *converter*, realiza la operación de transformar los dígitos q(i) ∈ {-1, 0, 1} en binario puro. Adicionalmente resta uno en caso que se deba realizar el ajuste en el resto, en aquellos casos que el último resto w(p) hubiese sido negativo. Básicamente lo

que implementa es un restador que suma el complemento más uno. Esta suma de uno se realiza a través del carry(-1) el que es a su vez se conecta a la señal que informa si ha habido ajuste (*adjust*).

El área necesaria es $p/2$ slices y el tiempo es similar que el de un sumador/restador. La función lógica implementada por cada LUT es: not(q(i)(0) ⊕ q(i)(1)), la señal que debe entrar en el muxcy es q(1)(0).

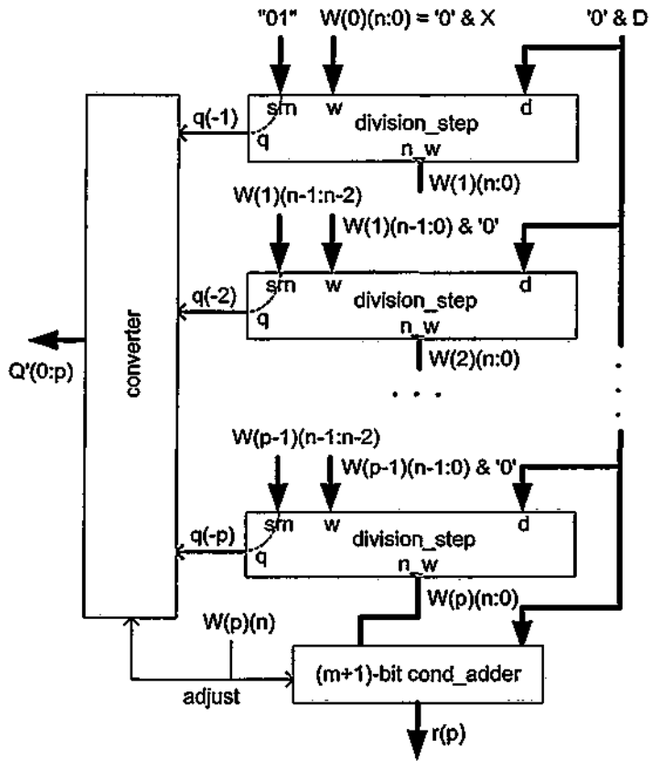


Figura 5.25. Esquema para el divisor SRT base 2.

El área total corresponde a p celdas *srt_r2_division_step* ($p.(n+1)/2$ slices), un sumador condicional ($n/2$) y una celda ajuste ($p/2$). El camino crítico pasa por los p *srt_r2_division_step* y por el *condicional_adder*, siendo similar al divisor sin restauración.

La diferencia es que los tiempos de conexión son más lentos productos del mayor *fan-out* de las conexiones.

$C(n,p) = p.(n+1)/2 + n/2 + p/2 = (pn+n+2.p)/2.$ *slices.*

$T(n,p) = p.T_{lut} + p.t_{connection} + t_{lut} + max(p+1,n+1).t_{muxcy} + (n+1).t_{muxcy}$

Un aspecto interesante en esta implementación, es la potencial reducción consumo debido a la menor actividad, ya que en promedio el 50% de las veces no se realiza ni suma ni resta.

5.5.4.3 Algoritmo SRT base 4 resto complemento a dos

En ésta implementación en base 4 se utiliza para la representación del residuo el complemento a la base. La tabla Qsel utiliza 5 bits para determinar el cociente en el rango {-3,-2,-1, 0, 1, 2, 3} (según PD-plot [Par00]). Para codificar el cociente se utilizan 3 bits en formato signo-valor absoluto (SVA). El primer bit del cociente (el signo) s_r_n(2) se determina por el primer bit del resto W(n+2) y se utiliza la lógica invertida (1 positivo, 0 negativo). El modulo del cociente se obtiene en función de los siguientes 3 bits del resto W(n+1:n-1) a excepción del caso "011" donde se consulta el bit n-1 del dividendo para decidir. El cociente se codifica según lo expresado en la Tabla 5.13.

Bits del Resto W(n+2:n-1)-d(n-1)	Qi binario	Qi decimal
0 0 0 0	1 0 0	0
0 0 0 1	1 0 1	1
0 0 1 0	1 1 0	2
0 0 1 1 - 0	1 1 0	2
0 0 1 1 - 1	1 1 1	3
0 1 0 0	1 1 1	3
0 1 0 1	1 1 1	3
0 1 1 0	1 1 1	3
0 1 1 1	1 1 1	3

Bits del Resto W(n+1:n-1)-d(n-1)	Qi binario	Qi decimal
1 0 0 0	0 0 0	-0
1 0 0 1	0 0 1	-1
1 0 1 0	0 1 0	-2
1 0 1 1 - 0	0 1 0	-2
1 0 1 1 - 1	0 1 1	-3
1 1 0 0	0 1 1	-3
1 1 0 1	0 1 1	-3
1 1 1 0	0 1 1	-3
1 1 1 1	0 1 1	-3

Tabla 5.13. Selección del dígito Qi en el SRT en base 4.

Para la multiplicación se utiliza las puertas *mul_and* que acompañan a cada LUT para llevar acabo la multiplicación eficiente por dos bits. Esto permite realizar la multiplicación prácticamente con el mismo costo en área que un multiplexor 2-1 de n bits. Requiere $n/2 + 1$ *slices* para multiplicar un natural binario de n nits por otro de 2 bits. El retardo es similar al de un sumador de n+2 bits, es decir: $T_{mul2bits} = t_{lut} + (n+1).t_{muxcy} + t_{connection} + t_{xor2}$,

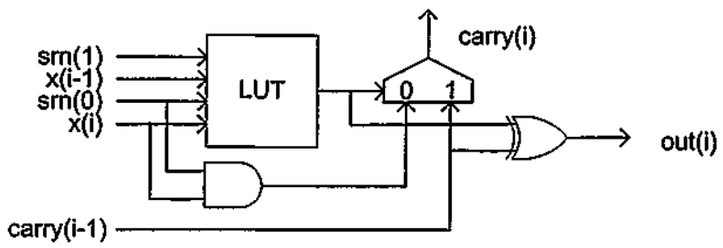


Figura 5.26. Celda para el cálculo de x(n:0) × srn(1:0)

El esquema del divisor SRT base 4 se puede ver en la Figura 5.28, mientras que la celda *srt_r4_division_step* se muestra en la Figura 5.27. La celda *srt_division_step_r4* necesita 3 LUTs para la función Qsel, es decir 2 *slices*; el multiplicador $(n+2)/2$ *slices*; y el sumador condicional $(n+2)/2$ *slices*. Con lo que el costo total: $C_{div\_step\_r4} = n+4$ *slices*.

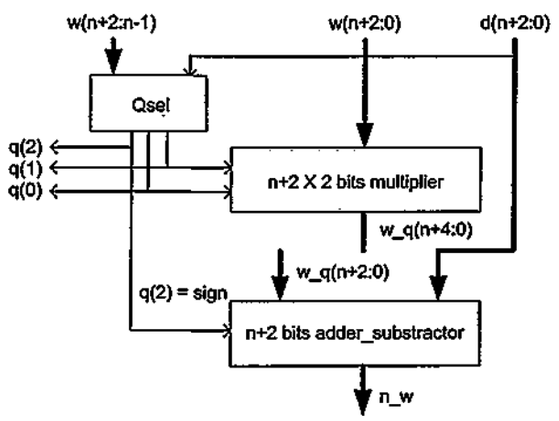


Figura 5.27. Celda *Division_step_r4* para el divisor SRT base 4 con resto en complemento a dos.

El retardo estará constituido por tiempo de *Qsel*, más el retardo de la multiplicación y el tiempo de ejecución de la suma_resta. Además, el cálculo de *Qsel* se realiza con un *slice* configurado como una función de 5 entradas (usando MuxF5) para el bit srn(0). El bit srn(1) se calcula con una LUT normal, mientras que el bit srn(2) -el del signo no- necesita calcularlo es W(n+2). Luego el retardo es:

$$T_{dir\_step\_r4} = 3.T_{lut} + (n+2).T_{muxcy} + 2.T_{net}$$

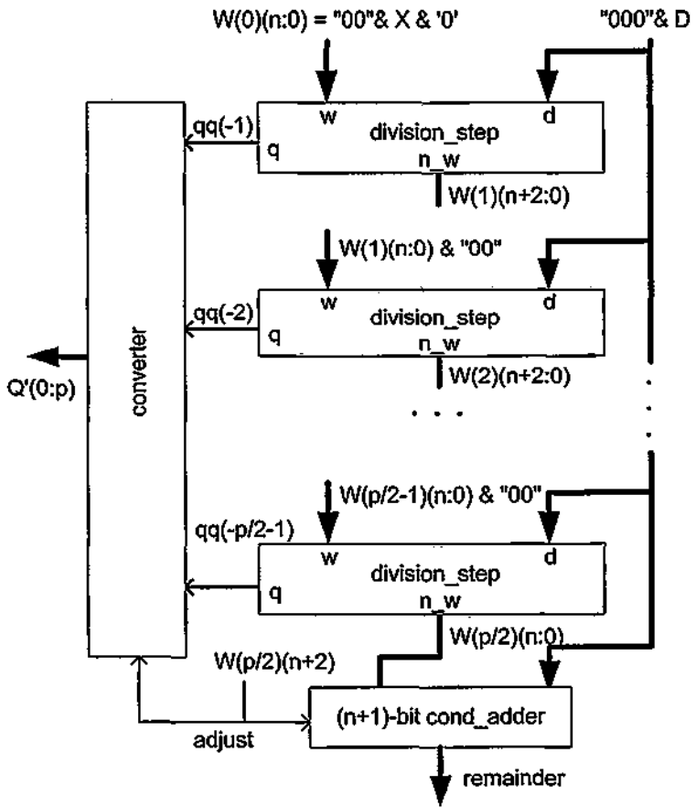


Figura 5.28. Divisor SRT base 4 con resto en complemento a la base.

Celda converter:

El conversor de dígitos en SVA en base 4 de un dígito (2 dígitos binarios), a binario complemento a dos se puede mapear eficientemente en la estructura de la FPGA. La Figura 5.29 muestra el contenido de un *slice* que implementa la conversión de un digito

SVA en 2 bits. La entrada *carry(-1)* se alimentará de la señal *adjust* que sumará uno cuando no se deba ajustar y cero en caso contrario.

El signo de dígito SVA (qq(i)(2)) se ingresa en la entrada complementada del MUXCY. La eficiente implementación de esta conversión depende de que la señal qq(i)(2) se haya codificado con lógica invertida, es decir (0-negativo, 1-positivo). La función o(2i) = not (qq(i)(0)) y o(2i+1) = not (qq(i)(1)).

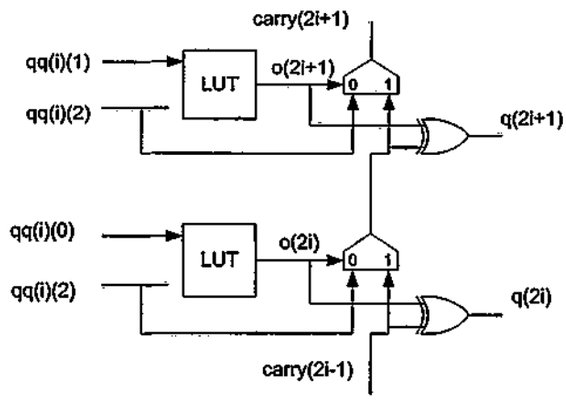


Figura 5.29. Contenido del *slice* perteneciente a la celda converter para el divisor SRT base 4.

Justificación de la elección del conjunto de cocientes:

Estas son algunas de las ventajas en el conjunto redundante máximo, es decir Qi ∈ {-3,...,3} en vez de mínimo redundante Qi ∈{-2,...,2} en la implementación en FPGAs:

- El producto W*q(i), si bien es más simple en el segundo caso, solo un multiplexor 2-1 de n+2 bits, ocupa lo mismo que el multiplicador por 2×(n+2) bits. El retardo en el primer caso es T_{lut} contra T_{lut} + (n+2).T_{muxcy} en el segundo caso.
- Como contrapartida la tabla *Qsel*, es mucho más simple en el caso de utilizar α=3. La función *Qsel* en la representación redundante mínima usa 5 bits del resto y 4 del divisor. El área y retardo de *Qsel* para la representación redundante mínima son 3

slices y $2T_{lut}+T_{net}$ respectivamente, versus 2 *slices* y T_{lut} para la representación redundante máxima.

- La conversión final a binario se puede hacer eficientemente si se utiliza lógica invertida en la representación del signo de los dígitos del cociente.

5.5.4.4 Algoritmo SRT base 8 resto complemento a dos

En la implementación en base 8 se utiliza para la representación del residuo el complemento a la base. La tabla *Qsel* utiliza 9 bits para determinar el cociente en el rango {-7, . . ., 0, . . ., 7} (según PD-plot de la Figura 5.30). Para codificar el cociente se utilizan 4 bits en formato signo-valor absoluto (SVA). El primer bit del cociente (el signo) s_r_n(3) se determina por el primer bit del resto W(n+3) y se utiliza la lógica invertida (1 positivo, 0 negativo). El modulo del cociente se obtiene en función de los siguientes 3 bits del resto W(n+3:n-2) y tres bits del divisor D(n-2:n-4).

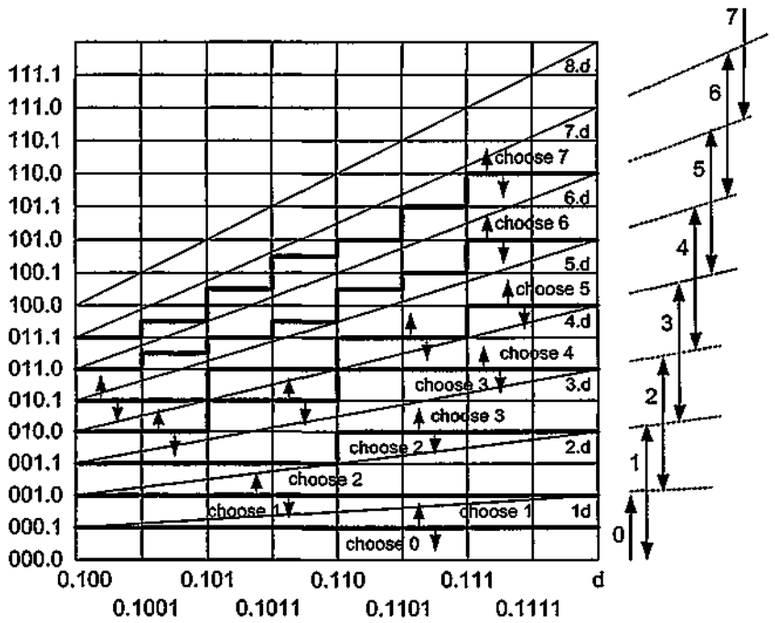


Figura 5.30. PD-Plot para SRT base 8 con cociente en el rango {-7, -6, -5, ... , -1, 0 , 1, ... , 6, 7}.

La celda *srt_division_step_r8* es similar a la de base 4 (Figura 5.27), solo que la multiplicación es por 3 bits y la tabla *Qsel* tiene 9 bits de entrada y 4 de salida (signo más 3 bits). La celda *converter* para base 8 es similar que en el caso de base 4, solo que se utilizan 3 LUTs por cada dígito. Por tanto área y retardo es:

$C_{srt\_div\_step\_r8} = 3/2.n + 22$ slices.

$T_{srt\_div\_step\_r8} = 7.T_{lut} + (n+3).T_{muxcy} + 6.T_{net}$.

El circuito, como en base-2 y base-4 necesita un sumador condicional para el ajuste del resto ($n/2+1$ *slices*), y un conversor a complemento a dos (($p+1$)/2 *slices*). Por tanto el área y el retardo del SRT base 8 son:

$$C_{srt\_r8}(n,p) = p/3.(C_{div\_step\_r8}(n)) + C_{cond\_adder}(n) + C_{converter}$$

$$= p/3.(3/2.n+22)+n/2+(p+1)/2+1 = p.n/2 + n/2 + 47/6.p + 2 \text{ slices}$$

$$T_{srt\_r8}(n, p) \approx p/3.(T_{div\_step\_r8}(n)+T_{net})+T_{cond\_adder}(n)+T_{net}$$

$$= (7/3.p+1).T_{lut}+(p.n/3+n+2).T_{muxcy}+(7/3.p+1).T_{net}$$

5.5.4.5 Algoritmo SRT base 16 resto complemento a dos

En la implementación en base 16 se utiliza para la representación del residuo el complemento a la base. La tabla Qsel utiliza 12 bits para determinar el cociente en el rango {-15, .. ., 0, ..., 15} (según PD-plot de la Figura 5.31). Para codificar el cociente se utilizan 5 bits en formato signo-valor absoluto (SVA). El primer bit del cociente (el signo) s_r_n(5) se determina por el primer bit del resto W(n+5) y se utiliza la lógica invertida (1 positivo, 0 negativo). El módulo del cociente se obtiene en función de los siguientes 6 bits del resto W(n+2:n-4) y cuatro bits del divisor D(n-2:n-6). Un total de 268 LUTs dentro de 141 *slices* son necesarios para la tabla *Qsel* en Virtex, y 221 LUTs en 181 *slices* en Virtex II. La diferencia entre las familias de FPGA son debido a la disponibilidad de los multiplexores dedicados *muxF7* y *muxF8* en Virtex II.

La celda *division_step_r16* es similar al caso base 4 y 8, solo que la multiplicación es por 4 bits y la tabla *Qsel* tiene 12 bits de entrada y 5 de salida (signo más 4 bits). La celda *converter* es también es similar a las anteriores, solo que se utilizan 4 LUTs por cada dígito. Por tanto $C_{div\_step\_r16} = 2.n + 144$ slices y $T_{div\_step\_r16} = 8.T_{lut} + (n+4).T_{muxcy} + 6.T_{net}$

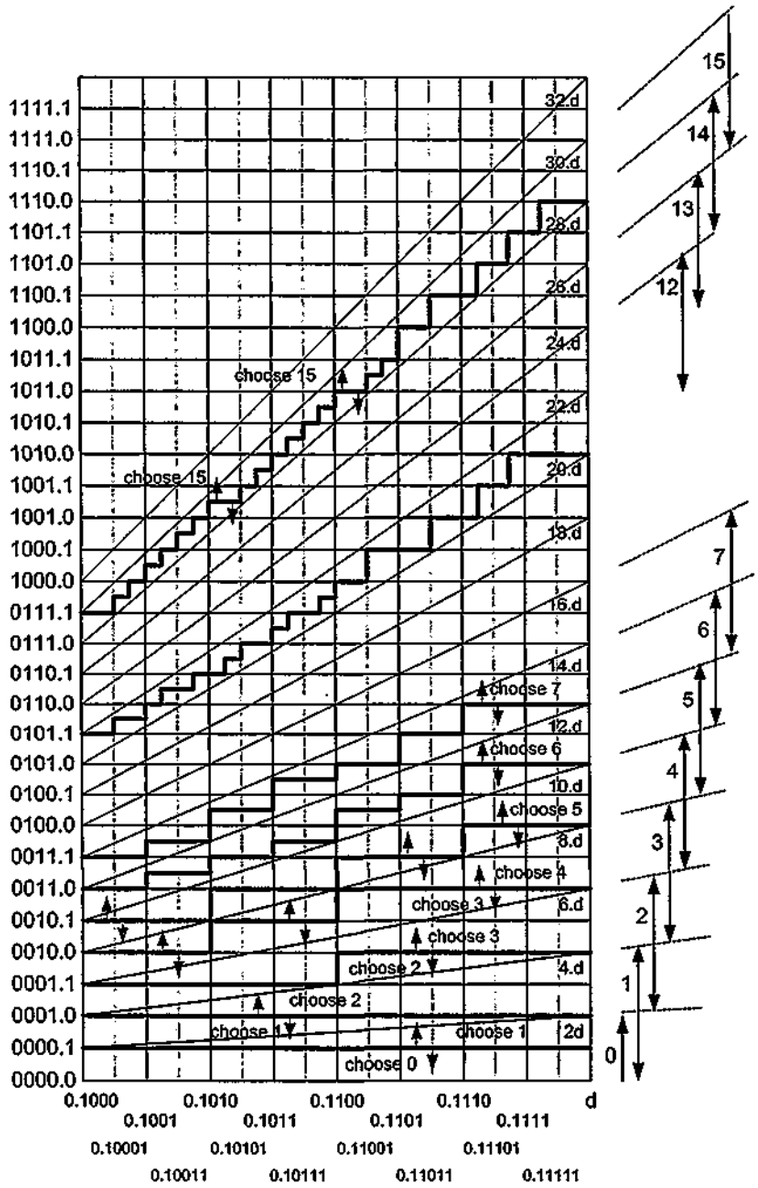


Figura 5.31. PD-Plot para SRT base 16 con cociente en el rango {-15, -14 , ... , -1, 0 , 1, ... , 14, 15}.

Como en los casos anteriores, un sumador condicional de (n+1) bits es necesario si se necesita el resto, así como la celda *converter* para el cociente. La implementación completa pues tiene:

$$C_{srt\_r16}(n,p) = p/4.(C_{div\_step\_r16}(n))+C_{cond\_adder}(n)+C_{converter} = p/4.(2.n+150)+n/2+p/2+2$$
$$= p.n/2+n/2+36.p+2 \ \ slices.$$

$$T_{srt\_r16}(n,p) \approx p/4.(T_{div\_step\_r16}(n)+T_{net})+T_{cond\_adder}(n)+T_{net}$$
$$= (2.p+1).T_{lut}+(p.n/4+n+2).T_{muxcy}+(7/4.p+1).T_{net}$$

5.5.4.6 Algoritmo SRT base 2 resto *carry-save*

En ésta implementación en base 2 se utiliza para la representación del resto o residuo en formato *carry-save*. El diagrama de bloques del divisor se puede ver en la Figura 5.32. Para la implementación de la celda *division_step* se analizan dos alternativas diferentes.

La primera aproximación para la celda *division_step* es la que se sugiere en la Figura 5.33, donde se utiliza un sumador de 3 bits para componer la "cabecera" del resto parcial y una tabla de *look-up* para obtener las señales *q_pos* y *q_neg* que codifican el digito seleccionado.

En la bibliografía [Par00][Erce04] se sugiere la necesidad de sumar los 4 bits de mayor peso para direccionar la tabla de selección de cocientes. No obstante en ésta tesis se ha probado empíricamente que 3 bits son suficientes. Los valores que calcula la tabla *Qsel* se pueden ver en la Tabla 5.14.

Por otra parte, el multiplexor de la Figura 5.23.a y el sumador *carry-save* se puede integrar en una sola LUT utilizando las puertas *xorcy* y el *muxcy*. Por cuestiones del rutado interno del *slice* solo se puede integrar la suma de un dígito en cada uno de ellos. En síntesis la celda tiene un área y retardo de:

$C_{cell\_cs\_add} = n + 4$ slices.

$T_{cell\_cs\_add} = 3.T_{lut} + 3.T_{muxcy} + 2.T_{net} + 2.T_{xor}$

Como en las implementaciones descritas anteriormente, es necesario agregar un sumador condicional para el ajuste del resto, y un conversor de dígitos con signo magnitud a complemento a dos. Con esto, la primera versión del divisor SRT con *carry-save* (usando un sumador) tendrá un área y retardo de:

$$C_{srt\_cs\_add}(n,p) = p.C_{cell\_cs1} + C_{cond\_adder} + C_{converter} = p.n + 4.p + n/2 + 1 + n/2 + 1$$

$$= (p+1).n + 4.p + 2 \text{ slices.}$$

$$T_{srt\_cs\_add}(n,p) \approx p.(T_{cell\_cs1} + T_{net}) + 2.T_{net} + T_{add} + T_{cond\_add}$$

$$= (3p+2).T_{lut} + (2n+3p+2).T_{muxcy} + 2p.T_{xor} + (3p+2).T_{net}$$

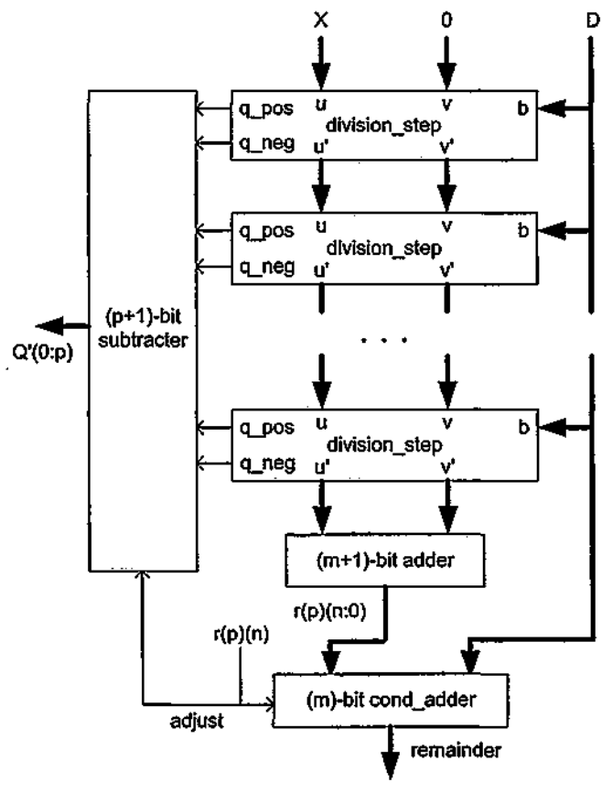


Figura 5.32. Arquitectura del SRT en base 2 con el resto en formato carry-save.

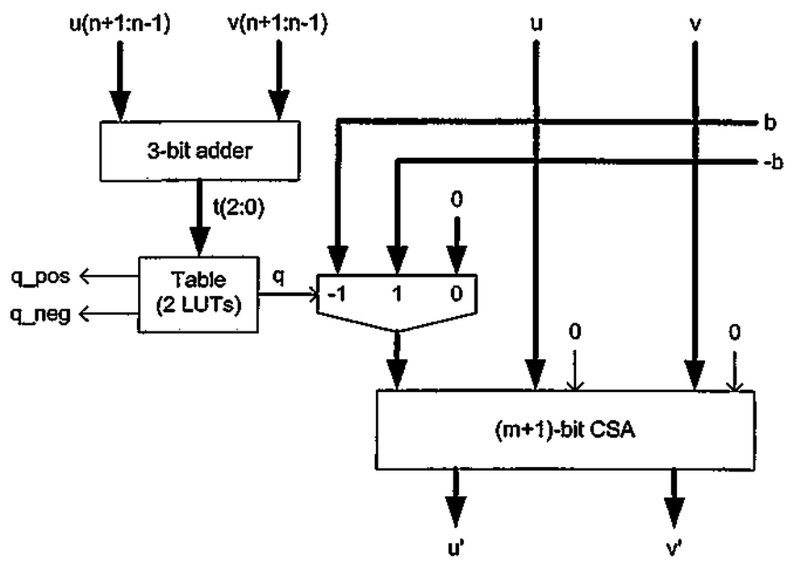


Figura 5.33. Celda básica del divisor SRT en base 2 con resto en formato *carry-save*.

Valor $u_{(n+1:n-1)} + v_{(n+1:n-1)}$	Min valor u+v	Max valor u+v	Min decimal	Max decimal	Digito elegido
000	000xx…x	001xx…x	0	½	1 (10)
001	001xx…x	010xx…x	½	1	1 (10)
010	010xx…x	011xx…x	1	1 ½	1 (10)
011	011xx…x	100xx…x	1 ½	-2	-1 (01)
100	100xx…x	101xx…x	-2	-1 ½	-1 (01)
101	101xx…x	110xx…x	-1 ½	-1	-1 (01)
110	110xx…x	111xx…x	-1	- ½	-1 (01)
111	111xx…x	000xx…x	- ½	0	0 (00)

Tabla 5.14. Tabla Qsel para el SRT base 2 con *carry-save*.

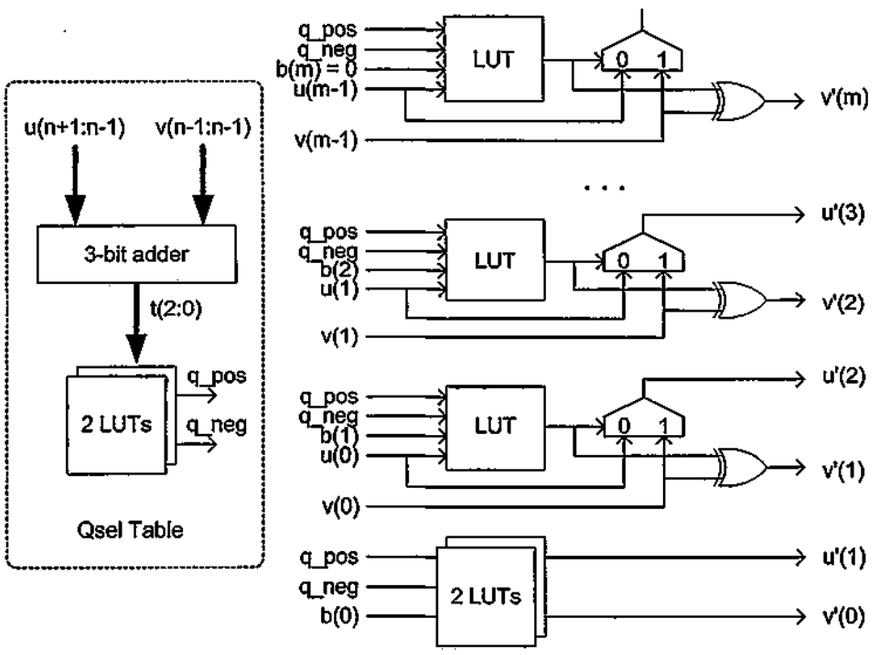


Figura 5.34. Implementación en FPGA de la celda básica del divisor SRT en base 2 con resto en formato *carry-save*.

La segunda versión, optimiza la selección del dígito Q(i), remplazando la suma y la selección de 3 bits, por una tabla de *look-up* (Figura 5.35). Dado que como se mencionó anteriormente solo bastan sumar 3 bits de u y v, la tabla para *q_pos* y *q_neg* solo será de 6 bits, esto se puede implementar con 8 LUTS (4 *slices*) usando además los multiplexores muxF5 y muxF6. Esto proporciona un área y retardo para la celda *cell_cs_lut* de:

$C_{cell\_cs\_lut} = n + 4$ slices.

$T_{cell\_cs\_lut} \approx 2.T_{lut} + T_{muxf6} + T_{net} + T_{xor} \approx 3.T_{lut} + T_{net} + 2.T_{xor}$

Por tanto el SRT con resto en *carry-save* y *Qsel* implementado completamente con LUTS (*srt_cs_L*) tiene un costo en área y tiempo de:

$C_{srt\_cs\_L}(n,p) = p.C_{cell} + C_{con\_adder} + C_{converter} = (p+1).n + 4.p + 1$ slices.

$$T_{srt\_cs\_L}(n,p) = p.(T_{cell} + T_{net}) + 2.T_{net} + T_{add} + T_{cond\_add}$$

$$= (3p+2).T_{lut} + (2n+2).T_{muxcy} + p.T_{xor} + (2p+2).T_{net}$$

Importante el hecho de que tanto $T_{cell\_cs\_add}$ como $T_{cell\_cs\_lut}$ son constantes, de modo que el tiempo de computación depende de la precisión p (y n if $n > p$) y no solo del ancho de los datos n.

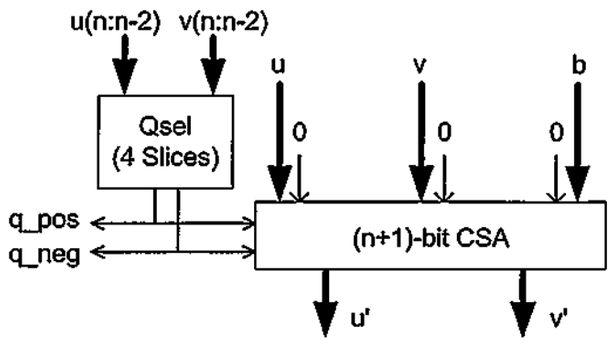


Figura 5.35. Celda de división SRT con resto en *carry-save* (2da versión).

5.5.5 Arquitecturas segmentadas

Para aumentar la velocidad, la segmentación (*pipelining*) es una técnica altamente efectiva sobre todo cuando se trata con lotes grandes de operaciones. En esta técnica cada *LD* etapas de división *division_step* sucesivas se agrega una etapa de elementos de almacenamiento (registros de segmentación) Figura 5.36. De este modo el camino crítico se reduce y consecuentemente se puede aumentar la frecuencia de operación (más detalles de la segmentación en la sección 3.4).

Se han llevado a cabo la segmentación para los divisores sin restauración (*non-restoring*), SRT base 2 y base 4 con resto en complemento a la base, y finalmente SRT con resto en *carry-save*. Se utiliza la denominación de LD (*Logic Depth*) para denominar la cantidad máxima de pasos de división *division_step* entre bancos de registros sucesivos. Las celdas *division_step* para cada implementación son las descritas en 5.4.4.1, 5.4.4.2, 5.4.4.3 y 5.4.4.6 respectivamente.

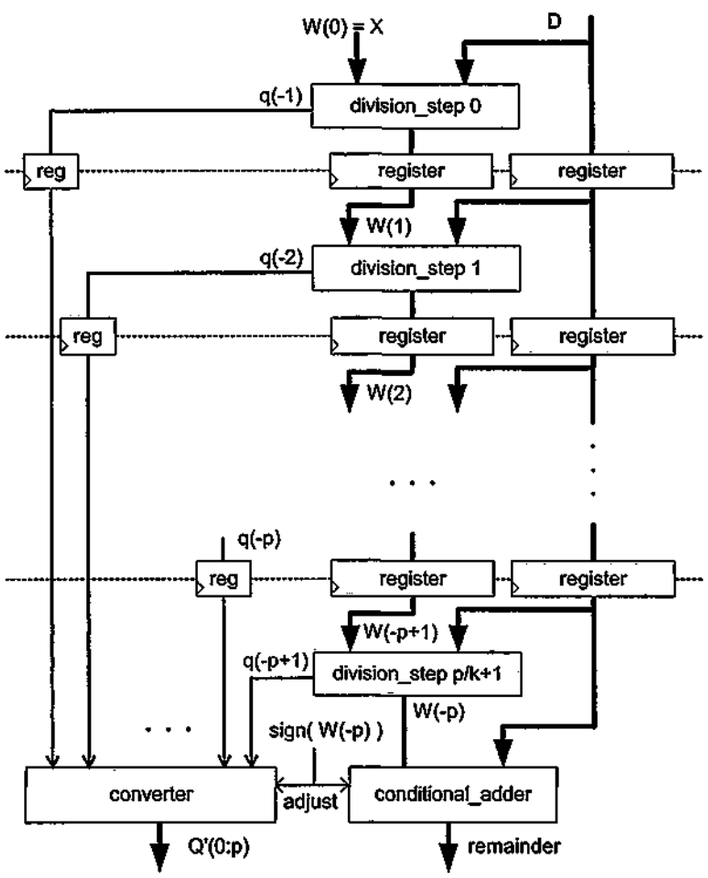


Figura 5.36. Arquitectura segmentada de un divisor SRT con LD = 1.

Para la implementación de las etapas de registros se utilizan los *flip-flops* distribuidos en los *slices*. Adicionalmente, los registros de *deskewing* pueden ser implementado usando registros de desplazamientos implementados en LUT, llamados SRL16 (*Shift Registers in LUT*). Los registros de desplazamiento basados en tablas de *look-up* permiten comprimir 16 *shift-registers* en una sola LUT. La técnica de *pipelining* tiene pues un bajo impacto en área dados los registros distribuidos dentro de los *slices* y de las características de SRL16 de las LUTs. Además, como fue descrito anteriormente, la

segmentación tiene un fuerte impacto en la reducción del consumo por la eliminación de las transiciones espurias (*glitches*).

5.5.6 Circuitos Secuenciales

Para la reducción del área, la principal técnica es secuencializar (al menos parcialmente) la ejecución del algoritmo de división. La arquitectura general agrega una máquina de estados que controla la ruta de datos, compuesta por *g* etapas de división consecutivas y los registros correspondientes para almacenar los valores intermedios (Figura 5.37).

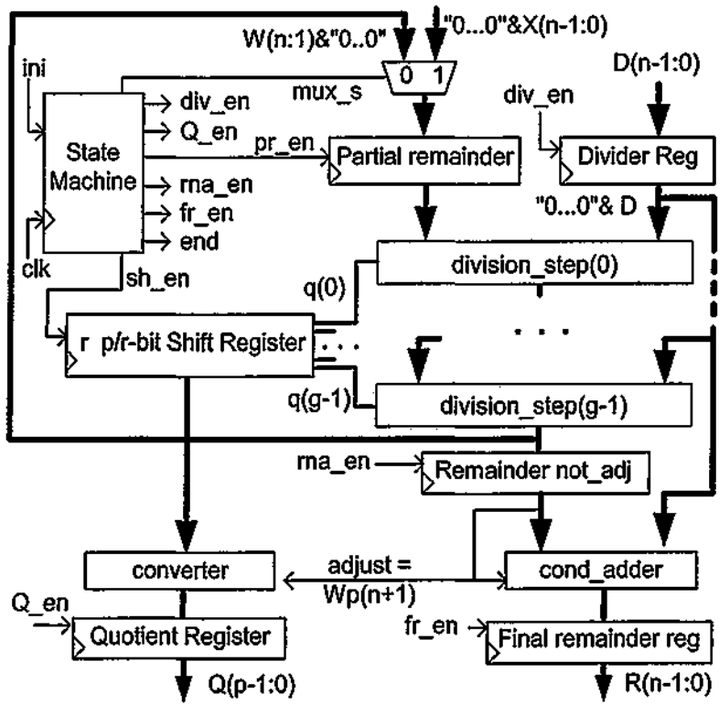


Figura 5.37. Arquitectura secuencial del algoritmo de división SRT.

En cada ciclo de reloj el circuito genera *g.r* bits, mientras un ciclo extra es necesario para calcular el resto. Con esto se necesitan $p/(g.r)$ ciclos para el cálculo completo, con

un ciclo extra para un eventual ajuste del resto. El valor *p* se refiere a la precisión en bits requerida, *g* a la cantidad de etapas de división *division_step* y *r* a la base del divisor.

Las implementaciones secuenciales de estos circuitos son atractivas también desde el punto de vista del consumo. La menor profundidad lógica producida por los registros a cada ciclo de reloj reduce las transiciones espurias, aunque como contrapartida existe un mayor consumo de sincronización debido a la máquina de estados y los registros intermedios.

5.5.7 Resultados en área y velocidad para los divisores para números fraccionarios

Primero se presentan los resultados respecto al área y la velocidad para dispositivos Virtex y Virtex II. Si bien el objetivo es la reducción de consumo, la cuidadosa implementación de los circuitos ha dado interesantes resultados tanto en área como en velocidad.

Los circuitos descritos en las secciones 5.4.4, 5.4.5 y 5.4.6 han sido implementados en Virtex (XCV800hq240-6) y en Virtex II (XC2V1000bg575-6). Los circuitos son descritos en VHDL instanciando cuando es necesario primitivas de bajo nivel como LUTs, *muxcy*, *xorcy*, etc. [Xil03c]. Se utilizo el entorno de desarrollo Xilinx ISE 6.1 [Xil03d], y XST (*Xilinx Synthesis Technologies*) [Xil03e] para la síntesis. La misma asignación de patas, la opción de preservar la jerarquía (*keep hierarchy*), optimización por velocidad y restricciones de tiempo (*timing constrains*) son parte de la estrategia de diseño.

La mayor diferencia entre las implementaciones en las familias Virtex y Virtex II se observan en el retardo del rutado. En Virtex la relación lógica-rutado en el peor camino es en promedio (55%, 45%), mientras que en Virtex II dicha relación es (63%, 37%). Otra diferencia se debe a las conexiones rápidas entre *slices* que conectan los multiplexores *muxF7* y *muxF8* presentes en Virtex II, que permiten implementar grandes bloques combinaciones de forma más eficiente. Así, las tablas de selección (*Qsel*) grandes presentes en base-8 y base-16 se implementan más rápido y requieren menos área.

5.5.7.1 Resultados en implementaciones combinacionales

La Tabla 5.15 muestra para el dispositivo Virtex el área (en LUTs y *slices*) y los retados (total, debido a la lógica y debido al rutado) expresados en *ns*. En la cantidad de LUTs (por ej 2420+12) se expresa por separado las utilizadas para la lógica y las que son solo empleadas para rutar. Los circuitos son los descritos en la sección 5.4.4: división con restauración y sin restauración (*rest, nonrest*); SRT base-2, base-4, base-8, y base-16 con resto en complemento a dos (*srt_r2, srt_r4, srt_r8, srt_16*) y finalmente los dos divisores SRT base-2 con resto representado en carry-save (con sumadores *srt_cs_ad* y con tablas de look-up *srt_cs_L*).

	N	Área		Retardo		
	P	slices	LUTs	total	lógica	rutado
NonRest	40	880	1719	251.7	129.2	122.5
	32	576	1119	180.6	93.3	87.3
	24	336	647	118.7	62.5	56.2
	16	160	303	68.8	37.8	30.9
Rest	40	1640	3279	329.1	146.4	182.7
	32	1056	2111	238.3	108.2	130.1
	24	600	1199	158.4	74.5	83.8
	16	272	543	91.5	44.2	47.3
srt_r2	40	861	1720	293.2	124.3	168.8
	32	561	1120	198.1	90.8	107.3
	24	325	648	125.5	60.8	64.6
	16	153	304	69.2	37.0	32.2
srt_r4	40	940	1936	243.7	114.7	129.0
	32	624	1245	187.8	87.3	100.5
	24	372	741	125.7	63.4	62.3
	16	184	344	82.4	39.8	42.6

	N	Área		Retardo		
	P	slices	LUTs	total	lógica	rutado
srt_r8	40	1112	2420+12	277.3	101.1	176.1
	32	804	1774+10	224.6	83.0	141.6
	24	487	1089+7	154.6	60.2	94.4
	16	243	584+4	83.9	41. 4	42.5
srt_r16	40	2258	4561+44	245.7	95.5	150.2
	32	1666	3358+35	191.1	75.9	115.2
	24	1137	2289+25	138.4	56.4	82.0
	16	676	1390+16	81.8	36.6	45.2
srt_cs_ad	40	1802	2172+1	336.9	132.1	204.8
	32	1186	1461+1	259.2	105.0	154.1
	24	698	901+1	192.8	80.6	112.2
	16	338	439+1	119.6	54.6	65.2
srt_cs_L	40	1779	2072+40	238.6	103.5	135.0
	32	1183	1400+32	179.2	81.1	98.1
	24	695	856+24	141.4	59.9	81.4
	16	335	440+16	87.9	42.8	45.1

Tabla 5.15. Resultados para implementaciones combinacionales en la familia Virtex

Hasta 24 bits, el divisor sin restauración y SRT base-2 muestran los mejores resultados respecto del retardo; para tamaño de operandos mayores SRT resto *carry-save* (*srt_cs_L*), SRT base-16 y SRT base-4 son las mejores opciones. Respecto del área, SRT base-2 y *non-restoring* son siempre las mejores alternativas. Por el contrario la división por restauración y SRT base-16 poseen las peores características. Los mejores resultados en área × retardo (*#slice* × *ns*) lo poseen SRT base-2 hasta 24 bits y SRT base-4 para operandos de mayor tamaño.

La Tabla 5.16 muestra los resultados para la familia Virtex II. Los circuitos implementados son los mismos que se han descrito anteriormente. Se han quitado los valores correspondientes a *restoring* y *srt_cs_ad* dado sus pobres resultados. La arquitectura propuesta en la sección 5.4.4.6, es decir SRT con resto representado en *carry-save* (*srt_cs_L*), posee la mejor *performance*, seguido por SRT base 16 (*srt_r16*). La mejora en el retardo respecto de la división sin restauración es de hasta un 42,5 %. Respecto del área, como en Virtex, SRT base-2 y *non-restoring* son los que menos recursos necesitan. Los mejores resultados en la métrica área × retardo (*#slice* × *ns*) lo posee SRT base-4.

	N	Area		Delay		
	P	slices	LUTs	total	logic	route
Non-Restoring	56	1680	3303	304.0	205.2	98.8
	48	1248	2447	244.9	157.3	87.6
	40	880	1719	175.5	120.2	55.2
	32	576	1119	123.2	85.4	37.9
	24	336	647	82.4	56.4	26.0
	16	160	303	50.1	33.1	17.0
srt_r2	56	1653	3306+2	294.7	201.3	93.5
	48	1225	2448+2	235.8	159.0	76.8
	40	861	1720+2	178.8	119.0	59.8
	32	561	1120+2	131.4	84.5	46.9
	24	325	648+2	89.1	54.8	34.3
	16	153	304	47.5	32.7	14.8
srt_r4	56	3417	3498+1	231.6	160.1	71.5
	48	2545	2614+1	194.3	128.5	65.9
	40	921	1858+1	150.6	101.2	49.5
	32	609	1230+1	114.4	76.0	38.3
	24	361	730+1	82.5	53.9	28.6
	16	177	358+1	49.8	33.4	16.4

	N	Area		Delay		
	P	slices	LUTs	total	logic	route
srt_r8	56	2508	4408+1	266.9	145.9	121.0
	48	1916	3324+1	190.2	122.1	68.1
	40	1399	2384+1	152.7	95.1	57.6
	32	1026	1744+1	129.0	77.1	51.9
	24	652	1068+1	89.8	53.0	36.8
	16	334	536+1	54.5	33.0	21.5
srt_r16	56	4237	6469+196	219.9	120.0	99.9
	48	3429	5147+168	173.9	99.5	74.5
	40	2685	3953+140	144.3	78.5	65.8
	32	2004	2887+112	109.6	62.7	47.0
	24	1388	1949+84	79.7	44.5	35.1
	16	834	1139+56	53.0	28.9	24.1
srt_cs_L	56	3467	3802+56	174.8	104.1	70.7
	48	2545	2874+48	152.0	92.5	59.5
	40	1802	2074+40	122.8	74.6	48.2
	32	1186	1402+32	99.7	62.6	37.1
	24	698	858+24	74.8	45.6	29.2
	16	338	442+16	50.4	32.0	18.4

Tabla 5.16. Resultados para implementaciones combinacionales en la familia Virtex II.

El retardo total en función del tamaño de los datos en Virtex se grafica en la Figura 5.38, en tanto la Figura 5.39 hace lo propio con Virtex II. Finalmente la Figura 5.40 representa el área en *slices* en función del tamaño de los operandos para el dispositivo Virtex II. Cabe aclarar que el área en los dispositivos Virtex es prácticamente el mismo que en Virtex II.

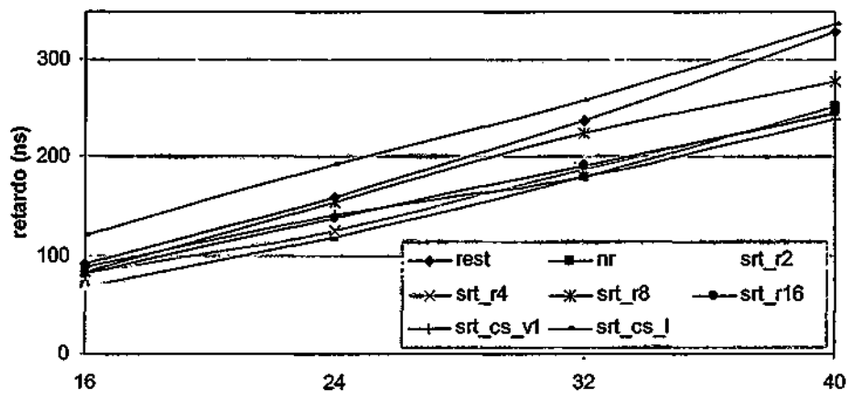


Figura 5.38. Retardo de los divisores en función del tamaño de datos en Vitex.

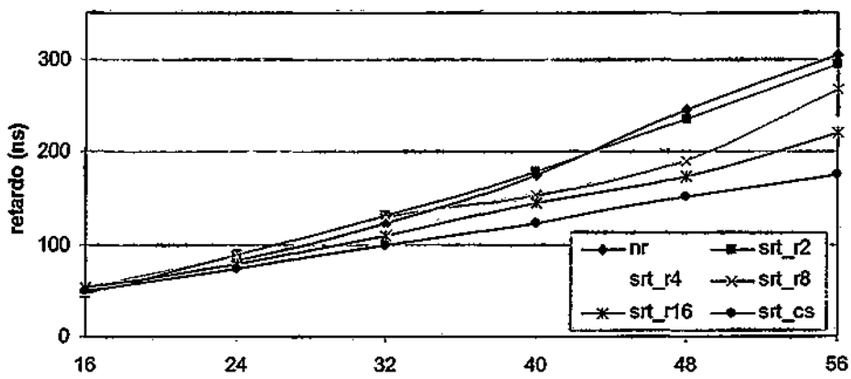


Figura 5.39. Retardo de divisores en función del tamaño de datos en Virtex II.

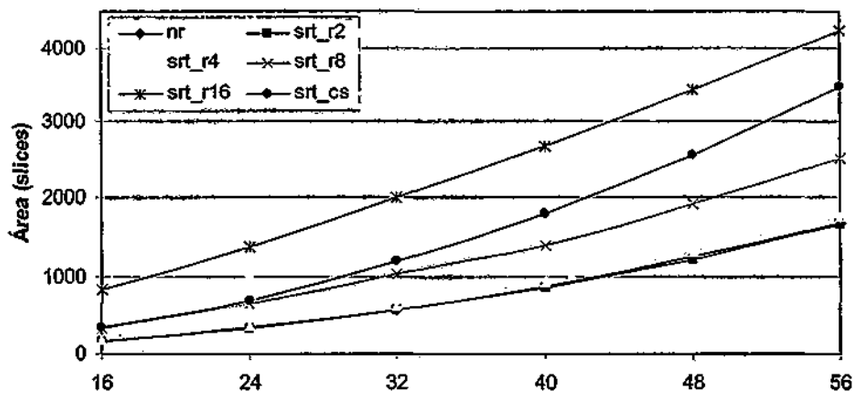


Figura 5.40. Área de los divisores en Virtex y Virtex II.

5.5.7.2 Resultados en implementaciones segmentadas

Los siguientes circuitos fueron segmentados con diferentes granularidades: división sin restauración; SRT base-2, SRT base-2 usando SRL SRT base-4, y SRT con resto representado en *carry-save*. La Tabla 5.17 muestra el área expresada en *slices*, la cantidad de registros (*flip-flops* y SRL, las LUTs configuradas como *shift registers*), y el máximo ancho de banda en MHz, para diferentes divisores de 32 bits y diferentes profundidades lógicas LD (la cantidad de pasos de división entre bancos de registros consecutivos) en dispositivos Virtex. La Tabla 5.18 hace lo propio para Virtex II.

Las diferentes arquitecturas de divisores muestran resultados similares en cuanto a la mejora en la velocidad vs. el incremento en área como lo muestra la Figura 5.41 para los dispositivos Virtex II.

La división SRT en base-4 exhibe los mejores resultados en ambas familias. SRT con resto en *carry-save* muestra muy pobres resultados tanto en área como en velocidad, en tanto la división sin restauración reporta valores muy buenos. Una arquitectura con máxima segmentación (LD = 1) acelera el sistema hasta en más de 20 veces respecto de una implementación totalmente combinacional (LD = 32), con un incremento en el área menor a tres veces.

LD	C	Non-Restoring			SRT radix 2 SRL				SRT radix 4				SRT *carry save*			
		slices	FF	Mhz	slices	FF	srl	Mhz	slices	FF	srl	Mhz	slices	FF	srl	Mhz
1	33	1968	2705	101.7	1747	2274	88	111.7	-	-	-	-	3356	3298	90	79.1
2	16	1256	1328	55.9	1169	1152	52	53.8	1288	1265	39	62,6	2257	1647	52	48.3
3	11	1066	933	49.5	1012	835	48	49.6	-	-	-	-	1939	1164	48	43.1
4	8	943	688	34.5	915	641	40	34.3	967	632	30	39,4	1745	871	40	33.7
5	7	905	617	33.4	888	583	40	32.5	-	-	-	-	1689	780	40	30.7
6	6	866	538	25.3	858	521	36	26.4	907	508	36	31,3	1629	685	36	28.4
7	5	822	454	22.5	825	455	28	23.9	-	-	-	-	1566	586	28	24.9
8	4	779	368	20.0	786	385	32	20.2	835	372	12	21,7	1508	483	16	20.0
11	3	738	289	15.6	725	321	-	14.7	-	-	-	-	1456	386	-	16.8
12	3	740	292	14.4	728	327	-	13.6	782	315	-	16,2	1462	392	-	15.4
16	2	697	208	10.8	675	224	-	10.2	736	222	-	10,6	1355	256	-	11.2
32	1	656	128	5.6	625	128	-	5.2	752	226	-	5,3	1251	147	-	5.9

Tabla 5.17. Profundidad Lógica (LD), ciclos necesarios, área en *slices*, cantidad de registros (FF y SRL) y máximo ancho de banda en MHz para diferentes arquitecturas en Virtex.

LD	C	Non-Restoring			SRT radix-2				SRT radix-4				SRT carry-save			
		slices	FF	Mhz	slices	FF	srl	Mhz	slices	FF	srl	Mhz	slices	FF	srl	Mhz
1	33	2000	2705	182.8	1720	2244	90	182.2	-	-	-	-	3356	3298	90	119.5
2	16	1286	1328	94.0	1171	1152	56	85.7	1305	1265	39	118.5	2257	1647	52	69.8
3	11	1067	933	79.8	1014	835	48	77.3	-	-	-	-	1939	1164	48	71.0
4	8	943	688	55.4	916	641	48	51.4	988	631	30	50.5	1745	871	40	48.0
5	7	908	617	50.1	891	583	40	50.8	-	-	-	-	1689	780	40	50.7
6	6	869	538	42.1	861	521	36	43.6	919	508	36	47.4	1629	685	36	53.0
7	5	827	454	36.2	828	455	28	37.4	-	-	-	-	1566	586	28	41.9
8	4	779	368	30.1	788	385	32	28.8	849	414	12	30.1	1508	483	16	32.0
11	3	738	289	23.9	757	321	-	24.0	-	-	-	-	1456	386		27.1
12	3	748	292	22.1	763	327	-	22.3	751	222	-	23.3	1462	392		28.3
16	2	697	208	15.9	722	255	-	15.9	768	226	-	17.1	1355	256		19.1
32	1	656	128	8.0	756	353	-	8.4	849	414	-	8.9	1251	147		10.0

Tabla 5.18. Profundidad Lógica (LD), ciclos necesarios, área en *slices*, cantidad de registros y máximo ancho de banda en MHz para diferentes arquitecturas en Virtex II.

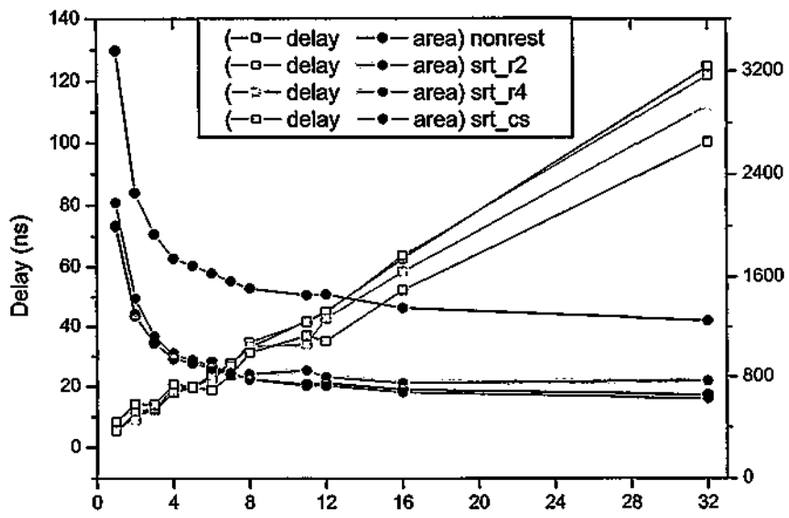


Figura 5.41. Área y retardo en función de la profundidad lógica para divisores de 32 bits en Virtex II.

5.5.7.3 Resultados en implementaciones secuenciales

La Tabla 5.19 muestra los resultados para las implementaciones iterativas en Virtex, en tanto que la Tabla 5.20 hace lo propio para Virtex II. Se muestran, la cantidad de bits calculados en cada ciclo de reloj (G), la cantidad total de ciclos utilizados (C), el área en *slices* y la cantidad de registros utilizados. Luego el periodo y la máxima frecuencia de funcionamiento. Finalmente la latencia en *ns* (L = periodo de reloj × cantidad de ciclos de reloj necesarios) y el área × latencia (A×L) en *ns* × *slices*.

Cuando G crece, la latencia decrece a expensa de mayor área. La mejor relación área × latencia se obtiene para G = 2. Los mejores resultados en cuanto a la latencia los ofrece SRT base-4 en ambas familias. En la Figura 5.42 se muestra los resultados de latencia para los diferentes circuitos secuenciales, en tanto que en la Figura 5.43 se muestra la relación área × latencia.

	G	C	slices	FF	P(ns)	F(MHz)	L(ns)	AxL
Non_rest	1	32	113	203	8,9	112,0	285,7	32280
	2	16	124	202	13,7	72,8	219,8	27258
	4	8	155	200	23,8	42,1	190,2	29480
	8	4	219	196	44,9	22,3	179,6	39331
Srt_r2	1	32	135	240	8,0	124,6	256,9	34677
	2	16	139	236	13,4	74,5	214,8	29864
	4	8	169	236	24,1	41,5	193,0	32612
	8	4	229	236	47,9	20,9	191,7	43902
Srt_r4	2	16	134	221	12,7	78,8	202,9	27192
	4	8	169	221	21,9	45,7	175,2	29607
	8	4	240	222	41,2	24,3	164,6	39504
Srt_r16	4	8	336	255	23,0	43,5	183,9	61786
	8	4	603	294	41,9	23,9	167,5	100974
Srt_cs	1	32	179	269	11,9	83,8	382,0	68375
	2	16	210	267	17,7	56,6	282,6	59338
	4	8	282	267	29,9	33,4	239,3	67481
	8	4	426	267	52,5	19,0	210,2	89538

Tabla 5.19. Resultados implementaciones iterativas en Virtex.

	G	C	slices	FF	P(ns)	F(MHz)	L(ns)	AxL
Non_rest	1	32	113	203	8,9	112,0	285,7	32280
	2	16	124	202	13,7	72,8	219,8	27258
	4	8	155	200	23,8	42,1	190,2	29480
	8	4	219	196	44,9	22,3	179,6	39331
Srt_r2	1	32	135	240	8,0	124,6	256,9	34677
	2	16	139	236	13,4	74,5	214,8	29864
	4	8	169	236	24,1	41,5	193,0	32612
	8	4	229	236	47,9	20,9	191,7	43902
Srt_r4	2	16	134	221	12,7	78,8	202,9	27192
	4	8	169	221	21,9	45,7	175,2	29607
	8	4	240	222	41,2	24,3	164,6	39504
Srt_r16	4	8	336	255	23,0	43,5	183,9	61786
	8	4	603	294	41,9	23,9	167,5	100974
Srt_cs	1	32	179	269	11,9	83,8	382,0	68375
	2	16	210	267	17,7	56,6	282,6	59338
	4	8	282	267	29,9	33,4	239,3	67481
	8	4	426	267	52,5	19,0	210,2	89538

Tabla 5.20. Resultados implementaciones iterativas en Virtex II.

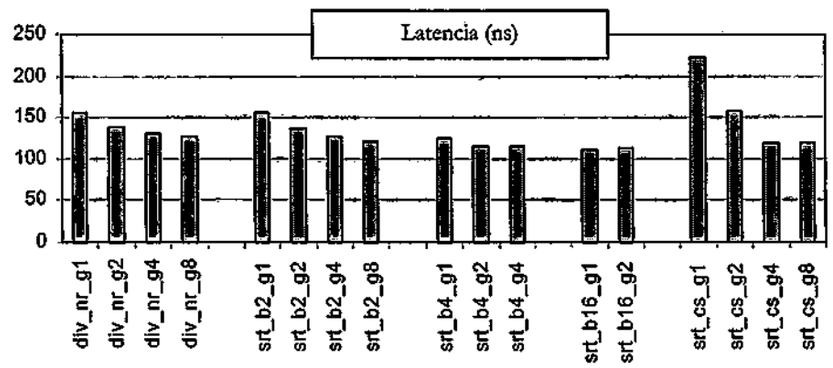


Figura 5.42. Latencia para los diferentes circuitos secuenciales en Virtex II.

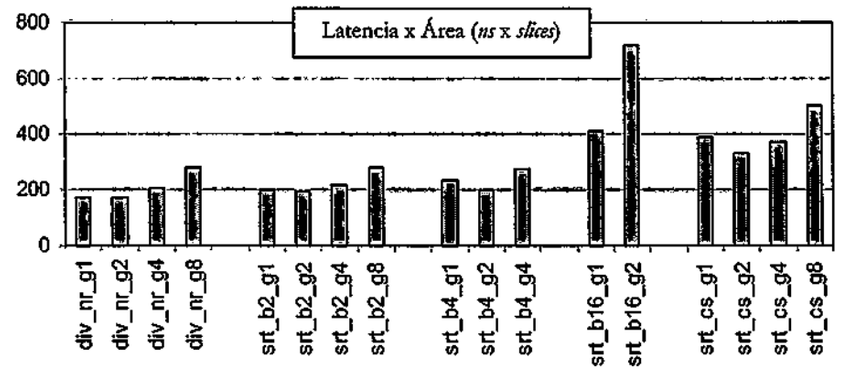


Figura 5.43. Relación área × latencia en circuitos secuenciales en Virtex II.

5.5.7.4 Comparación de resultados en área y velocidad

Se comparan en esta sección, los resultados obtenidos respecto de recientes contribuciones, así como la relación de retardo y área entre las arquitecturas combinacionales, segmentadas y secuenciales.

Trabajos previos: Diferentes implementaciones de los algoritmos SRT fueron presentadas recientemente. En [Beu02] los multiplicadores dedicados presentes en

Virtex II son utilizados para implementar divisores SRT en base-2, -4 y -8. Se utiliza para esto un generador descrito en C++ que produce código VHDL sintetizable. [Lee03] presenta un SRT base-8 con dígitos mínimamente redundantes, además, resultados previos en un SRT base-4 son señalados. En [Wan03], divisores SRT base 2, 4 y 8 son implementados iterativamente, totalmente combinacionales y totalmente segmentados. En [Wan03] se utilizan generadores de módulos descritos en JHDL.

En la Figura 5.44 se muestran las implementaciones presentadas en la sección 5.4.4 comparadas con los trabajos anteriores. De esta tesis se muestran las implementaciones combinacionales de SRT base-2, -4, -16 con resto en complemento a dos (*srt_r2, srt_r4, srt_r16*) y el SRT base-2 con resto representado en *carry-save* (*srt_cs_L*). Adicionalmente, los resultados de la implementaciones de [Lee03] del SRT base-4 y base-8 (*l_r4* y *l_r8*); el mejor resultado de [Beu02], un SRT base-8 que utiliza bloques multiplicadores dedicados de Virtex II *mult*18X18 (*b_r8*), y finalmente la implementación combinacional de divisores SRT base-2 y base-4 de 24 bits presentada en [Wan] (*w_r2, w_r4*) completan la figura.

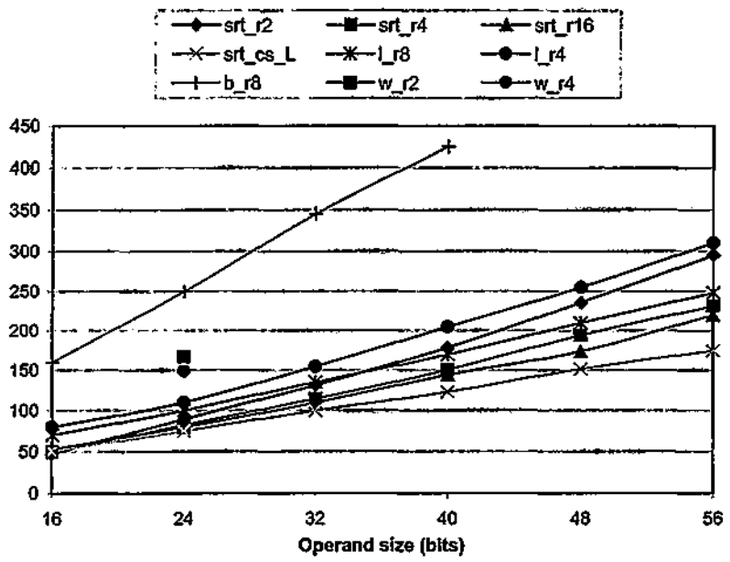


Figura 5.44. Retado (en ns) respecto del tamaño de los operandos para diferentes implementaciones de divisores publicadas recientemente y los circuitos descritos en esta sección.

La Figura 5.45 muestra la relación velocidad-latencia-área para los divisores de 32 bits presentados en esta sección. Se muestran los circuitos combinacionales (*array*), las versiones secuenciales (*sequential*) y dos de las arquitecturas segmentadas (*pipeSRTr2*, *pipeSRTr4*).

Como es de esperar, las implementaciones secuenciales son las que menos área requieren, mientras que las implementaciones segmentadas son las de menor retardo, con relativamente baja penalidad en área, pero a expensas de un retardo inicial. Por último la latencia mínima de las implementaciones combinacionales se logra a expensas de un alto costo en área.

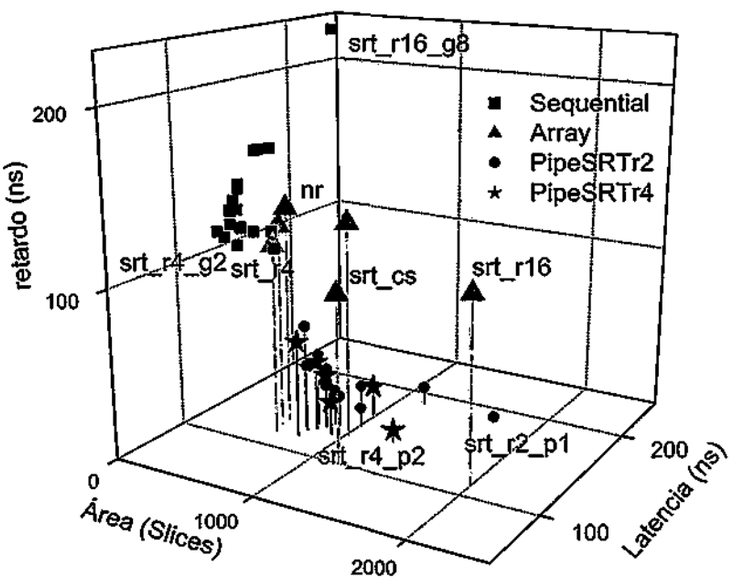


Figura 5.45. Relación retardo - latencia - área par las diferentes arquitecturas de divisores fraccionarios.

5.5.8 Resultados del consumo para divisores de números fraccionarios

El consumo fue medido en un dispositivo Virtex XCV800hq240-6, montada sobre una tarjeta de prototipo de Xilinx AFX PQ240-100 utilizando el arreglo experimental y metodología descritas en el Apéndice B. Los circuitos son descritos y implementados tal lo reseñado en la sección 5.4.7, es decir usando descripción VHDL instanciando cuando es necesario primitivas de bajo nivel. Se utilizó el entorno de desarrollo Xilinx ISE 6.1 [Xil03d] y XST (*Xilinx Synthesis Technologies*) para la síntesis. La misma asignación de patas, preservar la jerarquía, optimización por velocidad y restricciones de tiempo son parte de la estrategia de diseño.

Al igual que con la división números enteros, el consumo fue dividido en consumo estático, dinámico y *off-chip*. Se han utilizado para la medición de los circuitos tres secuencias diferentes (*avg_tog*, *max_tog* y *min_tog*), ingresadas a través de un generador de patrones [Tek04b]. Las salidas están conectadas a un analizador lógico [Tek04a].

5.5.8.1 Resultados en circuitos combinacionales

La Figura 5.47 representa el consumo de potencia expresado en mW/MHz para diferentes secuencias de vectores con divisores de 16 bits, en tanto la Figura 5.47 hace lo propio para operandos de 32 bits.

Para operandos de 16 bits, el divisor SRT base-2 muestra los mejores resultados, en promedio un 18,5 % menos que el divisor sin restauración (*non-restoring*), y 71 % respecto del divisor SRT con resto *carry-save*.

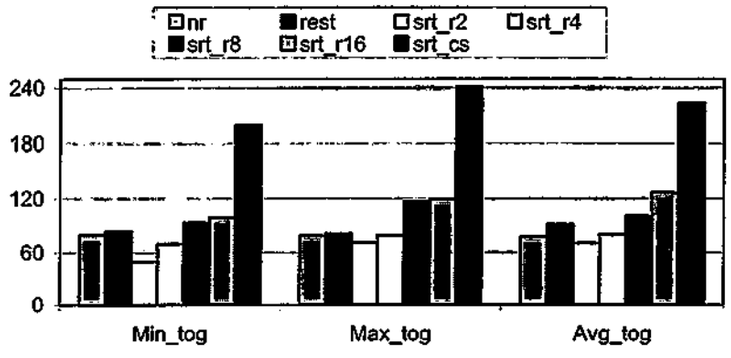


Figura 5.46. Consumo dinámico en mW/MHz para divisores de 16 bits.

Para el caso de los divisores de 32 bits SRT base-2 y SRT base-4 son las mejores opciones. SRT base-2 mejora hasta un 51,2 % respecto del divisor sin restauración y hasta un 78% respecto del divisor SRT con resto *carry-save*. Finalmente, se muestra la relación área-retardo-consumo en la Figura 5.48. Los divisores SRT base-2 (*srt_r2*), SRT base-4 (*srt_r4*), y sin restauración (*nr*) ofrecen la mejores figuras en ATP. Analizando la relación área × retardo × consumo, se observa al divisor SRT base-2 como la mejor opción seguida del SRT base-4.

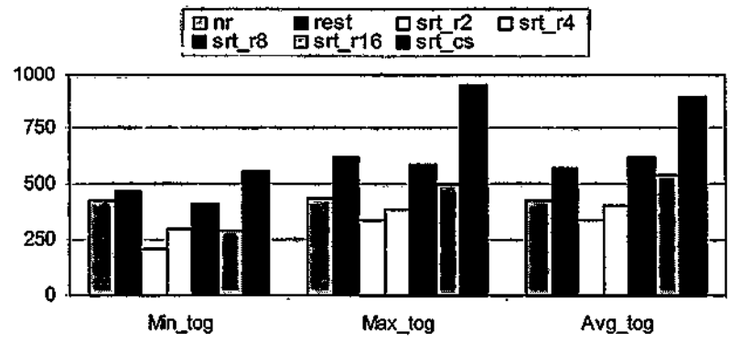


Figura 5.47. Consumo dinámico en mW/MHz para divisores de 32 bits.

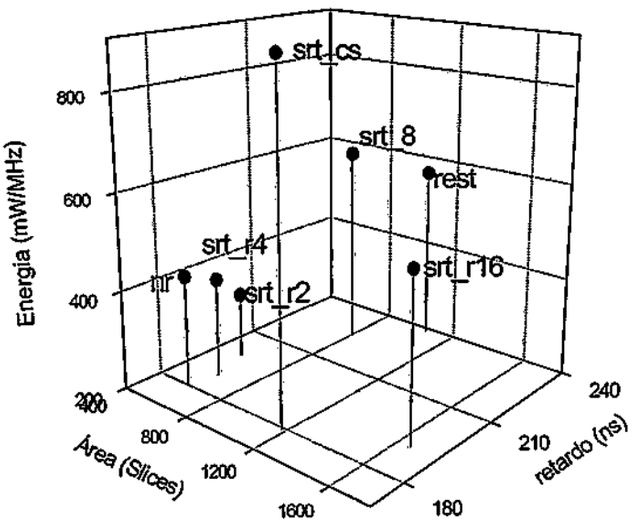


Figura 5.48. *Area-Time-Power* para divisores de 32 bits con la secuencia *avg_tog*.

5.5.8.2 Resultados en circuitos segmentados

Para medir y cuantificar la reducción de consumo debido a la segmentación en los circuitos divisores, se construyeron versiones segmentadas de los circuitos sin restauración (*nr*), SRT base-2 y base-4 con resto en complemento a dos (*srt_2*, *srt_r4*) y SRT base-2 con resto en carry-save (*srt_cs*). Los resultados en área y velocidad se muestran en la sección 5.4.7.2.

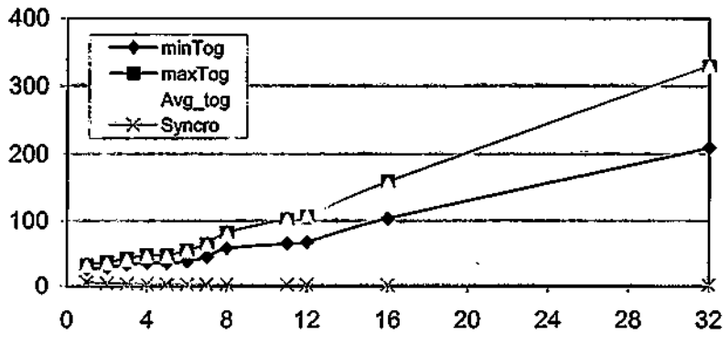


Figura 5.49. Consumo dinámico (mW/MHz) con respecto a la profundidad lógica para los divisores SRT base-2 con resto en complemento a dos.

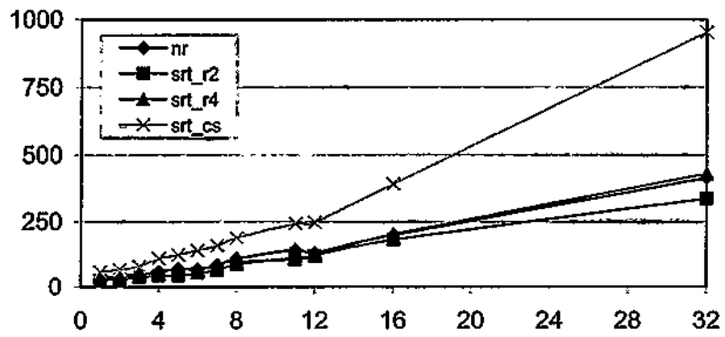


Figura 5.50. Consumo dinámico (mW/MHz) con respecto a la profundidad lógica para los diferentes divisores segmentados y la secuencia de prueba *avg_tog*.

La Figura 5.49 presenta el consumo dinámico con respecto a la profundidad lógica para la implementación de los divisores SRT base-2 con resto en complemento a dos. En la figura se puede apreciar que la relación del consumo respecto de la profundidad lógica es similar para los diferentes patrones de entrada: decrece prácticamente de forma lineal con la profundidad lógica (LD). Cabe destacar la baja influencia de la corriente de sincronización en el consumo dinámico.

Cuando más etapas de *pipeline* se agregan, menos *glitches* se producen, y el consumo se reduce. Esta reducción en la actividad hace menos importante la arquitectura elegida, de hecho las diferentes arquitecturas de divisores poseen similares figuras en el consumo respecto de la profundidad lógica, como lo muestra la Figura 5.50.

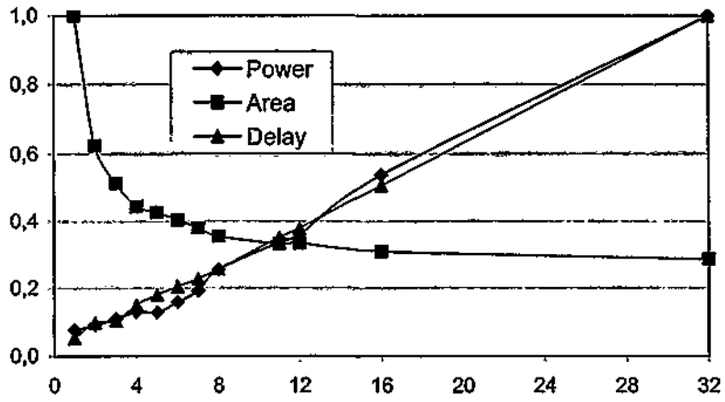


Figura 5.51. Consumo, retardo y área normalizados respecto de la profundidad lógica para el divisor SRT base-2 con resto en complemento a la base.

Una arquitectura totalmente segmentada (LD = 1) ahorra hasta un 93 % del consumo dinámico respeto de una arquitectura totalmente combinacional (LD = 32). Esto es, una arquitectura combinacional consume hasta doce veces más que una arquitectura totalmente segmentada.

Finalmente en la Figura 5.51 se puede ver el consumo, área y retardo normalizados respecto de la profundidad lógica (LD). Se destaca la linealidad del retardo así como del consumo respecto de la profundidad lógica y el relativamente baja penalidad en área de esta técnica.

5.5.8.3 Resultados en circuitos iterativos

Los resultados en el consumo de las implementaciones iterativas en Virtex se pueden ver en la Figura 5.52. La gráfica muestra el consumo energía promedio para llevar a cabo una operación expresado en nJoules para divisores de 32 bits. Se muestra por separado el consumo de sincronización y el debido a la ruta de datos. El consumo debido a la sincronización decrece a medida que *G* (bits calculados a la vez) crece, principalmente debido a la menor cantidad de ciclos necesarios. De manera opuesta, el consumo de la ruta de datos aumenta a medida que crece G debido al aumento de los *glitches*. El valor óptimo de G desde el punto de vista del consumo es 4 excepto para SRT con resto en *carry-save* (*srt_cs*).

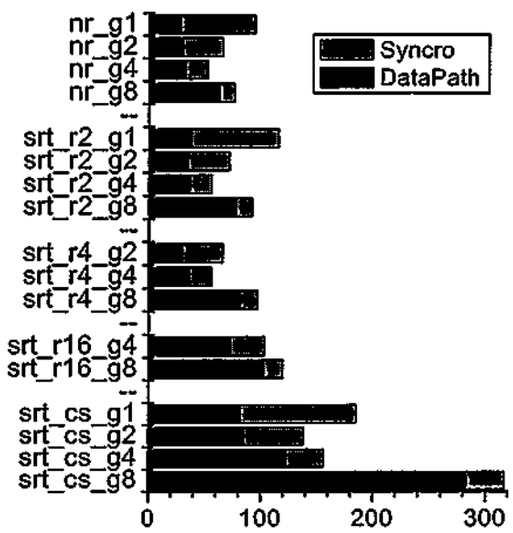


Figura 5.52. Consumo de energía tanto de sincronización como de la ruta de datos para los divisores secuenciales de 32 bits.

La división sin restauración con G=4 (*nr_g4*) muestra los consumos más bajos, así como SRT base-2 con G=4 (*srt_r2_g4*) y SRT base-4 con G=4 (*srt_r4_g4*), los que poseen consumos de energía similares.

Una observación importante es que el valor de G determina la figura del consumo con mayor importancia que el algoritmo aplicado. En SRT base-2 con G=4 existe un ahorro del 51 % de energía respecto a G=1. Por otra parte el ahorro de energía respecto de las implementaciones combinacionales son muy significativas, el circuito anterior consume un 85 % menos que SRT base-2, 89 % menos que el algoritmo sin restauración y 94 % respecto al SRT con resto en *carry save*.

5.5.8.4 Comparaciones arquitecturales en el consumo

La Figura 5.53 muestra área, retardo y consumo para algunos de los circuitos divisores de 32 bits implementados en esta sección. Se muestran los circuitos combinacionales (*array*), los circuitos segmentados SRT base-2 y base-4 para las todas profundidades lógicas (*pipe_SRT2* y *pipe_SRT4*) y las arquitecturas secuénciales (*iterative*).

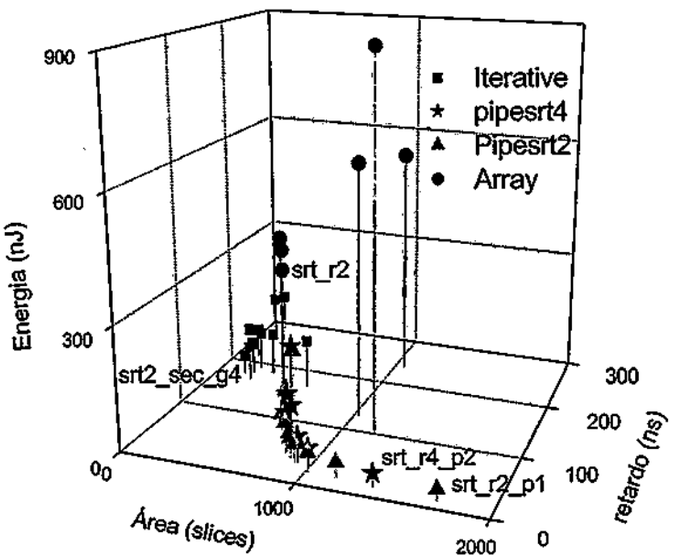


Figura 5.53. Área, retardo y consumo (ATP) para diferentes algoritmos división y arquitecturas secuenciales, combinacionales y segmentadas

Las arquitecturas combinacionales poseen la menor latencia a expensa de un mayor área y una excesiva disipación de potencia. Las arquitecturas segmentadas (*pipeline*) ofrecen el mayor caudal de trabajo (*throughput*) con un relativamente bajo incremento

en área respecto de la implementación combinacional y una buena figura de consumo, pero con una latencia inicial muy grande. Finalmente, las implementaciones secuenciales tienen un área mucho menor, un retardo que no duplica al de las arquitecturas secuenciales y un reducido consumo.

5.5.9 Conclusiones de la división de números fraccionarios

En esta sección se han presentado arquitecturas desarrolladas para la división de números fraccionarios (orientadas a la división de mantisas en punto flotante), analizado el consumo para diferentes formas de implementación.

Los circuitos fueron implementados en VHDL, instanciando componentes de bajo nivel cuando fue necesario. Implementaciones completamente combinacionales, con diferentes grados de segmentación y secuenciales con diferentes granularidades fueron construidos y medido el consumo.

Para implementaciones combinacionales el algoritmo SRT base-2 tiene la mejor figura en área, retardo y consumo, reduciendo este último hasta en un 51 % el consumo respecto del tradicional algoritmo de división sin restauración y un 93 % respecto del SRT base-2 con resto representado en *carry-save*.

Las arquitecturas segmentadas ofrecen características importantes para la reducción de consumo. Las medidas muestran una reducción de hasta un 93 % del consumo dinámico en una arquitectura totalmente segmentada, respecto de una totalmente combinacional. Esta mejora es obtenida con un relativamente pequeño impacto en el aumento de área.

Las implementaciones secuenciales usan menos recursos, con una relativamente pequeña penalidad en la latencia, pero una importante reducción en el consumo, llegando hasta un 89 % respecto a una implementación totalmente combinacional. Una observación importante en las implementaciones secuenciales es que, la cantidad de bits calculado por ciclos de reloj (G), es más determinante en el consumo que el algoritmo utilizado. Arquitecturas con G = 4 muestra el mejor consumo y mejor relación área × retardo × consumo, en tanto los circuitos con G = 2 optimizan la relación área × retardo. Los algoritmos sin-restauración y SRT base-2 obtienen los mejores resultados.

5.6 Conclusiones del Capítulo

En este capítulo se han examinado alternativas a nivel algorítmico para diferentes bloques aritméticos. Se han analizado la multiplicación modular, la suma de alta velocidad, la división entera y por último la división de números fraccionarios. Se pueden deducir algunos consejos y conclusiones generales derivados de los experimentos, los que se resumen a continuación:

- El nivel algorítmico es una interesante fuente de reducción de consumo, en los ejemplos descritos en esta sección queda claro que se trata de reducciones del orden del 50 % en el consumo debido a la correcta selección de un algoritmo.
- Las métricas de área y velocidad no son suficientes para calificar a priori la calidad del algoritmo respecto del consumo. Existen características más sutiles a tener en cuenta, como son la capacidad de producir o no movimientos espurios (*glitches*) dentro de la ruta de datos.
- Es sencillo encontrar ejemplos que contradicen la regla empírica "El circuito más rápido es el que menos consume" y también "el que menos área menos consume". A la hora de evaluar algoritmos para bloques aritméticos, la cuenta de transiciones en una simulación *post place & route* es más efectiva.
- La representación de los datos usados por el algoritmo, así como la correlación de datos que se producen dentro de la ruta de datos son muy relevantes. Por ejemplo en la división de enteros, el algoritmo *non-restoring* genera alternativamente restos intermedios muy pequeños positivos y negativos los que al ser representados en complemento a dos generan gran actividad.
- Existe un efecto directo sobre el consumo que poseen la complejidad y la precisión de los algoritmos. Un divisor de 32/32 bits consume alrededor de cuatro veces más que el homologo que divide 16/16 bits. Esto conduce a la recomendación de evitar sobredimensionar la ruta de datos.
- La regularidad y modularidad de los algoritmos son características altamente deseables para la segmentación y la secuencialización de los algoritmos, dos técnicas muy importantes de cara a la reducción de consumo.

- La aplicación de la segmentación ofrece importantes reducciones de consumo cuando se tienen grandes lotes de operaciones a realizar. La reducción depende del tamaño de la ruta de datos y los niveles de segmentación. Reducciones superiores al 80 % son corrientes en esta técnica.
- Las implementaciones secuenciales usan menos recursos, con una relativamente pequeña penalidad en la latencia. Es muy importante la regularidad del algoritmo para reducir la complejidad de la ruta de datos, así como la máquina de estados que la controla. Las reducciones de consumo pueden superar el 80 %.
- En las implementaciones secuenciales aumenta el consumo de sincronización, por ello la secuencialización óptima (cantidad de etapas por ciclo de reloj) desde el punto de vista del consumo depende del algoritmo. Por ejemplo para divisores de 32 bits, cuatro etapas de cálculo por ciclo de reloj ofrecen el mínimo consumo.

Además, producto de los diferentes experimentos llevados a cabo en este capítulo, surgen ciertas conclusiones particulares para cada bloque aritmético, las que se detallan a continuación:

De la multiplicación modular:

- Para calcular $e = y^x \bmod m$, donde m pertenece a un conjunto conocido de valores el algoritmo de Montgomery muestra la mejor figura de consumo (también área y retardo). Si solo se quiere calcular $z = x.y \bmod m$, la versión combinacional del algoritmo *m_r* (multiplicación y reducción) es mejor que la del algoritmo *s_d* (sumas y desplazamientos).
- Desde el punto de vista del consumo, se prueba que la elección del algoritmo correcto puede dar reducciones de consumo del orden del 50% en los casos combinacionales (Montgomery vs. Desplazar y sumar), del 54% en el caso secuencial (Montgomery vs. Multiplicar y reducir).
- Por otra parte la energía consumida para realizar la misma operación en las versiones secuenciales es menor que en las versiones combinacionales (hasta superar el 58% de reducción en el algoritmo de Montgomery), debido a la disminución de los *glitches* al reducir la profundidad lógica.

De la suma carry-skip:

- La técnica de sumador *carry skip* resulta muy interesante para ser aplicada en FPGAs para operandos largos. Una implementación cuidadosa, para sumadores con operandos de 1024 bits, puede aumentar en un factor de 6 la velocidad, con un incremento del área menor del 50% respecto a la implementación por defecto en FPGA.
- El mecanismo de cálculo del acarreo en el algoritmo *carry skip* genera menos *glitches* que el algoritmo *ripple carry* tradicional, producto de una menor propagación del acarreo y consecuentemente se puede prever menos consumo. El resultado en el consumo resulto ser peor para el *carry skip*. El error en el razonamiento parte del hecho de que el cálculo del camino crítico incluye la cadena de acarreo y el peor caso donde el acarreo debe propagarse desde la primera a la última etapa. En ese caso es cierto que el acarreo producirá gran actividad en la salida. Pero, probabilísticamente esto ocurre en muy pocos casos y la mayoría de las veces el acarreo se propaga por unas pocas etapas, por tanto los *glitches* por propagación del acarreo no son tan importantes en este ejemplo.

De la división entera:

- Los resultados en área y velocidad favorecen claramente a la división sin restauración (*nonrestoring*). El algoritmo con restauración (*restoring*) ocupa el doble de área y necesita alrededor del 50% más de tiempo que el algoritmo sin restauración.
- Sin embargo los resultados en el consumo son claramente favorables al algoritmo de división con restauración. Este efecto se refuerza cuando el tamaño del divisor es mayor a $n/2$ bits, siendo n el tamaño del dividendo.
- Para la división de números de 32 bits el algoritmo sin restauración consume ocho veces más que el algoritmo con restauración, en tanto el consumo medio para la división de 32 bits por 16 bits en *non-restoring* es solo alrededor del 20 % superior al algoritmo de división con restauración.

División de Números Fraccionarios

- Para implementaciones combinacionales el algoritmo SRT base-2 tiene la mejor figura en área-retardo y consumo, reduciendo este último hasta en un 51 % el

consumo respecto del tradicional algoritmo de división sin restauración y un 93 % respecto del SRT base-2 con resto representado en carry-save (el más veloz).

- Las arquitecturas segmentadas ofrecen características importantes para la reducción de consumo. Las medidas muestran una reducción de hasta un 93 % del consumo dinámico en una arquitectura totalmente segmentada, respecto de una totalmente combinacional. Esta mejora es obtenida con un relativamente pequeño impacto en el aumento de área.
- Las implementaciones secuenciales usan menos recursos, con una relativamente pequeña penalidad en la latencia, pero una importante reducción en el consumo, llegando hasta un 89% respecto a una implementación totalmente combinacional.
- Una observación importante en las implementaciones secuenciales es que, la cantidad de bits calculado por ciclos de reloj (G), es más determinante en el consumo que el algoritmo utilizado. Arquitecturas con G = 4 muestra el mejor consumo y mejor relación área × retardo × consumo, en tanto los circuitos con G = 2 optimizan la relación área × retardo. Los algoritmos sin-restauración y SRT base-2 obtienen los mejores resultados.

5.7 Referencias de Capítulo

[Beu02] J-L Beauchat and A. Tisserand, "Small Multiplier-Based Multiplication and Division Operators", *12th Conference on Field Programmable Logic and Applications*, pp. 513-522. September 2002.

[Bla99] I.Blake, G.Seroussi and N.Smart, "Elliptic Curves in Cryptography", *Cambridge University Press*, 1999.

[Blu99] T.Blum and C.Paar, "Montgomery Modular Exponentiation and Reconfigurable Hardware", *14th IEEE Symposium on Computer Arithmetic*, Adelaide, Australia. April 1999.

[Boe95] E. Boemo, G. Gonzalez de Rivera, S.Lopez-Buedo and J. Meneses, "Some Notes on Power Management on FPGAs", *Lecture Notes in Computer Science (LNCS), No.975*, pp.149-157. Berlin: Springer-Verlag 1995.

[Boe98] E. Boemo, S. Lopez-Buedo, C. Santos, J. Jauregui and J. Meneses, "Logic Depth and Power Consumption: A Comparative Study Between Standard Cells and FPGAs", *Proc. XIII DCIS Conference (Design of Circuit and Integrated Systems)*, Madrid, Univ. Carlos III: Nov 1998.

[Cha92] A. Chandrakasan, S. Sheng and R. Brodersen, "Low-Power CMOS Digital Design", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 4, pp. 473-484. April 1992.

[Des02] J-P. Deschamps, "Síntesis de Circuitos Digitales: un enfoque algorítmico", Thomson - Paraninfo, 2002.

[Erc04] Milos. D. Ercegovac and Tomas Lang, "Digital arithmetic", *Morgan Kaufmann, cop.* San Francisco, California. 2004.

[Erc94] M. D. Ercegovac and T. Lang. "Division and Square-Root Algorithms: Digit-Recurrence Algorithms and Implementations", *Kluwer Academic press*, 1994.

[Fis01] V.Fisher and M.Drutarovský, "Scalable RSA Processor in Reconfigurable Hardware - a SoC Building Block", *XVI Conference on Design of Circuits and Integrated Systems*, Porto, pp. 327 - 332. November 2001.

[Fpg01] FPGA Express page. Synopsis, inc.; www.synopsys.com/products/fpga /fpga_express.htm, 2001.

[Hau00] Scott Hauck, Member, IEEE, Matthew M. Hosler, and Thomas W. Fry, "high-Performance Carry Chains for FPGA's", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.8, No.2, April 2000.

[Kan98] V. Kantabutra, P. Corsonello, S. Perri and M. Iachino, "Efficient, Practical Adder for FPGA", *Circuit Cellar*, issue 148, pp.42 – 48, October 2002.

[Kor02] Israel Koren, "Computer Arithmetic Algorithms - second edition", *A.K.Peters*, 2002.

[Lee03] B.R. Lee and N. Burgess, "Improved Small Multiplier Based Multiplication, Squaring and Division" *11th Annual IEEE symposium on Field-Programmable Custom Computing Machines (FCCM2003)*. Napa, California, April 2003

[Lei95] J. Leiten, J. van Meerbegen and J. Jess, "Analisys and Reduction of Glitches in Synchronous Networks", *Proceedings of European Design and Test Conference. ED&TC 1995*, pp.1461-1464. New York: IEEE Press, 1995.

[Man01] D. Matilla, M. López-Vallejo and A. Rojo, "Hardware - Software Co-design of a Cryptographic Application", *XVI Conference on Design of Circuits and Integrated Systems (DCIS'01)*, Porto, Portugal, pp. 100 - 105. November 2001.

[Men96] A. Menezes, P. van Oorschot and S. Vanstone, "A Handbook of Applied Cryptography", *CRC Press*, 1996.

[Mon95] P. Montgomery, "Modular multiplication without trial division", *Mathematics of Computation*, vol.44, pp. 519 – 521. April 1895.

[Mus95] E. Mussol and J. Cortadella, "Low-Power Array Multiplier with Transition-Retaining Barriers", *Proceedings Fifth International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'95)*, pp. 227-235, Oldenburg, October 1995.

[Obe01] S.F.Oberman and M.Flynn, "Advanced Computer Arithmetic Design", *Wiley*, 2001.

[Obe97] S.F. Oberman and M.J. Flynn. "Division algorithms and implementations", *IEEE Transactions on Computers*, 46(8):833–854, August 1997.

[Par00] B. Parhami, "Computer Arithmetic, Algorithms and Hardware Designs", *Oxford University Press*, 2000

[Par03] Xilinx Inc, "Development System Reference Guide: chapter 10 PAR (Place & Route)", 2003.

[Ped96] M. Pedram, "Power Minimization in IC Design: Principles and Applications", *ACM Transaction on Design Automation of Electronic Systemi*, vol.1, n°1, pp.3-56, January 1996.

[Rab96] J.M.Rabaey, "Digital Integrated Circuits", *Prentice Hall*, 1996.

[Riv78] R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, vol.21, no2, pp.120-126, February 1978.

[Rob58] J. E. Robertson, "A New Class of Digital Division Methods", *IRE transaction on Electronic Computers*, Vol. 7, pp 218-222. September 1958.

[She92] A. Shen, A. Gosh, S. Devadas and K. Keutzer, "On average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks", *Proceeding of International Conference on Computer Aided Design (ICCAD-92)*, pp.402-407, IEEE Press, 1992.

[Sod96] P. Soderquist and M. Leeser. Area and Performance Tradeoffs in Floating-Point Divide and Square-Root Implementations, *ACM Computing Surveys*, Vol. 28, No. 3, September 1996.

[Sut03] Sutter G, Todorovich E, Lopez-Buedo S, and Boemo E. "Logic Depth, Power, and Pipeline Granularity: Updated Results on XC4K and Virtex FPGAs" *III Workshop on Reconfigurable Computing and Applications JCRA03*, Madrid, Spain, Sept 2003.

[Swe61] Sweeney, "High Speed Arithmetic in Binary Computers" *Proceedings of IRE*, Vol. 49, pp. 91-103, 1961.

[Tek04a] Tektronix inc, "TLA 700 Series Logic Analyzer User Manual", available at www.tektronix.com, 2004.

[Tek04b] Tektronix inc, "TLA7PG2 Pattern Generator Module User Manual", available at www.tektronix.com, 2004.

[Toc58] K. D. Tocher, "Techniques of multiplication and division for automatic binary computers", *Quarterly Journal of Mechanics and Applied Mathematics*. Vol. 11, pp. 364-384, 1958.

[Tod00] E. Todorovich, G. Sutter, N. Acosta, E. Boemo and S. López-Buedo, "End-user low-power alternatives at topological and physical levels. Some examples on FPGAs", *Proceedings of XV Conference on Design of Circuit and Integrated Systems (DCIS'2000)*. Montpellier, France, Nov. 2000.

[Wan03] X. Wang and B.E. Nelson, "Tradeoffs of Designing Floating Point Division and Square Root on Virtex FPGAs", *11th Annual IEEE symposium on Field-Programmable Custom Computing Machines (FCCM2003)*. Napa, California, USA. April 2003.

[Xil99] Xilinx Inc. "XC4000E and XC4000X Series Field Programmable Gate Arrays", Product Specification (Version 1.6), available at http://www.xilinx.com, May, 1999.

[Xil00] Xilinx Inc, "Logicore: Pipelined Divider V2.0" available at www.xilinx.com. June 30, 2000.

[Xil00b] Xilinx Inc, "Software Manuals and documentation for Foundation Series 3.1i." http://toolbox.xilinx.com/docsan/3_1i/, 2000

[Xil01] Xilinx, inc. "Spartan-II 2.5V FPGA Family: Functional Description"; available at http://www.xilinx.com, March 2001.

[Xil02a] Xilinx, inc. "Xilinx Synthesis Technology (XST) User Guide V2.21" Distributed with ISE 5.1, 2002, available at www.xilinx.com

[Xil02b] Xilinx, inc "Constraints Guide – ISE5.1 section 2-14, 2-15 Relative Location (RLOC) and Relationally Placed Macros (RPMs)", available at http://www.xilinx.com, 2002.

[Xil03a] Xilinx, inc. "Carry Logic in Virtex and Spartan-II", The Programmable Logic Data Book available on the Xilinx web site, http://support.xilinx.com, 2003

[Xil03b] Xilinx, inc. "Xilinx ISE 5.1 documentation: Online Software Manuals, available at support.xilinx.com", 2003.

[Xil03c] Xilinx Inc, "Libraries Guide for Xilinx ISE 6.1", available at www.xilinx.com, 2003.

[Xil03d] Xilinx Inc, "Xilinx ISE 6 Online Software Manuals", available at www.xilinx.com, 2003.

[Xil03e] Xilinx Inc, "XST (Xilinx Synthesis Technologies) User Guide 4.0", distributed with ISE 6.X, available at www.xilinx.com, June 2003.

[Xin98] S.Xing and W.W.H.Yu, "FPGA Adders: Performance Evaluation and Optimal Design", *IEEE Design & Test of Computers*, pp. 24 – 29, January - March 1998

[Xpo02] Xpower, "Xpower getting started", Release version 4.2.03i, 2002, available at support.xilinx.com.

[Xpo04] Xpower, "Xpower getting started" Release version 6.2.03i, 2004, available at support.xilinx.com.

Capítulo 6:

Conclusiones y Futuros Trabajos

Se resumen en este capítulo las principales conclusiones, aportes generados, los futuros trabajos que se desprenden de ésta tesis y las publicaciones generadas. Para finalizar, se seleccionan las principales recomendaciones de reducción de consumo a nivel del diseñador.

6.1 Conclusiones y Aportaciones

Se han presentado experimentos sobre varias generaciones de dispositivos del fabricante Xilinx (XC4000, Virtex y Virtex II). Estos dispositivos comparten la estructura básica de LUTs de 4 entradas, aunque tecnológicamente presentan una gran evolución. La mayor parte de los experimentos muestran que las técnicas propuestas son independientes del dispositivo utilizado, es decir, las conclusiones para una familia de dispositivos son aplicables a otras. Ésta tesis y las conclusiones que se detallan más adelante, se basa en medidas experimentales a más de 600 circuitos.

Las técnicas reseñadas durante esta tesis tratan de reducir el consumo siguiendo ciertos temas recurrentes como son: el balance área y velocidad para reducir consumo, evitar derroches, aprovechar la localidad de los datos y la reducción de la actividad espuria. Por otra parte, queda claro que las mayores reducciones de consumo se logran cuanto mayor sea el nivel de abstracción. Mientras a bajo nivel (nivel de puertas, rutado y emplazado y tecnológico) como mucho se puede lograr reducciones en un factor de dos, optimizaciones a nivel de arquitectura, algoritmo o sistema pueden llegar a reducir ordenes de magnitud en el consumo de potencia.

6.1.1 Recomendaciones para la reducción de consumo en FSMs

Los experimentos descritos en el capítulo 4, permiten generar la siguiente heurística para la minimización del consumo en máquinas de estados. La primera recomendación, aunque trivial, es llevar a cabo un diseño mínimo de la máquina de estados. Existen innumerables programas (mayoritariamente de libre distribución) para la minimización de estados, pero un diseño cuidadoso los hace prescindibles.

Una vez definida la máquina de estados y en función de la cantidad de estados existen diferentes alternativas. Si la máquina de estados es pequeña, no superando los ocho estados, las codificaciones binarias son la mejor alternativa. La codificación óptima depende de las transiciones más probables dentro de la FSM. No obstante, con transiciones con la misma probabilidad se demostró una clara correlación área-consumo, pudiéndose utilizar el área como métrica indirecta para la determinar la mejor codificación desde el punto de vista del consumo (sección 4.2).

Para máquinas de estado entre ocho y dieciséis estados no existe una regla clara respecto al tipo de codificación a utilizar. No obstante para más de dieciséis estados, *ONE-HOT* es mejor alternativa que las codificaciones binarias. Respecto de las máquinas grandes (más de dieciséis estados) las técnicas de particionamiento de máquinas de estado son una alternativa viable. La condición para que este método obtenga disminución en el consumo, es lograr realizar una partición de las máquinas de estados tal que la probabilidad de pasar de una máquina a otra sea pequeña (inferior al 5%). Para ejemplos concretos se han logrado disminuciones de hasta el 57% del consumo (sección 4.3).

6.1.2 Observaciones y consejos a nivel topológico

Durante el capítulo 3 se han presentado experimentos con el objeto de determinar relaciones en el consumo, y "consejos" a nivel usuario. Adicionalmente en el capítulo 5 dentro del estudio de bloques aritméticos se han analizado diversas alternativas topológicas. Las conclusiones más sobresalientes entres otros son:

Relación velocidad-consumo:

- Para una topología dada, el circuito con máxima frecuencia de funcionamiento normalmente implica el mejor circuito en término de

consumo. Esta optimización se puede obtener de manera gratuita, por ejemplo usando emplazado y rutado repetitivo (RPR - *Repetitivo Placement & Routing*) o ajustando las opciones de optimización. Siguen siendo válidos los experimentos de [Boe96] sobre XC3K. (sección 3.2.2.2)

- No obstante, si el diseñador debe elegir entre diferentes topologías, ni la velocidad de reloj, ni el área son parámetros por si solos validos para predecir el ahorro de consumo (sección 3.2.2.1).

- La relación entre área y consumo no es nada clara como sucede en los circuitos basados en celdas ó en el caso de las máquinas de estados. Algunas técnicas que ganan velocidad a expensas de mayor utilización de recursos (por ejemplo duplicar hardware para disminuir *fan-out*) son argumentos en este sentido (sección 3.2.2.4).

Efecto de la conmutatividad:

- Se prueba a través de diferentes ejemplos que bajo ciertas condiciones el hecho de permutar las entradas tiene un fuerte impacto en la reducción del consumo debido al cálculo. En la familia XC4K se utilizaron multiplicadores de 8 bits llegando la diferencia hasta un 8 %. En la familia Virtex con multiplicadores de 16 y 32 bits se llega a una diferencia de hasta un 39% (sección 3.4)

  Las causas de este desbalance en el consumo puede ser un diseño singular o la implementación irregular que se genera al mapear un diseño en una FPGA, esto produce que los *glitches* generados en una secuencia de operaciones no sean iguales al permutar las entradas. Otra causa es el uso de líneas globales para uno de los operandos, lo que puede generar diferencias en el consumo. Por ello, si la secuencia de valores a operar no es aleatoria o la frecuencia de uno de los operandos es diferente al otro (multiplicación de matrices, operaciones sobre video, filtros, etc.) conviene analizar el orden de los operandos. Cabe esperar que otros bloques combinacionales cuyas entradas sean conmutativas tiendan a tener esta característica de desbalance en el consumo. El uso de herramientas de estimación del consumo (y aun solo de la actividad) puede ser de utilidad a la hora de elegir el mejor orden de entrada de los operandos para estos casos.

Segmentación:

- La principal observación es que la segmentación disminuye notablemente el consumo, en el caso de la familia 4K una segmentación intermedia (2-4 LUTs de profundidad lógica) es lo ideal, en tanto que para la familias Virtex y Virtex II una segmentación cercana al máximo posible (1-2 LUTs) resulta ser la mejor opción.
- En las FPGAs, gracias a los registros distribuidos por todo el dispositivo, la aplicación de la segmentación no suele degradar demasiado el área, en tanto que, el consumo producto de la reducción de los movimientos espurios (*glitches)* se mejora ostensiblemente (sección 3.4).
- En las familias Virtex y Virtex II se ha observado la relación directa que posee la profundidad lógica con la degradación de velocidad y el consumo. De utilizar una versión totalmente combinacional a segmentar de manera óptima (para estos circuitos - profundidad lógica 2) se puede reducir el consumo dinámico hasta un 12% del original en la familia Virtex y a un 24 % en la familia Virtex II. (sección 3.4.2 y 3.4.3)

Registros a la Entrada y Salida:

- La existencia de registros en los datos de salida tiene un fuerte impacto en el consumo, las capacidades internas del chip son del orden de los femto-faradios en tanto que los PCBs se pueden medir en pico-faradios, con la consiguiente implicación en el consumo de los *glitches.* Para los ejemplos medidos en la familia 4K existe un aumento del 21 % quitando los registros, en tanto que en Virtex del orden del 90 % y en el dispositivo Virtex II estudiando supera el 110 %. (Sección 3.5)
- En el caso de la elección del tipo de *flip-flop* para registrar las salidas entre los disponibles en los *slices*/CLBs y los de los IOBs existe una gran diferencia desde el punto de vista del consumo. En el caso de la familia XC4K para multiplicadores de 8 bits puede haber una diferencia del orden del 14 %, en tanto que en Virtex para multiplicadores de 32 bits y teniendo en cuenta el efecto que produce la diferencia de tensión de alimentación del *core* y la periferia, puede llegar a ser un 40 % superior. Para Virtex II la influencia es algo menor, en torno al 9 %. (Sección 3.5)

Implementaciones secuenciales:

- La secuencialización de los algoritmos se asocia al ahorro de área. En FPGAs esta afirmación no solo es verdad, sino que además la penalidad en la latencia suele ser bastante reducida (en el orden del 20 %). Es muy importante la regularidad del algoritmo para reducir la complejidad de la ruta de datos y la máquina de estados. Las reducciones de consumo pueden superar el 80 % (sección 5.5.8.3).

- En las implementaciones secuenciales aumenta el consumo de sincronización. Por ello, la secuencialización óptima (cantidad de etapas por ciclo de reloj) desde el punto de vista del consumo depende del algoritmo. Por ejemplo, para divisores de 32 bits, cuatro etapas de cálculo por ciclo de reloj ofrecen el mínimo consumo (sección 5.5.8.3).

Inhabilitación de partes inactivas del circuito:

- Dejar que partes inactivas del circuito generen consumo, es claramente un derroche que se debe evitar. Las técnicas de inhabilitación descritas en las secciones 3.6.2 son fácilmente aplicables en FPGAs.

- La técnica más simple de implementar y que mejores resultados ofrece tanto en XC4K como en Virtex y Virtex II es la inhabilitación de registros con la señal CE. La opción de utilizar puertas ANDs es otra alternativa simple de utilizar, que además reporta buenos resultados.

- La técnica de *gated clock* en Virtex II ofrece buenos resultados en la reducción de consumo utilizando el componente específico BUFGMUX, aunque la penalidad en el retardo no es despreciable. Esta técnica gana relevancia cuando se tiene largos periodos de inactividad, ya que elimina también el consumo de sincronización (sección 3.6.2.1).

- La alternativa de los *buffers* de tercer estado, no solo degrada el camino crítico fuertemente, sino que ofrece los peores resultados en cuanto al ahorro de consumo (sección 3.6.2.4).

- Debido a la gran cantidad de registros distribuidos en la FPGA, la implementación de estas técnicas de inhabilitación poseen un escaso (a veces nulo) impacto en el área y velocidad (sección 3.6.3).

6.1.3 Resultados a nivel algorítmico

Durante el capítulo 5 se examinaron algunas alternativas a nivel algorítmico. Por un lado se examinan las opciones para la multiplicación modular, operación central en los cálculos criptográficos; luego se presentó un experimento cuyos resultados no fueron los esperados desde el punto de vista del consumo (la suma por el algoritmo de *carry-skip*). Por último se analizó la operación de división, tanto para operandos naturales como para fraccionarios.

- El nivel algorítmico es una interesante fuente de reducción de consumo, en los ejemplos estudiados en el capítulo 5 queda claro que se trata de reducciones del orden del 50 % en el consumo debido a la correcta selección de un algoritmo.
- Las métricas de área y velocidad no son suficientes para calificar la calidad del algoritmo respecto del consumo. Existen características más sutiles a tener en cuenta, como son la capacidad de producir o no movimientos espurios (*glitches*) dentro de la ruta de datos (sección 5.4.3.2).
- La representación de los datos usados por el algoritmo, así como la correlación de datos que se producen dentro de la ruta de datos son muy relevantes. Por ejemplo en la división de enteros, el algoritmo *non-restoring* genera alternativamente restos intermedios muy pequeños positivos y negativos los que al ser representados en complemento a dos generan gran actividad. (sección 5.4.3.2).
- Existe un efecto directo sobre el consumo que poseen la complejidad y la precisión de los algoritmos. Un divisor de 32/32 bits consume alrededor de cuatro veces más que el homologo que divide 16/16 bits. (sección 5.5.8.1).
- La regularidad y modularidad de los algoritmos son características altamente deseables para la segmentación y la secuencialización de los algoritmos, dos técnicas muy importantes de cara a la reducción de consumo.
- La aplicación de la segmentación sobre los algoritmos ofrece importantes reducciones de consumo cuando se tienen un flujo continuo de datos. La reducción depende del tamaño de la ruta de datos y los niveles de

segmentación. Reducciones superiores al 80 % son corrientes en esta técnica. (sección 5.5.8.2).

- Las implementaciones secuenciales usan menos recursos, con una relativamente pequeña penalidad en la latencia. Es muy importante la regularidad del algoritmo para reducir la complejidad de la ruta de datos y la máquina de estados. Las reducciones de consumo pueden superar el 80 %. (sección 5.5.8.3).
- En las implementaciones secuenciales aumenta el consumo de sincronización, por ello la secuencialización óptima (cantidad de etapas por ciclo de reloj) desde el punto de vista del consumo depende del algoritmo y la complejidad de la ruta de datos. (sección 5.5.8.3).

6.1.4 Bloques aritméticos

Los resultados del capítulo 5 brindan ciertas conclusiones particulares para cada bloque aritmético, las que se detallan a continuación:

Multiplicación modular:

- Para calcular $e = y^x \bmod m$, donde m pertenece a un conjunto conocido de valores, el algoritmo de Montgomery muestra la mejor figura de consumo (también área y retardo). Si sólo se quiere calcular $z = x.y \bmod m$, la versión combinacional del algoritmo de multiplicación y reducción es mejor que la del algoritmo de sumas y desplazamientos (sección 5.2.3.4).
- Desde el punto de vista del consumo, se prueba que la elección del algoritmo correcto puede dar reducciones de consumo del orden del 50% en los casos combinacionales, del 54% en el caso secuencial (sección 5.2.3.5).
- Por otra parte la energía consumida para realizar la misma operación en las versiones secuenciales es menor que en las versiones combinacionales (hasta superar el 58% de reducción en el algoritmo de Montgomery), debido a la disminución de los *glitches* al reducir la profundidad lógica (sección 5.2.5.2).

Suma carry-skip:

- La técnica de sumador *carry skip* resulta muy interesante para ser aplicada en FPGAs para operandos largos (a partir de 64 bits). Una implementación

cuidadosa, para sumadores de 1024 bits, puede superar el 500 % de aumento en la velocidad, con un incremento del área menor del 50 % respecto a la implementación por defecto en FPGA. Los resultados en el consumo, sin embargo son mejores para el *ripple carry* tradicional (sección 5.3).

División entera:

- Los resultados en área y velocidad favorecen claramente a la división sin restauración (*nonrestoring*). El algoritmo con restauración (*restoring*) ocupa el doble de área y necesita alrededor del 50 % más de tiempo que el algoritmo sin restauración. Sin embargo los resultados en el consumo son claramente favorables al algoritmo de división con restauración. Este efecto se refuerza cuando el tamaño del divisor es mayor a $n/2$ bits, siendo n el tamaño del dividendo (sección 5.4).
- Para la división de números de 32 bits el algoritmo sin restauración consume más de ocho veces que el algoritmo con restauración, en tanto el consumo medio para la división de 32 bits por 16 bits en *non-restoring* es sólo alrededor del 20 % superior al algoritmo de división con restauración (sección 5.4.3.1).

División de Números Fraccionarios

- Para implementaciones combinacionales, el algoritmo SRT base-2 tiene la mejor figura en área-retardo y consumo, reduciendo este último hasta en un 51 % el consumo respecto del tradicional algoritmo de división sin restauración y un 93 % respecto del SRT base-2 con resto representado en *carry-save* (el más veloz) (sección 5.5.7.1 y 5.5.8.1).
- Las arquitecturas segmentadas ofrecen características importantes para la reducción de consumo. Las medidas muestran una reducción de hasta un 93 % del consumo dinámico en una arquitectura totalmente segmentada, respecto de una totalmente combinacional. Esta mejora es obtenida a expensas de triplicar el área (sección 5.5.8.2).
- Las implementaciones secuenciales usan menos recursos, con una relativamente pequeña penalidad en la latencia, pero una importante reducción en el consumo, llegando hasta un 89% respecto a una implementación totalmente combinacional (sección 5.5.8.3).

- Una observación importante en las implementaciones secuenciales es que, la cantidad de bits calculado por ciclos de reloj (G), es más determinante en el consumo que el algoritmo utilizado. Arquitecturas con G = 4 muestra el mejor consumo y mejor relación área × retardo × consumo, en tanto los circuitos con G = 2 optimizan la relación área × retardo. Los algoritmos sin-restauración y SRT base-2 obtienen los mejores resultados (sección 5.5.8.3).

6.1.5 Conclusiones sobre la herramientas de estimación de consumo

Paralelamente a las mediciones en los arreglos experimentales, se ha realizado la estimación del consumo con la herramienta XPOWER. Ante todo, hay que destacar que es un proceso lento, que requiere varios gigabytes de ficheros intermedios, que implica una simulación *post-place & route* y un posterior análisis con la herramienta. Esto lo convierte en muchos casos en un método impráctico. El apéndice F muestra la forma de utilización de la herramienta.

Los resultados observados para circuitos secuenciales son bastante aceptables (secciones 3.3.4.2 y 5.3.3.1), coincidiendo las arquitecturas que más consumen con la estimación, aunque no en un porcentaje constante. Para el caso del análisis de la segmentación (sección 3.4.2.2) los resultados fueron bastante decepcionantes, no coincidiendo las figuras del consumo medido con el estimado.

En el flujo de diseño normal se debería poder generar automáticamente la actividad del circuito, ahorrándose horas de simulación, gigabytes de datos intermedios y búsqueda de patrones de entrada significativos (En la línea de la investigación de [Tod04]). No obstante, resulta interesante esta herramienta, no para la estimación de un sistema completo, sino para la caracterización de bloques aritméticos particulares, donde tanto el tiempo de simulación como la generación de los vectores de prueba podrían ser acotados.

6.2 Trabajos futuros

Como se ha mencionado anteriormente, el objetivo general de este trabajo es dar soluciones en el diseño de bajo consumo en FPGAs. Dado que ésta es una meta extremadamente ambiciosa, se ha reducido la cantidad de experimentos y evaluación de técnicas llevadas a cabo. La ampliación de estos experimentos y estudios implica posiblemente varios temas de tesis. Las líneas de trabajos que se desprenden incluirían entre otras:

Estudio de bloques aritméticos complejos. Si bien las dos operaciones fundamentales en el procesamiento de señal (suma y multiplicación) están muy estudiadas desde el punto de vista del consumo (la división se ha estudiado aquí), sería útil estudiar otras primitivas de gran aplicación como son la exponenciación y la raíz cuadrada. Otro aspecto importante es el procesamiento digital de la señal y las múltiples variables que afectan al consumo en filtros, transformadas y otros bloques de procesamiento. También el tratamiento con operandos grandes (más de 128 bits) por sus aplicaciones en criptografía y de las operaciones en punto flotante merecen gran interés.

Generación de cores de bajo consumo. Este punto está íntimamente ligado con el anterior, aunque con matices propios. Los *cores* facilitados por los fabricantes de silicio y otros proveedores independientes de IPs (*intelectual properties*) aún no proveen características de bajo consumo. Es por tanto útil generar *cores* parametrizables optimizados en consumo de las operaciones más frecuentes y demandadas. Además sería deseable que se provea la caracterización de la disipación de potencia, de modo de permitir estimar a nivel de bloques el consumo total de una aplicación.

Estudio de microprocesadores (μP) embebidos. Cada vez más la lógica programable tiende a solucionar sistemas completos (*SoC - System on a Chip*) donde la integración de un microprocesador es esencial. Esta característica genera dos áreas importantes de estudio, por un lado la generación de *cores* de microprocesadores de bajo consumo y por otro compiladores que generan código de bajo consumo para estos μP.

Taxonomía de las técnicas de reducción de consumo en FPGAs. Como resultado final se debería generar una taxonomía general de las técnicas para reducir el consumo, donde el diseñador pueda consultar, dependiendo del dominio de

aplicación, que técnicas se deberían aplicar para lograr optimizaciones de consumo. Varias alternativas se han evaluado aquí aunque se debería ampliar este estudio a técnicas de precomputación, representación y codificación de los datos y programación (*scheduling*) para el manejo dinámico del consumo.

Herramientas de síntesis para bajo consumo. Los sintetizadores tradicionalmente han optimizado en área y velocidad. La síntesis de bajo consumo está presente en herramientas para ASICs pero no en FPGAs. En base a algunos resultados obtenidos aquí y otros publicados recientemente se podría incorporar características de síntesis para bajo consumo.

Herramientas de estimación de alto nivel. Las herramientas de estimación de consumo actuales parten de la descripción del circuito y una simulación *post-layout* del mismo. Al margen de su relativa precisión, son por cierto lentas de ejecutar, sería útil contar con versiones que permitan estimar el consumo a un mayor nivel de abstracción.

Estudio de la interacción con bloques embebidos. Los dispositivos actuales integran bloques de memoria, multiplicadores y microprocesadores con una clara tendencia a incrementarse la cantidad y variedad de bloques específicos. Resulta de interés investigar sus características de consumo y más aún desarrollar métodos para integrarlos en diseños de bajo consumo. Es decir estudiar cuando y de que forma conviene utilizarlos.

6.3 Publicaciones relacionadas con éste trabajo

Se han generado artículos relativos a partes de este trabajo en diferentes congresos y medios de divulgación sobre FPGAs y diseño electrónico. Las publicaciones más relevantes son las siguientes:

- G. Sutter, G.Bioul, J-P. Deschamps, and E.Boemo "Power Aware Dividers in FPGA", *Lecture Notes in Computer Science (LNCS)*, Vol. 3254, pp. 574 - 584. Berlin: Springer-Verlag, 2004.
- G. Sutter, G.Bioul, and J-P. Deschamps, "Comparative Study of SRT-Dividers in FPGA", *Lecture Notes in Computer Science (LNCS)*, Vol.3203, pp. 209 - 220. Berlin: Springer-Verlag, 2004.

- G. Sutter, E. Todorovich, S. Lopez-Buedo, and E. Boemo, "Low-Power FSMs in FPGA: Encoding Alternatives", *Lecture Notes in Computer Science*, Vol.2451, pp.363-370, Berlin: Springer-Verlag, 2002.
- E. Todorovich, M. Gilabert, G. Sutter, S. Lopez-Buedo, and E. Boemo, "A Tool for Activity Estimation in FPGAs", *Lecture Notes in Computer Science*, Vol.2438, pp.340-349. Berlin: Springer-Verlag, 2002.
- G. Sutter, E. Todorvich, S. Lopez- Buedo, and E. Boemo, "FSM Decomposition for Low Power in FPGA", *Lecture Notes in Computer Science*, Vol.2438, pp.350-359. Berlin: Springer-Verlag, 2002.

Además, se han presentado otros resultados parciales en:

- G. Sutter, E. Todorovich, E. Boemo, "Design of Power Aware FPGA-based Systems", *IV JCRA*, pp 81-90, Barcelona, Spain, September 2004.
- J-P. Deschamps and G. Sutter, "Multiplication in a Finite Extension Ring", *XVIII Conference on Design of Circuits and Integrated Systems (DCIS 2003)*, Ciudad Real, Spain, November, 2003.
- E. Boemo and G. Sutter, "Permutación de los Datos de Entradas como Estrategia de Diseño de Bajo Consumo: Algunos Ejemplos en FPGAs", *III JCRA 03*, pp 225- 232, Madrid, Spain, September 2003.
- Sutter G., Todorovich E, López-Buedo S, and Boemo E., "Logic Depth, Power and Pipeline Granularity: Updated Results on XC4K and Virtex FPGAs", *III JCRA*, pp 201-207, Madrid, Spain, September 2003.
- Bioul G, Deschamps J-P. Sutter G, "Efficient FPGA implementation of Carry Skip Adders", *Jornadas de Computación Reconfigurable y Aplicaciones*, pp 81-90. Madrid, Spain, September 2003.
- E. Todorovich, G. Sutter and E. Boemo, "Estimación de Actividad para FPGA Basada en una Técnica Estadística", *III Jornadas de Computación Reconfigurable y Aplicaciones*, pp 217-224, Madrid, Spain, September 2003.
- J-P. Deschamps and G. Sutter , "FPGA Implementation of Modular Multipliers", *XVII Conference on Design of Circuits and Integrated Systems (DCIS 2002)*, pp. 107-112 (ISBN 84-8102-311-6). Santander, November 19-22, 2002

- G. Sutter and J-P. Deschamps, "Multiplicadores modulares", *II Jornadas sobre Computación Reconfigurable y Aplicaciones (JCRA 2002)*, Almuñecar (Granada), Spain, pp. 209-214 (ISBN 84-699-9448-4), sept de 2002.
- G. Sutter and E. Boemo "Low Power Finite state machines in FPGA: Bynary vs One hot encoding" *VIII Workshop IBERCHIP*, Guadalajara, México, 3-5 April 2002.
- G. Sutter, E. Todorovich and E. Boemo. "Metodología para la Reducción de Consumo en Circuitos Integrados Reprogramables" *III Workshop de Investigadores de Ciencias de la Computación (WICC 2001)*, San Luis, Argentina, Agosto 2001.
- E. Todorovich, G. Sutter, N. Acosta, E. Boemo and S. López-Buedo "Relación entre Velocidad y Consumo en FPGAs" *VII Workshop de IBERCHIP*, Montevideo, Uruguay, 21-23 marzo 2001.
- G. Sutter, E. Todorovich, E. Boemo and S. López-Buedo "Propiedad Conmutativa y Diseño de Bajo Consumo: Algunos Ejemplos en FPGAs", *VII Workshop de IBERCHIP*, Montevideo, Uruguay, 21-23 marzo 2001.
- E. Todorovich, G. Sutter, N. Acosta, E. Boemo and S. López-Buedo "End-user low-power alternatives at topological and physical levels. Some examples on FPGAs", *XV Conference on Design of Circuits and Integrated Systems (DCIS2000)*, Le Corum, Montpellier, France, November 21-24, 2000.

6.4 Reglas empíricas para el diseño de bajo consumo en FPGAs

A modo de resumen final de los resultados de esta tesis se presentan estas seis simples reglas empíricas que debería seguir un diseñador de sistemas de bajo consumo basados en tecnología FPGA. Aquí se seleccionan aquellas reglas de aplicación inmediata por parte del diseñador.

1. No sobredimensionar la capacidad de cálculo así como la ruta de datos. Operar a más frecuencia de la necesaria implica un crecimiento lineal con el consumo, además exigir gran capacidad de cálculo puede implicar desechar arquitecturas secuenciales u otras técnicas más afines a la reducción de consumo. Por otro lado el tamaño de la ruta de datos puede implicar aumentos con órdenes mayores que dos en el consumo. Por ejemplo la multiplicación de 32*32 bits consume al menos cuatro veces más que uno de 16*16, lo mismo sucede para la división de números fraccionarios.

2. Reducir la profundidad lógica a un máximo de 2-3 LUTs en circuito de gran actividad. Los glitches son la principal fuente en el consumo, la manera más directa de reducirlo en FPGAs, es a través de la segmentación en caso de tener lotes de datos que procesar o bien sequencializando la ruta de datos. Para aplicar efectivamente estas técnicas la regularidad de la ruta de datos es fundamental.

2.a. Uso de la segmentación: Es la técnica más adecuada cuando se trabaja con grandes lotes de datos. La reducción del consumo suele ser incluso proporcional al aumento de velocidad del algoritmo. La penalidad en área suele ser despreciable dada la abundancia de registros distribuidos en la FPGA.

2.b. Uso de arquitecturas secuenciales: La secuencialización de los algoritmos se asocia al ahorro de área. En FPGAs esta afirmación no solo es verdad, sino que además la penalidad en la latencia suele ser bastante reducida. En las implementaciones secuénciales aumenta el consumo de sincronización, por ello la secuencialización óptima (cantidad de etapas por ciclo de reloj) desde el punto de vista del consumo depende del algoritmo.

3. Desactivar partes inactivas del diseño. En sistemas complejos, partes del diseño suelen no estar produciendo datos efectivos. Desactivar esas partes es fundamental de

cara a evitar el derroche de consumo. Utilizar las señales de habilitación CE (*chip-enable*) de los registros para frenar los datos ó el uso de puertas ANDs brinda importantes posibilidades de ahorro en el consumo, con impacto casi nulo en área y velocidad. En las familias de FPGAs más modernas (Virtex II, spartan 3, etc.) los múltiples árboles de reloj y los multiplexores de reloj permiten implementar fácilmente técnicas tipo *gated clock*.

4. Explorar y explotar el nivel algorítmico. El diseño de los algoritmos posee una de las fuentes más directas de ahorro de energía. La regularidad y modularidad de los algoritmos son características altamente deseables para la segmentación y la secuencialización de los algoritmos, dos técnicas muy importantes de cara a la reducción de consumo.

Definitivamente las métricas de área y velocidad no son suficientes para calificar la calidad del algoritmo respecto del consumo. Existen características más sutiles a tener en cuenta, como son la capacidad de producir o no movimientos espurios (*glitches*) dentro de la ruta de datos, esta en general producida por la correlación de datos que se generan.

5. Registrar siempre que sea posible. Es importante registrar siempre que sea posible tanto las entradas como las salidas. Más aun en las señales de salida es conveniente registrar "cerca" de la lógica que produce el dato de salida usando FF en los *slices* en vez de los registros asociados a las patas. No sólo se evita la activad en las líneas que conectan la lógica con los IOBs, sino que sobre todo se evita la actividad en los *buffers* que transforman la señal de la tensión del *core* a la de las patas. Cabe recordar que la alimentación del *core* de los dispositivos modernos es menor que en la periferia.

6. En el diseño de máquinas de estados, cuidar la codificación y eventualmente particionar. Para máquinas de estados pequeñas hasta 8 estados utilizar codificaciones densas (binarias). Para máquinas con más de 15 estados utilizar *one-hot*. Eventualmente para máquina de estados grandes (más de 12 estados) cuando se pueda realizar una partición con poca probabilidad de cambios entre ellas utilizar partición de máquinas de estados. Como caso especial de máquinas de estados, los contadores con codificaciones tipo gray son una buena opción de bajo consumo, siempre que no se deba utilizar decodificaciones extras de las salidas.

Apéndice A:

Placa de prueba familia XC4K

A.1 Introducción

Durante el desarrollo de esta tesis se ha utilizado una placa de prueba para dar soporte a distintos circuitos que se han desarrollando. Se ha utilizado una placa desarrollada por S.Lopez-Buedo [Lop97]. La correcta funcionalidad de los circuitos fue comprobada mediante el uso de un analizador lógico [Tek00], y adicionalmente se mide el consumo indirectamente mediante la utilización de voltímetro y amperímetro.

A.2 Características de la placa de prueba

Las dos principales características de diseño son por un lado que esta dotada de conectores especialmente dedicados para acoplar el analizador lógico, y por otro, la inclusión de una segunda FPGA para ser utilizada como generador de vectores de test para los circuitos que se analizan.

La placa se componen de un generador de vectores de prueba, implementado en una FPGA del tipo Xilinx XC3K (se utilizaron concretamente los dispositivos Xilinx XC3120PC84-3 y XC3130PC84-5). Tras ésta se sitúa un primer banco de 24 conectores más uno de reloj, cuyo objetivo es poder monitorear estos vectores de prueba, permitiendo el acoplamiento directo de un los conectores del analizador lógico usado, el Tektronix TLA704 [Tek00]. Éstos pasan a continuación a la FPGA de prueba, una Xilinx XC4000 en encapsulado PLCC84. Por último, un segundo banco de conectores igual al primero permite observar los resultados.

La configuración de las FPGAs de prueba se hace mediante cable de *download* (en particular se utiliza el Xilinx Xchecker [XCH99]), mientras que la FPGA generadora de vectores de prueba se configura mediante una EPROM serie (Atmel AT17C128). La diferencia en el modo de programación estriba en la diferente cantidad de veces que se han de reprogramar cada FPGA. Por un lado es de esperar que, la generación de patrones de *test* se reconfiguren una única vez (o unas pocas) para cada experimento. Por el contrario, la FPGA de prueba va a necesitar ser reconfigurada en gran cantidad de ocasiones.

Pod	XC3120	XC4000	Pod	XC4000
D0_7	25	18	O2_7	25
D0_6	26	17	O2_6	24
D0_5	27	16	O2_5	27
D0_4	28	15	O2_4	26
D0_3	29	14	O2_3	29
D0_2	30	10	O2_2	28
D0_1	34	9	O2_1	36
D0_0	35	8	O2_0	35
D1_7	36	7	O2_7	37
D1_6	37	6	O2_6	38
D1_5	39	5	O2_5	39
D1_4	40	4	O2_4	40
D1_3	44	3	O2_3	44
D1_2	45	83	O2_2	45
D1_1	46	84	O2_1	46
D1_0	47	81	O2_0	47
D2_7	48	82	O2_7	48
D2_6	49	79	O2_6	49
D2_5	52	80	O2_5	50
D2_4	53	77	O2_4	51
D2_3	56	78	O2_3	56
D2_2	57	70	O2_2	57
D2_1	58	69	O2_1	58
D2_0	59	68	O2_0	59

Tabla A.1. Conexiones del analizador en la placas de prueba.

En la Tabla A.1 se muestran los *pines* de conexión al analizador lógico, la tabla A.2 muestra la asignación de *pines* para el cable de programación (*download*) del dispositivo XC4K y la asignación de botones y conmutadores de la XC3K.

Conmutadores y Botones		Conector de *download*	
Elemento	Pin XC3000	Pin conector	
switch 1	83	1	VCC
switch 2	84	2	GND
switch 3	81	3	NC
switch 4	82	4	CCLK
		5	DONE
Botón 1	75	6	DIN
Botón 2	76	7	_PROG
Botón Reset	Rst_fpga	8	_INIT
		9	NC

Tabla A.2. Conexiones del cable de programación y botones

A.3 Dispositivos utilizados

Se han utilizado en este arreglo experimental 3 dispositivos de la familia XC4K. Ellos son XC4003PC84-4, XC4005PC84-4 y una XC4010PC84-4. El resumen de los recursos de cada dispositivo se puede ver en la tabla A.3.

Dispositivo	Puertas equivalente	Matriz CLBs	Total CLBs	Cantidad de *Flip-Flops*	Máxima cantidad E/S
XC4003	3000	10 x 10	100	360	80
XC4005	5000	14 x 14	196	616	112
XC4010	10000	20 x 20	400	1120	160

Tabla A.3. Recursos de los dispositivos de la serie XC4K utilizados en las mediciones

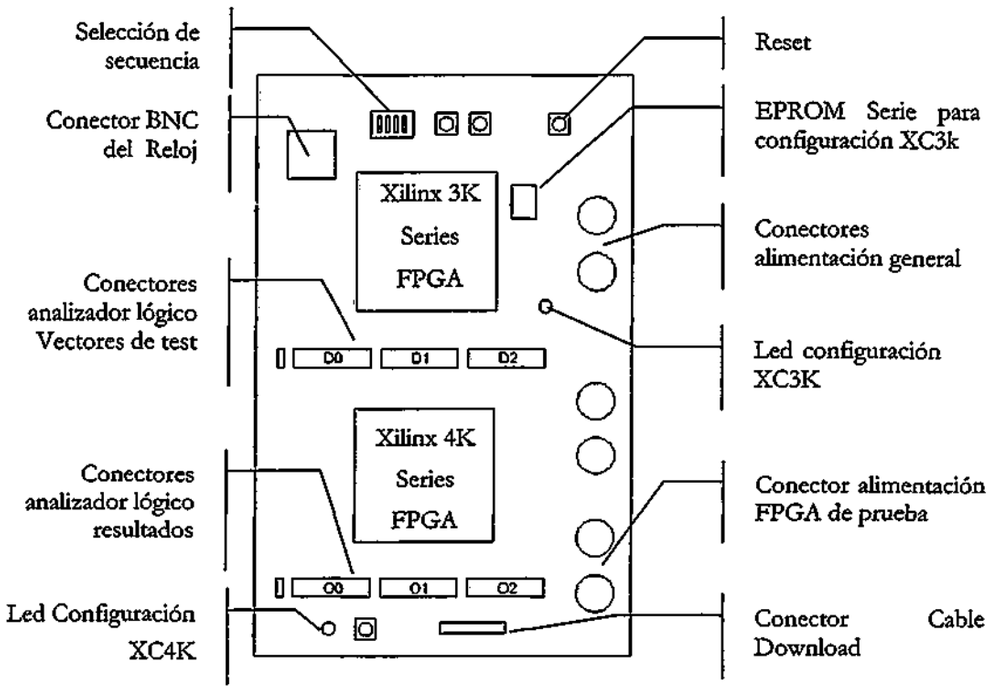


Figura A.1. Esquema de la placa de prueba de la familia XC4000.

A.4 Conexión de la placa de pruebas

Como se mencionó anteriormente, la placa posee alimentación separada para el generador de vectores de *test* así como para el circuito a medir. Una fuente de alimentación doble alimenta por separado cada parte de la placa. Sobre la FPGA a medir se colocan los elementos de medición (voltímetro y amperímetro) Figura A.2. Se conectan además el generador de reloj (Metrix GX245), el analizador lógico y el cable de programación conectado al puerto serie del PC (*Xchecker*).

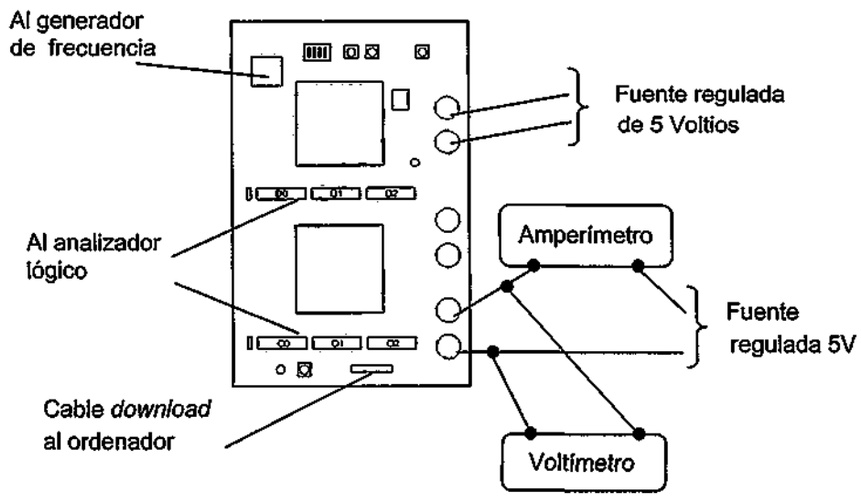


Figura A.2. Esquema de conexión del arreglo experimental en XC4K

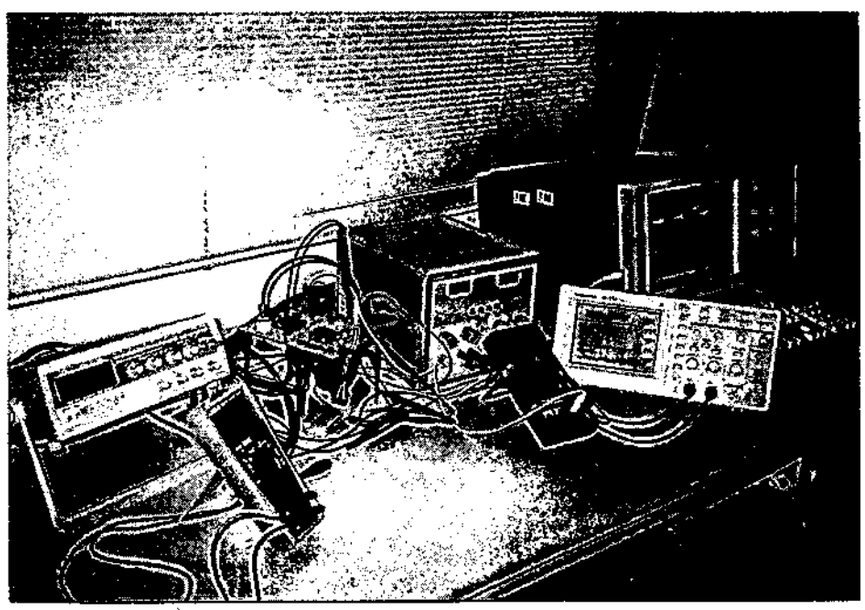


Figura A.3. Fotografía del arreglo experimental para XC4K

A.5 Método de medición de consumo

La técnica utilizada en este trabajo para aislar las distintas componentes del consumo fue propuesta en [Boe96]. Otra alternativa puede verse en [Men99]. Según [Boe96], la componente estática se puede medir dejando las entradas de la parte combinatoria fijas sin actividad en el reloj, luego de haber configurado la FPGA. Para aislar la componente externa de consumo el circuito se mide dos veces. Primero en condiciones normales de operación, luego deshabilitando los *buffers* de tres estados en las salidas. De esta manera, la componente *off-chip* se puede aproximar mediante la diferencia entre estos dos resultados.

Consumo Estático *(Static Power)*	El circuito es configurado pero no se le conectan estímulos ni señal de reloj. El resto de elementos externos que requiere la FPGA quedan conectados.
Consumo Externo *(Off-chip power)*	Para aislar la componente externa se mide dos veces. Primero en condiciones normales de operación, luego deshabilitando los *buffers* de tres estados en las salidas La componente off-chip se aproxima mediante la diferencia entre estos dos valores.
Consumo de Sincronización *(Synchronization power)*	Un dato constante (por ejemplo todas las entradas a cero) es usado como entrada al circuito, mientras el reloj es aplicado. De esta manera no existirá actividad en el circuito ya que no hay cambios en los datos mientras el reloj será el único elemento que consume. Cabe señalar que las FPGAs usan multiplexores para emular el efecto de la habilitación de reloj (*clock enable*), como consecuencia el uso del pin *clock enable* de del CLB no interrumpe realmente la alimentación de reloj a los flip-flops
Consumo Dinámico *(Dynamic Power)*	Surge de restar al consumo total, el consumo estático y externo. Dada la linealidad del consumo respecto de la frecuencia, el circuito se mide a diferentes velocidades de reloj para minimizar los errores.

Tabla A.4: Componentes del consumo y forma de medición en XC4K

Finalmente, para que quede solamente la potencia dinámica de la parte combinatoria, resta aislar la potencia de sincronización que se puede calcular como diferencia, midiendo el consumo con el circuito sin actividad en las entradas, mientras opera la

señal de reloj normalmente, de manera que sólo el árbol de reloj tenga actividad. Esta técnica de medición de la corriente de sincronización solo es aplicable a bloques que no tengan realimentaciones de registros internos. Por ello es que no se utiliza en las máquinas de estado. En la tabla A.4 se resumen estos conceptos.

Respecto al consumo *off-chip*, todos los circuitos a medir se implementan con *buffers* tri-estados en las patas de salida. Al margen, cada pata de salida tiene una carga producto del analizador lógico menor de 3 pf [Tek00].

La componente utilizada para realizar comparaciones y optimizaciones es la dinámica, ya que es la que es susceptible de ser optimizada por el diseñador. Eventualmente puede ser importante en las comparaciones revisar la componente de sincronización (la que depende del reloj), que se verá afectada por decisiones arquitecturales y topológicas. La componente estática y *off-chip* suele quedar fuera del espacio de diseño, dado que la primera depende de características tecnológicas y la segunda de cuestiones de interacción con otros sistemas o el mundo exterior.

Error en la medición:

Los voltímetros y amperímetros utilizados (Fluke 175 [Flu02]) para el cálculo de la potencia tienen la siguiente precisión: Como voltímetro de corriente continua y en el rango utilizado 0,15% de la lectura, en tanto que como amperímetro el 1%. La potencia medida como I × V por tanto tiene una precisión del orden del 1%.

A.6 Generador de vectores

En este arreglo experimental se utilizan diferentes configuraciones para la FPGA que realiza la generación de patrones (la XC3K) dependiendo de los vectores necesarios. La configuración del generador de vectores de test se lleva a cabo a través de la programación de una EEPROM serial, una AT17C128.

En la figura A.4, se muestra el esquema de un generador utilizado para comprobar la relación de consumo P(A×B) vs P(B×A) en multiplicadores. En este generador se puede elegir entre secuencias aleatorias y secuencias de máxima conmutación, el orden de salida y la frecuencia relativo de los operandos.

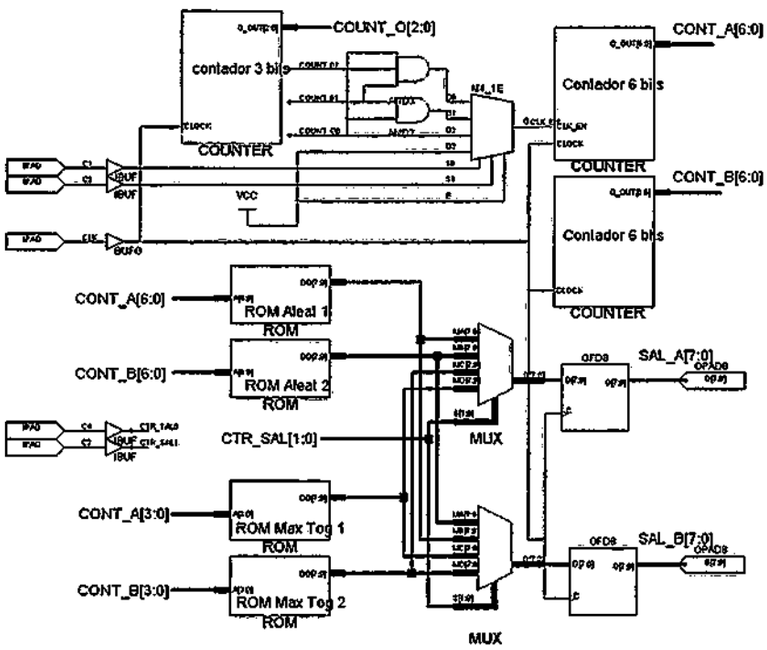


Figura A.4. Esquema de un generador de patrones en las XC3K

A.7 Referencias del apéndice

[Tek01] Tektronix inc; "TLA 704 Logic Analyzer user Guide". Available at www.tektronix.com, 2001.

[Lop97] S. Lopez-Buedo, "Técnicas de diseño de Alta Velocidad y Bajo Consumo" *Memoria Proyecto Fin de Carrera, ETSI Telecomunicación - Universidad Politécnica de Madrid.* Septiembre 1997.

[Xch99] Xchecker Xilinx download cable. "Hardware User Guide: XChecker Cable" http://toolbox.xilinx.com/docsan/3_1i/

[Boe96] E. Boemo, "Contribution to the Design of Fine-grain Pipelined VLSI Arrays", *Tesis doctoral, ETSI Telecomunicación, Universidad Politécnica de Madrid,* Madrid, Spain. 1996.

[Men99] L. Mengíbar, M. García, D. Martín, and L. Entrena, "Experiments in FPGA Characterization for Low-power Design", *Proceedings of XIV Conference on Design of Circuits and Integrated Systems (DCIS'99),* Palma de Mallorca, November 1999.

[Flu02] Fluke Ibérica "Multímetros digitales de la serie 170", avilable at http://www.fluke.es, 2002.

Apéndice B:

Placa de prueba AFX (Virtex)

B.1 Introducción

Para llevar a cabo los experimentos con FPGAs de la familia Virtex se empleó una tarjeta de prototipado diseñada y distribuida por Xilinx, la AFX PQ240-100 [Xil99a]. Esta tarjeta posee la ventaja de ser muy simple y prácticamente no tener componentes adicionales que pudiesen interferir con las medidas de consumo. En esta placa se puede montar cualquier FPGA de la familia Virtex de 2,5 V con encapsulado PQ240 [Xil01a].

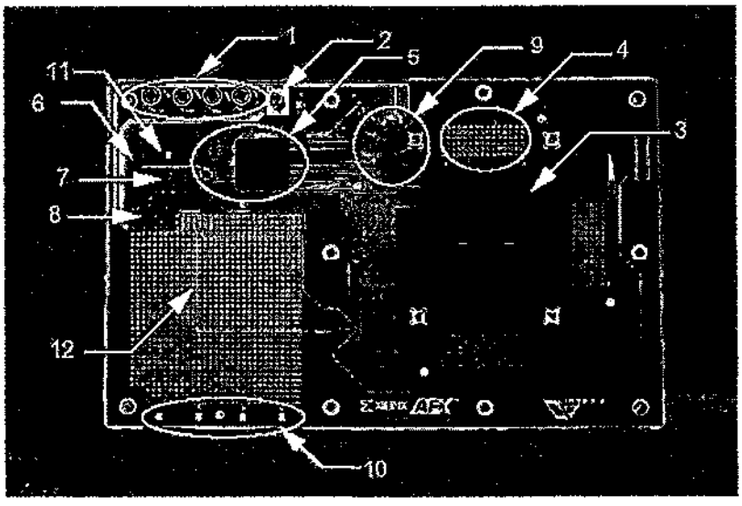


Figura B.1. Detalle de la placa de prototipado AFX

B.2 Características de la placa de prueba AFX

En la figura B.1 se puede ver una fotografía extraída de las notas técnicas de Xilinx. Básicamente, la tarjeta tiene unos conectores tipo bananas de alimentación (1), un *switch* de conexión (2), área con *pines* conectados a las patas de la FPGA (3) para poder monitorizar y excitar el circuito, un zócalo para una EPROM de configuración (5), un conector para programar la FPGA(6), zócalos para cuatro osciladores (9) (uno por cada línea global de reloj en Virtex), cuatro LEDs (10), *jumpers* de configuración (7 y 8) (principalmente selección de tensiones en los bancos de E/S y modo de configuración: M0, M1 y M2), y por último, un área de prototipado (12). A la placa se le han soldado (en la zona 4) conectores para facilitar la conexión y desconexión del generador de patrones y el analizador lógico.

Para programarla se utiliza cualquier cable de los provistos por Xilinx (por ejemplo *Paralell Cable* III ó IV [Xil03]). En este caso se utiliza una interfaz *ad hoc* descrita en [Lop03] que se maneja a través de JBits. La interfaz se basa en el modo bidireccional de funcionamiento del puerto paralelo, de esta manera se puede hacer *readbacks* de 8 bits en paralelo de una manera muy sencilla. Figura B.2 y B.3

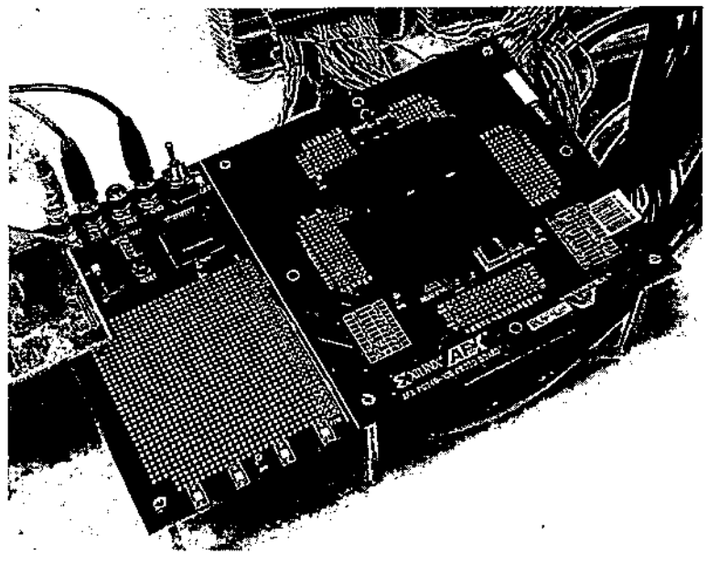


Figura B.2. Fotografía de la tarjeta AFX.

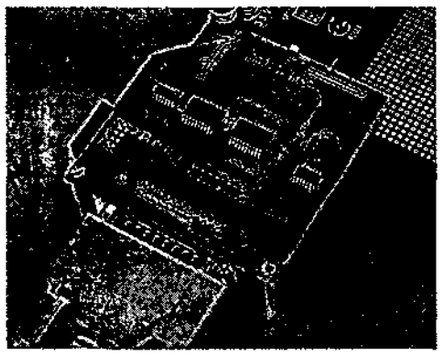


Figura B.3. Detalle de la interfaz con el puerto paralelo de la tarjeta AFX

B.3 Dispositivos Virtex utilizados

Los dispositivos utilizados con esta placa son una XCV50PQ240-4 (un arreglo de 16x24 CLBs, 768 slices, 57906 puertas equivalentes) y una XCV800PQ240-4 [Xil01b] (56x84 CLBs, 9408 slices, 888439 puertas equivalentes), fundamentalmente esta última FPGA se ha utilizado para la mayor parte de las medidas.

B.4 Conexión de la placa de pruebas

Las FPGAs de la familia Virtex se alimentan con 2,5 V el *core* del circuito en tanto que la periferia se alimenta con 3,3 V. Una fuente de alimentación doble alimenta por separado cada parte de la placa con las tensiones correspondientes. En el arreglo experimental se controlan ambas tensiones y la corriente del *core*, ya que en general se estudia el consumo dinámico del *core*, aislándose en el cálculo también el consumo de la periferia. En la figura D.3 se observa una fotografía del arreglo experimental.

Para la excitación de los circuitos se utiliza un generador de vectores de prueba TLA7PG2 de la empresa Tektronix [Tek02]. La señal de reloj para los circuitos se obtiene igualmente de este generador. Para comprobar los resultados se utiliza un analizador lógico TLA704 de la misma empresa [Tek01].

A la placa se han conectado 64 bits de entrada (procedente del generador de vectores de *test*). La tabla B.1 muestra los detalles de la asignación de *pines*. Un total de 64 patas de salida de la FPGA se conectaron al analizador lógico, cuya asignación de patas se puede ver en la tabla B.2.

Pin	A0	Pin	A1	Pin	B0	Pin	B1	Pin	C0	Pin	C1	Pin	D0	Pin	D1
5	A0_0	3	A1_0	33	B0_0	31	B1_0	186	C0_0	187	C1_0	217	D0_0	215	D1_0
6	A0_1	4	A1_1	34	B0_1	35	B1_1	189	C0_1	188	C1_1	218	D0_1	216	D1_1
9	A0_2	7	A1_2	38	B0_2	36	B1_2	193	C0_2	192	C1_2	221	D0_2	220	D1_2
13	A0_3	12	A1_3	41	B0_3	40	B1_3	194	C0_3	195	C1_3	222	D0_3	224	D1_3
17	A0_4	20	A1_4	46	B0_4	48	B1_4	201	C0_4	199	C1_4	229	D0_4	228	D1_4
21	A0_5	24	A1_5	49	B0_5	52	B1_5	202	C0_5	200	C1_5	230	D0_5	232	D1_5
25	A0_6	27	A1_6	53	B0_6	55	B1_6	205	C0_6	207	C1_6	234	D0_6	235	D1_6
26	A0_7	28	A1_7	57	B0_7	56	B1_7	206	C0_7	208	C1_7	237	D0_7	236	D1_7

Tabla B.1. Conexiones del generador de patrones a la FPGA.

Pin	A0	Pin	A1	Pin	B0	Pin	B1	Pin	C0	Pin	C1	Pin	D0	Pin	D1
5	A0_0	3	A1_0	33	B0_0	31	B1_0	186	C0_0	187	C1_0	217	D0_0	215	D1_0
6	A0_1	4	A1_1	34	B0_1	35	B1_1	189	C0_1	188	C1_1	218	D0_1	216	D1_1
9	A0_2	7	A1_2	38	B0_2	36	B1_2	193	C0_2	192	C1_2	221	D0_2	220	D1_2
13	A0_3	12	A1_3	41	B0_3	40	B1_3	194	C0_3	195	C1_3	222	D0_3	224	D1_3
17	A0_4	20	A1_4	46	B0_4	48	B1_4	201	C0_4	199	C1_4	229	D0_4	228	D1_4
21	A0_5	24	A1_5	49	B0_5	52	B1_5	202	C0_5	200	C1_5	230	D0_5	232	D1_5
25	A0_6	27	A1_6	53	B0_6	55	B1_6	205	C0_6	207	C1_6	234	D0_6	235	D1_6
26	A0_7	28	A1_7	57	B0_7	56	B1_7	206	C0_7	208	C1_7	237	D0_7	236	D1_7

Tabla B.2. Conexiones del generador de la FPGA al analizador lógico

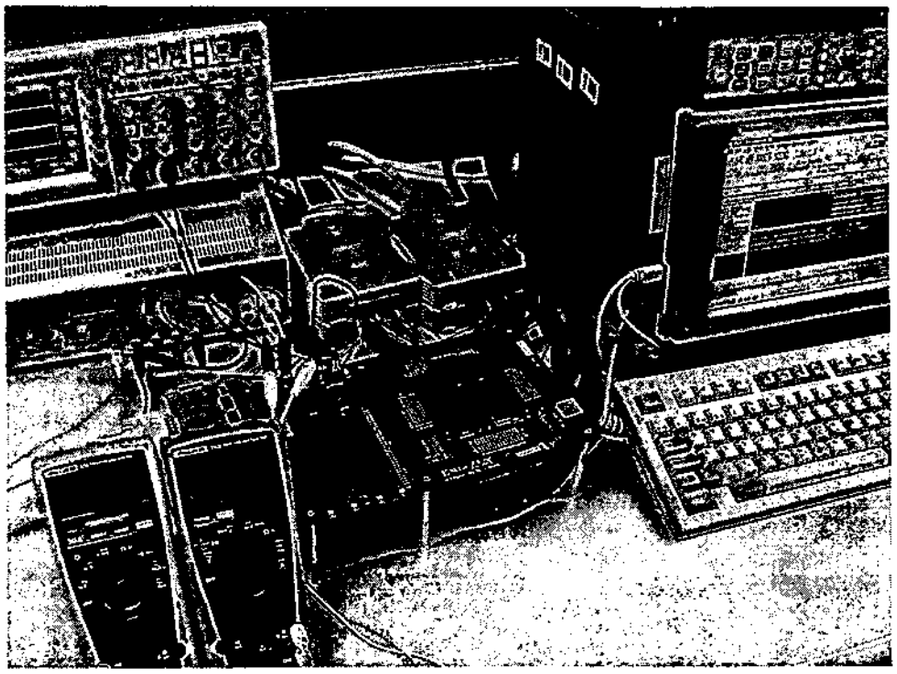


Figura B.4. Fotografía del arreglo experimental placa AFX

B.5 Metodología de medición de consumo

La metodología de medición es similar a la descrita en el apéndice A.4, solo que aquí se tiene en cuenta el hecho de que la periferia del circuito se alimenta a 3,3 V en tanto que el *core* del circuito se alimenta con 2,5 V. A consecuencia de lo antedicho el consumo externo (*off-chip power*) depende de la tensión de 3,3 V, teniendo una influencia despreciable sobre el consumo de 2,5V. Se realizaron pruebas para comprobar esta última afirmación construyendo circuitos con *buffers* tri-estados en las salidas y comprobando el consumo sobre la alimentación del *core* con los *buffers* conectados y desconectados respectivamente. En la tabla B.3 se resumen las componentes del consumo.

Consumo Estático *(Static Power)*	El circuito es configurado pero no se le conectan estímulos ni señal de reloj. El resto de elementos externos que requiere la FPGA quedan conectados. Se obtiene consumo estático tanto en la alimentación del *core*, como en la de la periferia.
Consumo Externo *(Off-chip power)*	El consumo externo está soportado por la alimentación de la periferia, con lo que rentándole el consumo estático se obtiene la componente *off-chip*.
Consumo de Sincronización *(Synchronization power)*	Un dato constante es usado como entrada al circuito, mientras el reloj es aplicado, por tanto solo el reloj produce consumo. Si no se utilizan FF en los IOBs solo se notará consumo sobre la alimentación del *core*, sino en ambas alimentaciones.
Consumo Dinámico *(Dynamic Power)*	Surge de restarle al consumo total de la alimentación del *core*, su consumo estático. Los IOBs contienen FF y un mínimo de lógica la que se trata de evitar de usar para no influir en las medidas.

Tabla B.3: Componentes del consumo y forma de medición en Virtex

Para obtener el consumo dinámico, solo bastará con medir el consumo sobre la alimentación del *core*, retándole el consumo estático. Por tanto el voltímetro y amperímetro (Fluke 175 [Flu02]) se colocan sobre la línea de alimentación de 2,5V. No obstante se controla la tensión de 3,3 V para evitar errores en las medidas.

El error en la medición es similar al expresado en la sección A.5.1, ya que se utiliza un instrumental similar. En el rango utilizado, el voltímetro posee un error 0,15% de la lectura, en tanto que el amperímetro el 1%.

B.6 Referencias del apéndice

[Flu02] Fluke Ibérica, "Multímetros digitales de la serie 170" available at http://www.fluke.es, 2002.

[Lop03] Sergio López-Buedo, "Técnicas de Verificación Térmica para Arquitecturas Dinámicamente Reconfigurables"; *Tesis doctoral Departamento de Ingeniería Informática; Universidad Autónoma de Madrid*, julio 2003.

[Tek01] Tektronix inc; "TLA 704 Logic Analyzer user Guide". available at www.tektronix.com

[Tek02] Tektronix inc; "TLA7PG2 Pattern Generator Module"; available at www.tektronix.com

[Xil01] Xilinx inc; "Virtex™ 2.5 V Field Programmable Gate Arrays Product Specification" Data Sheet DS003-1 (v2.5) April 2, 2001. Available at www.xilinx.com.

[Xil03] Xilinx Inc; "Xilinx Parallel Cable IV Advance Product Specification"; Data Sheet DS097 (v1.4) March, 2002 available at http:// toolbox.xilinx.com/docsan/

[Xil99] Xilinx inc "Xilinx Prototype Platforms User Guide for Virtex and Virtex-E Series FPGAs" Data Sheet DS020 (v1.1) December, 1999. Available at www.xilinx.com.

Apéndice C:

Placa de prueba para VIRTEX II

C.1 Introducción

Las medidas de consumo sobre la familia Virtex II se llevaron a cabo sobre una tarjeta de prototipado diseñada y distribuida por Xilinx denominada *Virtex-II prototype platform* [Xil03].

Si bien esta tarjeta es más compleja que la descrita en el apéndice B y al margen de tener componentes adicionales (FPGA de servicio, PROMs, etc.) que consumen corriente, éstas se alimentan por una conexión separada, con lo que las mediciones se pueden llevar a cabo sin inconvenientes. En esta placa se puede montar cualquier FPGA de la familia Virtex II de 1,5 V con encapsulado FG676 [Xil04].

La tarjeta de prototipado para Virtex II contiene dos FPGAs, una es el dispositivo a medir (DUT - *Device Under Test*) y una FPGA de servicio. El DUT puede ser programada por una PROM vía el conector a tal efecto (*socket* PROM) o utilizando uno de los dos conectores auxiliares a través de cualquier sistema de configuración de Xilinx como *MultiLINX cable* [Mul03], *Parallel Cable III* [Par00], *Parallel Cable IV* [Par04], *ó System ACE* [Ace01].

La FPGA de servicio es una matriz de interconexión que controla el rutado para todas las señales de configuración en la placa. Esto permite evitarse *jumpers* y *switches* para controlar la placa. Una PROM serie XC18V01 configura la FPGA de servicio. Ni la FPGA de servicio, si su PROM son parte de la cadena de configuración.

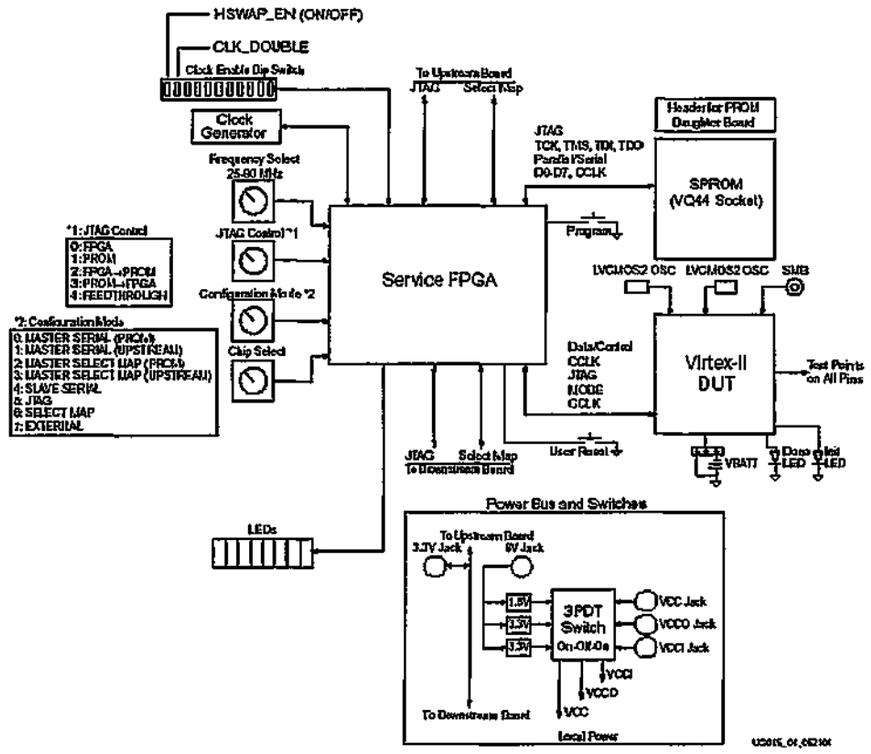


Figura C.1. Diagrama de bloques de la placa de prototipado para Virtex II [Xil03]

C.2 Características de la placa de prueba para Virtex II

La figura C.1 muestra un diagrama de bloques de la placa de pruebas, en tanto la figura C.2 es una fotografía detallada extraída de las notas de aplicación de Xilinx. Los elementos más destacables de la placa son: Interruptor de corriente (1) posee tres posiciones, conecta la corriente a la placa vía los reguladores de tensión ubicados en la parte inferior de la placa, desconecta o conecta la alimentación vía los bornes de alimentación separados (2). Conector de programación (3), se puede utilizar *MultiLINX* ó *Parallel Cable III*. Selector de frecuencia para el reloj interno (4); Selector

para el modo de programación de la DUT (5), los conectores 5a, 5b son para *system ACE* [Ace01], en tanto 5c y 5d son conectores de interfase para los modos de programación *select-map* ó *slave-serial*. Control de la cadena JTAG (6). Control del *chip select* (7) asigna un número entre 0 y 3 para el DUT en la cadena de *chip-select*. Conector para la EPROM del usuario (8), acepta las memorias de la serie XC17V01-V04 y XC18V01-V04 con empaquetado VQ44. FPGA de servicio (11), se configura con una EPROM propia que no está en la cadena de JTAG principal, vía el conector específico (9). Conexiones para osciladores externos (15). Conexión para generadores de onda externos (16); Socket para la FPGA a medir (19), las patas para ingresar y sacar datos están mapeadas en la zona de conexiones (20), allí se pueden soldar conectores para utilizar generador de patrones y analizador lógico. Posee leds programables por el usuario (21), e interruptores de *reset* y programación (22,23). Led de inicialización (*init*) y finalización de la configuración (directo de la pata *done*) (24 y 25).

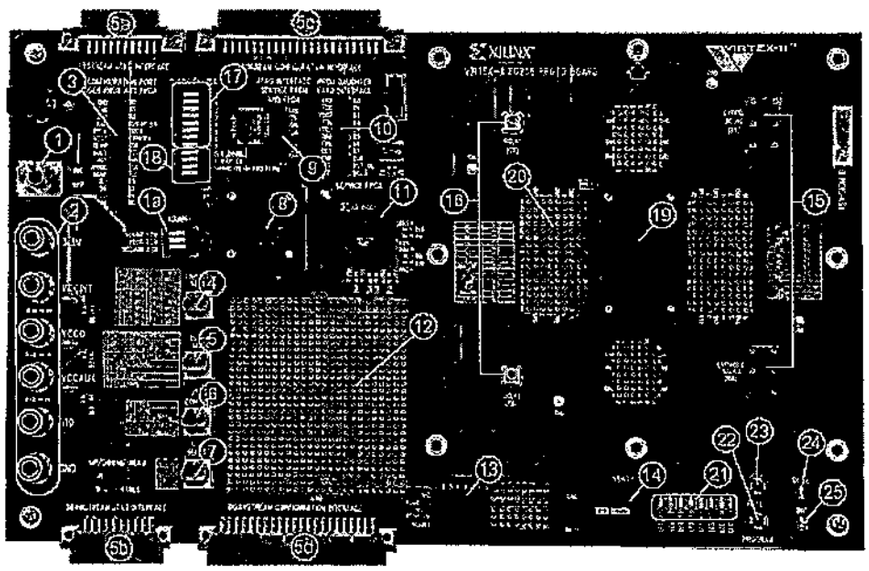


Figura C.2. Detalle de la placa de prototipado para Virtex II [Xil03].

C.3 Dispositivos Virtex II utilizados

En esta placa se pueden conectar dispositivos Virtex II con encapsulado FG676. Las combinaciones de encapsulado y dispositivo de las hojas de datos de Virtex II [Xil04] indican que existen tres dispositivos con este encapsulado: XC2V1500, XC2V2000 y XC2V3000, con 1,5, 2 y 3 millones de puertas equivalentes respectivamente.

Los dispositivos disponibles para esta tesis son una XC2V1500FG676-6 (una matriz de 48 x 40 CLB, 7680 slices, 48 BlockRAMs y 48 multiplicadores) y una XC2V3000FG676-6 (64 x 56 CLB, 14336 slices, 96 BlockRAMs y 96 multiplicadores). Para las medidas de ésta tesis se utiliza mayoritariamente el primer dispositivo.

C.4 Conexión de la placa de pruebas

Las FPGAs de la familia Virtex II utilizada alimenta el *core* del circuito a 1,5 V (Vccint), la periferia a 3,3 V (Vccout), en tanto que la tensión auxiliar depende del estándar de salida. En este caso se utiliza LVTTL y por ende se alimenta con 3,3 V (Vccaux). Adicionalmente, la placa posee una alimentación de 3,3 V para la FPGA de servicio, las memorias EPROMs y demás circuitos de la placa.

Dos fuentes de alimentación dobles alimentan por separado cada parte de la placa con las tensiones correspondientes. En el arreglo experimental se controlan todas las tensiones y las corrientes del *core*, periferia y auxiliar. Por lo general el consumo dinámico del *core*, obviándose el resto de las corrientes. En la figura C.4 se observa una fotografía del arreglo experimental.

Para la excitación de los circuitos se utiliza un generador de vectores de test TLA7PG2 de la empresa Tektronix [Tek02], la señal de reloj para los circuitos se obtiene igualmente de este generador. Para comprobar los resultados se utiliza un analizador lógico TLA704 de la misma empresa [Tek01].

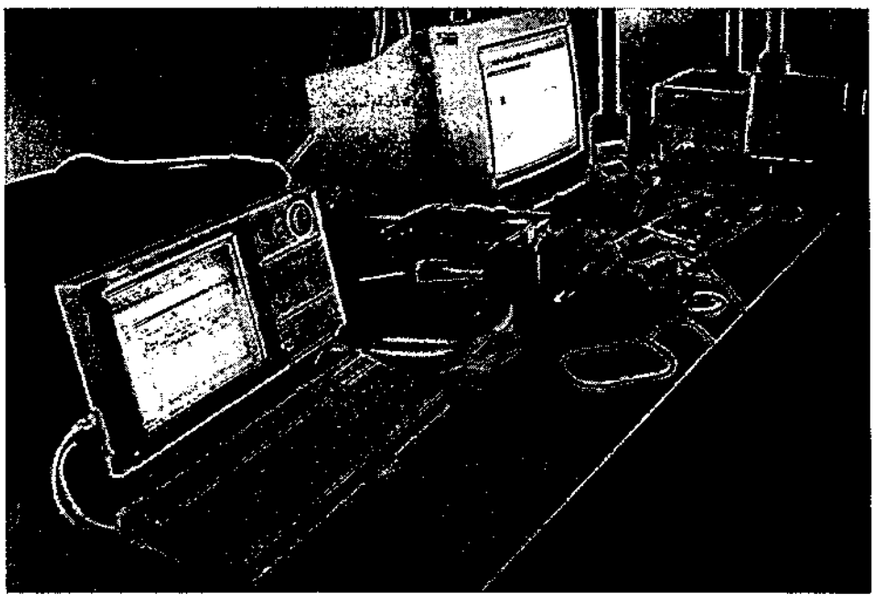


Figura C.4. Fotografía del arreglo experimental placa Virtex II

C.5 Metodología de medición de consumo

La metodología de medición es similar a la expresada en los apéndices A y B, solo que aquí se tiene en cuenta el hecho de que la periferia y la tensión auxiliar del circuito se alimenta a 3,3 V, en tanto que el *core* del circuito se alimenta con 1,5 V. Como consecuencia, el consumo externo (*off-chip power*) será alimentado por la tensión de 3,3 V (Vccaux y Vccout), teniendo una influencia despreciable sobre el consumo del *core* (1,5V). Se realizaron pruebas para comprobar esta última afirmación. En la tabla C.1 se resumen las componentes del consumo.

Consumo Estático *(Static Power)*	El circuito es configurado pero no se le conectan estímulos ni señal de reloj. El resto de elementos externos que requiere la FPGA quedan conectados. Se obtiene consumo estático tanto en la alimentación del *core* (Vccint), como en la de la periferia (Vccout y Vccaux).
Consumo Externo *(Off-chip power)*	El consumo externo está soportado por la alimentación de la periferia y la auxiliar, con lo que restándole el consumo estático de cada componente se obtiene la componente *off-chip*.
Consumo de Sincronización *(Synchronization power)*	Un dato constante es usado como entrada al circuito, mientras el reloj es aplicado, por tanto solo el reloj produce consumo. Si no se utilizan FF en los IOBs solo se notará consumo sobre la alimentación del *core*, sino en ambas alimentaciones.
Consumo Dinámico *(Dynamic Power)*	Surge de restarle al consumo total de la alimentación del *core*, su consumo estático. Los IOBs contienen FF y un mínimo de lógica la que se trata de evitar su utilización para no influir en las medidas.

Tabla C.1. Componentes del consumo y forma de medición en Virtex II

Para obtener el consumo dinámico, solo basta con medir el consumo sobre la alimentación del *core*, restándole el consumo estático de esa alimentación. Por tanto el voltímetro y amperímetro se colocan sobre la línea de 1,5V. No obstante se controlan la tensión y corriente sobre las alimentaciones de 3,3 V, ya queen ciertos experimentos se mide el consumo sobre Vccout y Vccaux.

Respecto al error en las medidas al igual que en los arreglos experimentales del Apéndices A y B, el error del voltímetros en el rango utilizado es del 0,15% de la lectura, en tanto que en los amperímetros el error llega al 1%. El error en la potencia calculado como el producto I × V es del orden del 1 %.

C.6 Referencias del apéndice

[Ace01] Eric Thacker, Xilinx, Inc. "System Ace: Configuration Solution for Xilinx FPGAs" *White Paper 151 (v1.0).* Available at www.xilinx.com. September, 2001

[Mul03] Xilinx inc, "Getting Started with the MultiLINX Cable" *Xilinx Application Note XAPP168 (v2.1)* April 29, 2003.

[Par00] Xilinx inc, "Hardware User Guide, Chapter 1: Cable Hardware - Parallel Cable III" Xilinx Foundation documentation Ver 3.1. Available at http://toolbox.www.xilinx.com/docsan/3_1i/. 2000

[Par04] Xilinx inc, "Xilinx Parallel Cable IV", Data Sheet DS097 (v2.0) January, 2004. Available at www.xilinx.com.

[Tek01] Tektronix inc; "TLA 704 Logic Analyzer Family user Guide version 4.1"; 2001. available at www.tektronix.com

[Tek02] Tektronix inc; "TLA7PG2 Pattern Generator Module Instruction Manual"; 2002. Available at www.tektronix.com.

[Xil03] Xilinx inc, "Virtex II Prototipe platform user guide UG015 v.1.1", January 2003, Available at www.xilinx.com

[Xil04] Xilinx Inc, "Virtex-II Platform FPGAs: Complete Data Sheet". Data sheet DS031 (v3.3) June, 2004. Available at http://www.xilinx.com.

Apéndice D:

Traductor Kiss2VHDL

D.1 Introducción

Se describe aquí la herramienta software construida para la realización de los experimentos de consumo con diferentes codificaciones en máquinas de estados finitos (Capítulo 4). Como prólogo se describen el formato de especificación de máquinas de estado KISS2 [Sen92] y los grupos de bancos de pruebas (*benchmarks tests*) MCNC [Lis88] y PREP [Pre00].

D.2 El formato KISS

El formato KISS se origino con el programa de minimización de funciones lógicas de dos niveles usados para PLAs llamado *espresso* [Bra94]. El formato evolucionó para dar soporte a las máquinas de estado finitos (llamado KISS2) y se ha extendido su utilización en múltiples herramientas de asignación y minimización de estados.

En KISS2 cada estado es simbólico, dado un estado y un vector de bits de entrada, la tabla de transición, indica el próximo estado en forma simbólica y el vector de bits de salida. Las condiciones externas sin importancia (*don't care*) se indican a través de no escribir la transición (es decir no se escribe la combinación estado actual-combinación de entrada). Otra alternativa es escribir que no importa el estado de un bit de entrada o salida, y para ello se utiliza el carácter '-', es decir, indica que puede ser indiferentemente 0 ó 1.

Los archivos KISS2 son en texto plano. El carácter '#' da inicio a un comentario hasta el fin de línea (hasta el carácter salto de línea). Las siguientes palabras claves son reconocidas, donde [d] indica un número decimal y [s] una cadena de texto

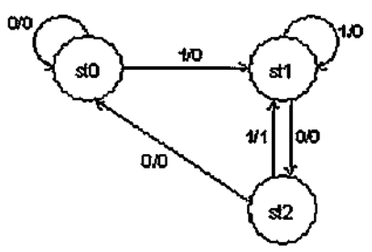
.i [d] Especifica la cantidad de variables de entrada.
.o [d] Especifica la cantidad de variables de salida
.p [d] Especifica la cantidad de reglas (opcional)
.s [d] Especifica la cantidad de estados (opcional)
.r [s] Especifica el estado de reset (opcional)

Luego se colocan las reglas en la tabla de estados, donde **current_state** y **next_state** son nombres simbólico que representan estados, e **input** y **output** son los vectores de bits de la entrada y salida respectivamente

inputs current_state next_state outputs

.e o .end marcan el fin de la desscripción

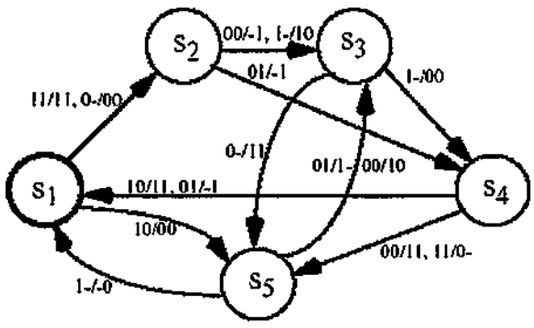
Un primer ejemplo: Un diagrama de transición de estados (STG) de una máquina de estados con un bit de entrada y otro de salida, tres estados 6 reglas. Las palabras claves correspondientes a la cantidad de estados y reglas son opcionales y no se han descrito en este ejemplo.




```
.i 1
.o 1
0 st0 st0 0
1 st0 st1 0
0 st1 st2 0
1 st1 st1 0
0 st2 st0 0
1 st2 st1 1
.e
```

Figura D.1. Un primer ejemplo de descripción KISS2.

Segundo ejemplo: Cada posición en la columna de variables de entrada o salida indica una variable del mundo exterior. En las entradas los '0' indican términos complementados, en tanto que los '1' términos directos. El carácter '–' que la variable no está presente (*don't care*).




```
.i 2
.o 2
.p 15
.s 5
.r s1
11 s1 s2 11
0- s1 s2 00
10 s1 s5 00
00 s2 s3 -1
1- s2 s3 10
01 s2 s4 -1
1- s3 s4 00
0- s3 s5 11
10 s4 s1 11
01 s4 s1 -1
00 s4 s5 11
11 s4 s5 0-
1- s5 s1 -0
00 s5 s3 10
01 s5 s3 1-
.end
```

Figura D.2. Ejemplo de grafo de transición de estados y su correspondiente especificación en KISS2

D.3 Bancos de pruebas (*Benchmarks*)

Para medir y comparar consumo en máquinas de estado se han utilizados los circuitos bancos de prueba del MCNC [Lis88] y los del consorcio PREP [Pre00].

MCNC Benchmarks. El banco de pruebas para síntesis lógica y optimización (*logic synthesis and optimization benchmarks*) es distribuido por el Centro de microelectrónica de Carolina del Norte (Microelectronics Center of North Carolina) e incluye los conjuntos de bancos de prueba ISCAS'85 e ISCAS'89. El primer informe de gran difusión (versión 2.0) data de 1988 [Lis88] y ha sido ampliado en la versión 3.0 [Yan91], siendo ésta última la versión utilizada en esta tesis.

El conjunto de circuitos de prueba esta divido en cuatro categorías fundamentales: Máquinas de estado en formato KISS2; Lógica secuencial multinivel en extended BLIF *(Berkeley Logic Interchange Format)* o SLIF (*Structure Logic Interchange Format*); Lógica combinacional multinivel en extended BLIF o SLIF. Por última lógica de dos niveles en *Berkley PLA format* (Como KISS pero sin estados) o SLIF. Se puede conseguir en la dirección ftp://mcnc.mcnc.org, directorio pub/benchmark /LGSynth91.

PREP Benchmarks. El consorcio de compañías de electrónica programable (*Programmable Electronics Performance Company* - PREP) se creo con el ánimo de generar una serie de pruebas que midiesen las prestaciones en FPGAs y otros dispositivos programables. De los nueve circuitos que distribuye la versión 1.3, nos interesan los circuitos 3 y 4 que implementan máquinas de estados (de 8 y 16 estados respectivamente).

Actualmente el consorcio PREP se ha disuelto pero se pueden conseguir los circuitos en VHDL o Verilog en una implementación de Symplicity [Syn88] o bien desde la página web del proyecto http://www.prep.org.

D.3 Traductor Kiss2VHDL

El programa Kiss2VHDL codifica los estados de la máquina que se ingresa en formato KISS2 según los parámetros ingresados y genera dos ficheros VHDL. Uno con la entidad de la máquina de estados y un *top-level* que instancia la máquina de estados y agrega *buffers* de tercer estado en las patas del circuito.

Modo de uso: *Kiss2VHDL [options] [input]*

```
-n          No state Encoding, Only transform KISS into VHDL
-h          print help message
-e option   specify encoding option
            r: random encoding

            h: one hot encoding
            t: two hot encoding
            o: output dominant algorithm (default) "out-oriented"
[input]     file to be processed (default stdin)
```

El programa esta escrito en C, posee un analizador sintáctico (*parser*) para transformar el fichero KISS2 leído en una representación de diagrama (o grafo) de transición de estados. Posee rutinas para la asignación de estados y funciones para la escritura del código VHDL.

A continuación se muestra el ejemplo de la traducción del ejemplo de la figura C.2 con codificación *One Hot* en código VHDL.

```
-- MEALY machine
--
-- generado por el traductor kiss2vhdl
--
library ieee;
use ieee.std_logic_1164.all;
library SYNOPSYS;
use SYNOPSYS.attributes.all;

entity ejemplo is
        port(clk, rst: in std_logic;
                i : in std_logic_vector (1 downto 0);
                o : out std_logic_vector (1 downto 0) );
end ejemplo;

architecture behave of ejemplo is
type state is ( s1, s2, s5, s3, s4);
attribute enum_encoding of state: type is
        "10000 " & -- s1
        "01000 " & -- s2
        "00100 " & -- s5
        "00010 " & -- s3
        "00001 " ; -- s4

signal machine, next_state : state;
signal o_i : std_logic_vector (1 downto 0);

begin

-- Sincronization
state_machine: process (rst, clk)
begin
        if rst = '1' then
                machine <= s1;
        elsif rising_edge(clk) then
                machine <= next_state;
        end if;
end process state_machine;

next_state_process: process (machine, i)
begin
case machine is
when s1 =>
        if i = "11" then
                next_state <= s2 ;
                o_i <= "11";
        elsif i(1) = '0' then
                next_state <= s2 ;
                o_i <= "00";
        elsif i = "10" then
                next_state <= s5 ;
                o_i <= "00";
        else
                next_state <= s1;
                o_i <= (others => 'X');
        end if;
when s2 =>
        if i = "00" then
                next_state <= s3 ;
                o_i <= "-1";
        elsif i(1) = '1' then
```

```
                next_state <= s3 ;
                o_i <= "10";
        elsif i = "01" then
                next_state <= s4 ;
                o_i <= "-1";
        else
               next_state <= s2;
               o_i <= (others => 'X');
        end if;
when s5 =>
        if i(1) = '1' then
                next_state <= s1 ;
                o_i <= "-0";
        elsif i = "00" then
                next_state <= s3 ;
                o_i <= "10";
        elsif i = "01" then
                next_state <= s3 ;
                o_i <= "1-";
        else
               next_state <= s5;
               o_i <= (others => 'X');
        end if;
when s3 =>
        if i(1) = '1' then
                next_state <= s4 ;
                o_i <= "00";
        elsif i(1) = '0' then
                next_state <= s5 ;
                o_i <= "11";
        else
               next_state <= s3;
               o_i <= (others => 'X');
        end if;
when s4 =>
        if i = "10" then
                next_state <= s1 ;
                o_i <= "11";
        elsif i = "01" then
                next_state <= s1 ;
                o_i <= "-1";
        elsif i = "00" then
                next_state <= s5 ;
                o_i <= "11";
        elsif i = "11" then
                next_state <= s5 ;
                o_i <= "0-";
        else
               next_state <= s4;
               o_i <= (others => 'X');
        end if;
when others =>
               next_state <= s1;
               o_i <= (others => 'X');
end case;
end process next_state_process;

o <= o_i;
end behave;
```

D.4 Referencias del Apéndice

[Syn98] Synplicity, Inc. "Synplify User Guide Release 5.0: PREP VHDL Benchmark Examples", August 1998. http://www.synplicity.com

[Lis88] Bob Lisanke. "Logic synthesis and optimization benchmarks". *Technical report, MCNC,* Research Triangle Park, North Carolina, December 1988.

[Pre00] Programmable Electronics Performance Corporation (PREP) Benchmarks (Programmable Electronics Performance Company), see: http://www.prep.org.

[Sen92] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. "SIS: A System for Sequential Circuit Synthesis". *Technical Report Memorandum No. UCB/ERL M92/41,* Univ. of California, Berkeley, 1992.

[Bra84] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. "Logic Minimization Algorithms for VLSI Synthesis". *Kluwer Academic Publishers,* 1984.

[Yan91] Saeyang Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0", *Technical report, MCNC, Research Triangle Park,* North Carolina, January 1991.

Apéndice E:

Particionador de máquinas de estado: *Part_FSM*

E.1. Introducción

El programa *Part_FSM* es un programa C++ para la partición de máquinas de estados. Como entrada acepta máquina de estados descritas en formato KISS2 [Sen92] (apéndice D), generando como resultado la partición de la máquina de estados según lo expresado en la sección 4.4. En primer lugar lee la máquina de estados y la transforma en una representación interna de grafo de transición de estados (STG – *State Transition Graph*), luego le calcula la probabilidad estática. Más tarde, se aplica el algoritmo de partición de máquinas de estado, para por último generar el código VHDL de la máquina de estados particionada. El programa acepta parámetros para especificar el tipo de arquitectura (*arquitectura I y II* según la sección 4.3.4) u ortogonal (sección 4.3.3) además del tipo de método de bloqueo necesario.

E.2 Estructuras de datos

La principal estructura de datos es el grafo de transición de estados (STG – *State Transition Graph*), donde se almacena la máquina de estado. La estructura básica es el estado (*state*) y la transición entre estados (*transition*) (la declaración de la parte privada de las clase se puede ver en la figura D.1), luego el STG es un arreglo de estados. Otra estructura importante es la partición (*partition*) que almacena en dos arreglos los estados de cada submáquina de estados.

E.3 Cálculo de probabilidad estática

La probabilidad estática se calcula tal lo expresado en la sección 4.3.5. A medida que se leen las líneas que implican transiciones en el código KISS2, se cargan en el grafo de transición de estados y se asignan las probabilidades a los arcos. La probabilidad condicional de transición de un arco hacia otro, si se considera equiprobabilidad de las entradas, es tan simple como dividir la proporción de combinaciones que generan la transición respecto del total.

En la Figura 4.16 se puede observar un ejemplo de cálculo de probabilidades. Se puede ver en la Figura 4.16.b las probabilidades calculadas. Por otra parte, en el ejemplo de la Figura E.3 todas las probabilidades de transición son de 0,5.

```
class state
{ private :
     int present_state;
     int no_of_fanout;
     int no_of_fanin;
     int inputs;
     float steady_state_probability;
     transition* init, *ptr;
     vector<int> fanin_states;
     vector<transition*> fanin_edges;
  public :
            . . .
};
```

```
class transition
{ private :
     int nextstatenum;
     char* input_bits;
     char* output_bits;
     state* next_state;
     float transition_probability;
     transition* next_transition;
  public :
          . . .
};
```

Figura E.1. Declaración de la parte privada de las clases *state* y *transition*.

Para resolver el sistema de *n* ecuaciones y así obtener la probabilidad estática se implementa un algoritmo iterativo para el cálculo. Una vez obtenido la probabilidad estática para cada estado se procede a calcular la probabilidad de transición, que surge

de multiplicar la probabilidad estática de cada estado origen por la probabilidad de dicha transición. Una vez obtenido un grafo con la información que muestra la Figura 4.16.c y la Figura 4.16.d, se procede a invocar al algoritmo de partición.

E.4 Partición de la máquina de estados

El criterio para la división de la máquina de estados es la minimización de la probabilidad de transiciones entre las máquinas de estados, es decir: $\min(\sum p(i,j)), \quad \forall i \in \Pi_A, j \in \Pi_B$; donde *p(i,j)* es la probabilidad de transición de la máquina original y $\Pi_A = \{S_{a1}, S_{a2}, \ldots S_{an}\}$ y $\Pi_B = \{S_{b1}, S_{b2}, \ldots S_{bm}\}$son las particiones generadas.

Existe otro criterio extra dentro del algoritmo que es la diferencia de cardinalidad de las dos particiones, siendo esta un parámetro. Las pruebas empíricas sobre las arquitecturas diseñadas muestran que los mejores resultados se logran con la misma cantidad de estados en las submáquinas, no obstante es un parámetro por si se desea implementar sobre otra arquitectura.

Se ha implementado un algoritmo de *backtracking* (que explora todas las combinaciones posibles) con funciones de poda para reducir el tiempo de cálculo. No fue necesario desarrollar una heurística para resolver el problema, ya que la búsqueda exhaustiva resuelve el peor caso presentado dentro del banco de pruebas del MCNC91 [Lis88][Yan91] en unos pocos segundos. En la figura E.2 se puede observar el código para la generación de las particiones.

```
// código del algoritmo de partición
void partitioner (state* STG, partition *P, float &minDist, int n_state,
        int tot_st, int max_st, float actual_dist, int n_P0, int n_P1)
{ int numEdges;  float grow_dist;
  if (n_state <  tot_st){ si no se han asignado todos los estados
      //asigno n_state a subconj (particion) 0
      if (n_P0 < max_st) { //siempre que no me pase del limite
         grow_dist = aumento_distance(P,STG,numEdges,n_state,0);
         if ((grow_dist + actual_dist) < minDist){
              P->put(n_state,0);
              partioner(STG,P,minDist,n_state+1,
                tot_st, max_st,(grow_dist+actual_dist),n_P0+1, n_P1);
              P->del_last(0);
          }
      }
       //asigno n_state a subconj (particion) 1
       if (n_P1 < max_st) { //siempre que no se pase del limite
          grow_dist = aumento_distance(P,STG,numEdges,n_state,1);
          if ((grow_dist+actual_dist) < minDist){
              P->put(n_state,1);
              partitioner(STG,P,minDist,n_state+1,
                  tot_st, max_st,(grow_dist+actual_dist),n_P0,n_P1 + 1);
              P->del_last(1);
          }
      }
  }
    else{ //se han asignado todos los estados
       float pd = partition_distance(P,STG,numEdges);
       if ( pd < minDist) { //si es la major sssolución se guarda
           minDist = pd;
           P->print_partition();
           if (minP != NULL) delete (minP);
            minP = P->copy();
        }
      }
}
```

Figura E.2. Código para la partición de máquinas de estados.

E.5 Generación del código VHDL

Una vez realizada la partición de la máquina de estados se genera el código VHDL. El generador de código tiene en cuenta los parámetros que se han ingresado para generar el código correspondiente. Se puede generar la *arquitectura I* o *arquitectura II*; elegir el tipo de bloqueo de los datos entre puertas ANDs, *Latches* y *buffers* de tercer estado y además el tipo de codificación de las submáquinas.

A continuación se muestran un ejemplo en base al circuito *dk27* del MCNC que se puede ver en la figura E.3 En la figura E.4 se muestra el código KISS2 que se utiliza como entrada al programa y la salida por pantalla con la información de la probabilidad calculada que exhibe la herramienta. La figura E.5 muestra la información que brinda la herramienta respecto de la partición realizada. Más adelante, la figura D.6 muestra gráficamente los datos de la partición.

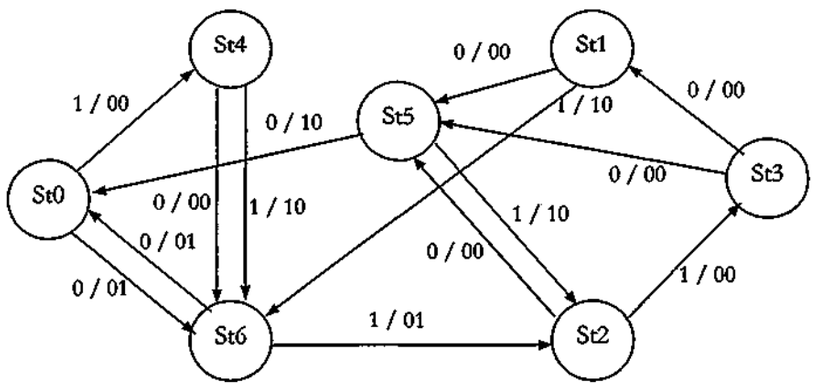


Figura E.3. Diagrama de transición de estados de la máquina de estados DK27.

```
.model dk27
.i 1
.o 2
.p 14
.s 7
.r st0
0 st0 st6 00
0 st2 st5 00
0 st3 st5 00
0 st4 st6 00
0 st5 st0 10
0 st6 st0 01
0 st1 st5 00
1 st6 st2 01
1 st5 st2 10
1 st4 st6 10
1 st1 st6 10
1 st0 st4 00
1 st2 st3 00
1 st3 st1 00
.end
```

```
Reading Design: "dk27"
Finished              reading
successfully
Model Name: dk27
Number of states: 7
Number of Inputs: 1
Number of Outputs: 2
Number of transitions: 14
Reset State: 0

Steady State probabilities:
 st0 --> 0.25
 st1 --> 0.0357143
 st2 --> 0.25
 st3 --> 0.0714286
 st4 --> 0.0714286
 st5 --> 0.142857
 st6 --> 0.1785710.142857
```

```
Transitions
probabilities:
st0-->st6    0.125
st0-->st4    0.125
st1-->st5    0.0178571
st1-->st6    0.0178571
st2-->st5    0.125
st2-->st3    0.125
st3-->st5    0.0357143
st3-->st1    0.0357143
st4-->st6    0.0357143
st4-->st6    0.0357143
st5-->st0    0.0714286
st5-->st2    0.0714286
st6-->st0    0.0892857
st6-->st2    0.0892857
```

Figura E.4. a) Código KISS2 de entrada. b) Información de la herramienta con las probabilidades calculadas.

```
Time elapsed (seconds): 0
Time elapsed (clocks): 230     (CLOCKS_PER_SEC: 1000)

Detail of Partitions
Number of States original FSM: 7
Size Partition 0: 3
Size Partition 1: 4
States Partition 0:  st0  st4  st6
States Partition 1:  st1  st2  st3  st5

Transitions of Partition 0 to Partition 1:
Edge st6 -> st2  Prob:0.0892857
Total probabilities of transition part0 to part1: 0.0892857

Transitions of Partition 1 to Partition 0:
Edge st1 -> st6  Prob:0.0178571
Edge st5 -> st0  Prob:0.0714286
Total probabilities of transition part0 to part1: 0.0892857
```

Figura E.5. Información de las particiones generadas.

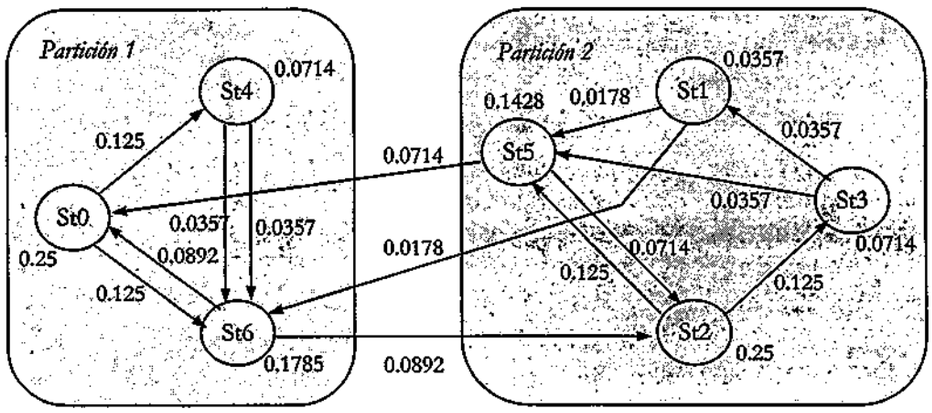


Figura E.6. Partición generada para el circuito *dk27*.

Por último, se muestra el código VHDL para el circuito dk27 con la arquitectura I, *latches* para bloquear los datos y *buffers* tri-estados para las salidas. En la estructura se reconocen cuatro procesos: *buf_tri_output* que genera *buffers* de tercer estado para las salidas; *state_machine* para la inferencia de registros; *blocking_latches* para inferir los *latches* de las entradas; *next_state_proces_a* y *next_state_proces_a*; son los bloques combinacionales de las submáquinas de estado. Por último, *output_process* para generar el multiplexor de la salida.

```
-- Partitioned FSM generated by part_FSM
-- Design: dk27.kiss2
-- Number of States: 7
-- Partition 0:  0  4  6
-- Partition 1:  1  2  3  5

library ieee; use ieee.std_logic_1164.all;

entity dk27 is
      port(clk, rst, tri: in std_logic;
            i : in std_logic_vector (0 downto 0);
            o : out std_logic_vector (1 downto 0) );
end dk27;

architecture behave of dk27 is

type state is ( st0, st1, st2, st3);
attribute enum_encoding of state: type is
      "1000 " &   -- P0: 0   -- P1: 1
      "0100 " &   -- P0: 4   -- P1: 2
```

```
       "0010 " &  -- P0: 6  -- P1: 3
       "0001 " ;  -- P1: 5

-- Internal signal declarations
signal next_a, next_b, machine, machine_a, machine_b: state;
signal activeFSM_a, activeFSM_b: std_logic;
signal activeFSM: std_logic;
signal inp_a, inp_b: std_logic_vector (0 downto 0);
signal o_i_a, o_i_b: std_logic_vector (1 downto 0);
signal sal: std_logic_vector (1 downto 0);

begin

-- Output Triestate Buffer
buf_tri_output: process (tri,sal)
begin
       if tri = '1' then
              o <= sal;
       else
              o <= others => 'Z';
       end if;
end process buf_tri_output;

-- Sincronization
state_machine: process (rst, clk)
begin
if rst = '1' then
       machine <= st0;
elsif rising_edge(clk) then
       if activeFSM = '0' then
              machine <= next_a;
              activeFSM <= activeFSM_a;
       else
              machine <= next_b;
              activeFSM <= activeFSM_b;
       end if;
end if;
end process state_machine;

-- Inputs Latches, states and inputs
blocking_latches: process (activeFSM, machine, i)
begin
       if activeFSM = '0' then
              inp_a <= i;
              machine_a <= machine;
       else
              inp_b <= i;
              machine_b <= machine;
       end if;
end process input_latches;

next_state_process_a: process (machine_a, inp_a)
begin
case machine_a is
```

```
when st0 =>  --st0
       if inp_a = "0" then
              next_a <= st2; -- st6
              activeFSM_a <= '0';
              o_i_a <= "00";
       elsif inp_a = "1" then
              next_a <= st1; -- st4
              activeFSM_a <= '0';
              o_i_a <= "00";
       else
              next_a <= st0; -- st0
              activeFSM_a <= '0';
              o_i_a <= (others => 'X');
       end if;
when st1 =>  --st4
       if inp_a = "0" then
              next_a <= st2; -- st6
              activeFSM_a <= '0';
              o_i_a <= "00";
       elsif inp_a = "1" then
              next_a <= st2; -- st6
              activeFSM_a <= '0';
              o_i_a <= "10";
       else
              next_a <= st1; -- st4
              activeFSM_a <= '0';
              o_i_a <= (others => 'X');
       end if;
when st2 =>  --st6
       if inp_a = "0" then
              next_a <= st0; -- st0
              activeFSM_a <= '0';
              o_i_a <= "01";
       elsif inp_a = "1" then
              next_a <= st1; -- st2
              activeFSM_a <= '1';
              o_i_a <= "01";
       else
              next_a <= st2; -- st6
              activeFSM_a <= '0';
              o_i_a <= (others => 'X');
       end if;
when others =>
              next_a <= st0; -- reset state
              activeFSM_a <= '0';
              o_i_a <= (others => 'X');
end case;
end process next_state_process_a;

next_state_process_b: process (machine_b, inp_b)
begin
case machine_b is
when st0 =>  --st1
       if inp_b = "0" then
```

```
                next_b <= st3; -- st5
                activeFSM_b <= '1';
                o_i_b <= "00";
        elsif inp_b = "1" then
                next_b <= st2; -- st6
                activeFSM_b <= '0';
                o_i_b <= "10";
        else
                next_b <= st0; -- st1
                activeFSM_b <= '1';
               o_i_b <= (others => 'X');
        end if;
when st1 =>  --st2
        if inp_b = "0" then
                next_b <= st3; -- st5
                activeFSM_b <= '1';
                o_i_b <= "00";
        elsif inp_b = "1" then
                next_b <= st2; -- st3
                activeFSM_b <= '1';
                o_i_b <= "00";
        else
                next_b <= st1; -- st2
                activeFSM_b <= '1';
               o_i_b <= (others => 'X');
        end if;
when st2 =>  --st3
        if inp_b = "0" then
                next_b <= st3; -- st5
                activeFSM_b <= '1';
                o_i_b <= "00";
        elsif inp_b = "1" then
                next_b <= st0; -- st1
                activeFSM_b <= '1';
                o_i_b <= "00";
        else
                next_b <= st2; -- st3
                activeFSM_b <= '1';
               o_i_b <= (others => 'X');
        end if;
when st3 =>  --st5
        if inp_b = "0" then
                next_b <= st0; -- st0
                activeFSM_b <= '0';
                o_i_b <= "10";
        elsif inp_b = "1" then
                next_b <= st1; -- st2
                activeFSM_b <= '1';
                o_i_b <= "10";
        else
                next_b <= st3; -- st5
                activeFSM_b <= '1';
               o_i_b <= (others => 'X');
        end if;
```

```
when others =>
              next_b <= st0; -- reset state
              activeFSM_b <= '1';
              o_i_b <= (others => 'X');
end case;
end process next_state_process_b;

-- select the output (multiplexor)
output_process: process (activeFSM, o_i_a, o_i_b)
begin
       if activeFSM = '0' then
              sal <= o_i_a;
       else
              sal <= o_i_b;
       end if;
end process output_process;
end behave;
```

Apéndice F:

Utilización de la herramienta de estimación de consumo Xpower

F.1. Introducción

Xilinx incorpora desde la versión 4.1 de su software de diseño ISE (*Integrated Software Enviroment*) la herramienta de estimación de consumo XPower [Xpo04]. Xpower es una herramienta de análisis de consumo *post Place & Route* para FPGAs (y *post-fit* en CPLDs) donde se puede analizar interactivamente el consumo. Incluye una herramienta con interfaz gráfica (GUI - *Grafical User Interface*) denominada Xpower y una aplicación de proceso por lotes (*batch*) Xpwr.

Esta herramienta utiliza como entrada ficheros de simulación con formato VCD (*Value Change Dump*) creados con la herramienta Modelsim [Men04] para calcular los estímulos en cada nodo del circuito. La información final es presentada en formato HTML o texto plano ASCII. Según Xilinx, Xpower estima el consumo con una precisión +/- 10 %.

F.2. Uso de Xpower con FPGAs

Xpower se puede invocar desde el entorno ISE, donde se carga el archivo NCD (*Native Circuit Description*) conteniendo el diseño, opcionalmente el fichero PCF (*Physical Constraints File*) con las restricciones del usuario ó las generadas por el *mapper* y

por último, los datos de una simulación previa. Se pueden además agregar la tensión de alimentación y la temperatura ambiente.

La opción de utilizar la interfaz gráfica (figura F.1) es útil por el corto tiempo de aprendizaje, no obstante para realizar múltiples mediciones y comparaciones la herramienta de procesamiento por lotes (Xpwr) es más adecuada.

En vez de colocar los datos de simulación, se puede colocar explícitamente la actividad que uno cree conveniente en cada nodo, ó bien colocar valores por defecto (12 % de actividad) o estimar los valores de actividad (*Estimate Activity Rates* en el menú *Tools*). Ninguna de éstas alternativas es aconsejable, dado que los valores de resultado que brinda no tienen nada que ver con la actividad real ya que no se tiene en cuenta la tremenda influencia de los *glitches* en las FPGAs. Por ejemplo, en un multiplicador de 16 bits pueden existir nodos internos permutando más de 8 veces la frecuencia de reloj, mientras que, otros nodos solo tienen una actividad del 5 % respecto de la señal de reloj.

F.2.1 Generación del fichero VCD.

El fichero que recoge los movimientos del circuito se llama VCD (*Value Change Dump* – Volcado del cambio de valores). El fichero VCD automatiza la anotación del movimiento de las señales. Normalmente se parte de un *testbench* significativo del funcionamiento del circuito del cual se extraen los movimientos de los datos internos. Los simuladores soportados por Xpower son el Model Technology ModelSim [Mod04] (en sus versiones XE, PE, and SE) y Cadence Verilog XL; Cadence NC-Verilog; Cadence NC-VHDL; Cadence NC-SIM [Cad04]. En esta sección solo se aborda el uso de Modelsim.

La lista completa de los comandos para el fichero VCD se puede ver en [Mod03b]. Los comandos necesarios en el flujo VHDL es el ingreso de forma interactiva o bien en un fichero de comandos (*.do file*) de los siguientes comandos (estos comandos se pueden utilizar en un flujo Verilog):

```
vcd file my_design.vcd
vcd add testbench/uut/*
```

Los comandos anteriores generan un fichero VCD llamado "my_design.vcd". Asumiendo que el nombre de la entidad del banco de pruebas, es "testbench" y que el

nombre de instancia de la unidad a simular es uut (*unit under test*), la segunda línea de comandos indica agregar todas las señales al archivo VCD.

Se puede agregar el modificador `-r` al comando "`vcd add`" para agregar todas las señales en un nivel de lógica mayor que uno (que es el valor por defecto). El uso del modificador –r puede aumentar considerablemente el tamaño del fichero de simulación, pero naturalmente será mucho más preciso.

A partir de la versión 6.1 del entorno de desarrollo del ISE [Ise03], se incorpora la opción de generar automáticamente el fichero VCD para un *testbench*. En las propiedades de la simulación *post place & route* se agrega esta opción para cualquier *testbench* ya sea Verilog o VHDL.

El tamaño de los ficheros generados suelen se muy grandes, dependiendo básicamente del tiempo de simulación y el tamaño del circuito. Simular un multiplicador de 32x32 bits con datos de entrada cada 100 ns durante 4 µs (40 multiplicaciones) requiere unos 13 MB. Es fácil de imaginar que simular el orden de segundos en un diseño medio ocupará en el orden de los GB. El tiempo de simulación es un tema nada despreciable para diseños grandes con bancos de pruebas importantes.

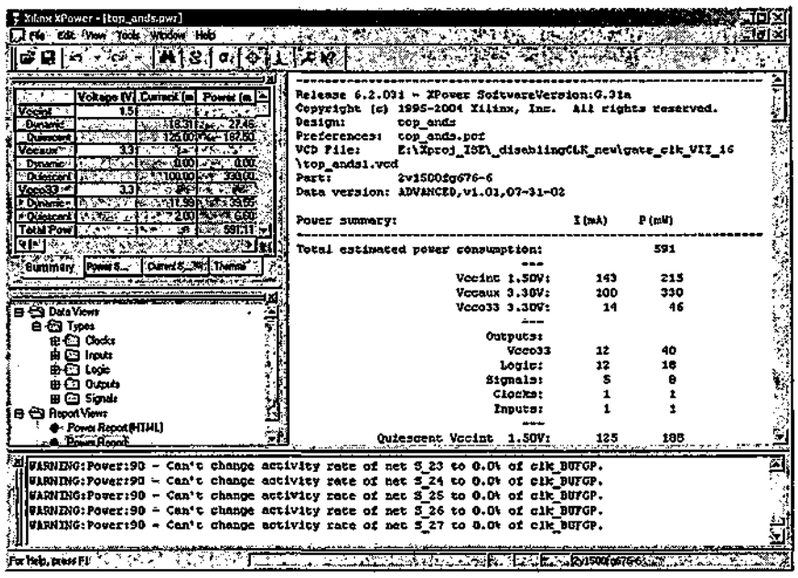


Figura F.1. Aspecto de la interfaz gráfica Xpower.

F.2.2 Uso de la interfaz gráfica Xpower

Se puede invocar independientemente (*stand alone mode*) en cuyo caso se deberá cargar posteriormente el diseño, ó desde el entorno ISE. En este último caso se cargará el fichero NCD y PCF correspondiente al diseño activo. De existir un fichero VCD con el mismo nombre del diseño se cargará también los movimientos allí descritos. Si no se ha cargado el fichero VCD, se lo puede cargar desde la barra de menús "*file → open simulation file…*".

Otra opción para introducir los movimientos es cargar un fichero XML (*extended markup Language*) con un resumen de los movimientos de cada señal y otras informaciones como tensión de alimentación, temperatura ambiente, etc. "*file → open settings file…*". Este fichero, que resume la actividad en cada nodo, es generado por una previa ejecución de Xpower "*file → save settings file as…*".

Otros datos que se pueden agregar interactivamente son la tensión de alimentación, la capacidad de la batería, la temperatura ambiente, cambiar el empaquetado y agregar ventilación. Finalmente, se pueden ver los informes en formato texto o HTML presionado en "*report view*". Al margen de la visualización en la ventana a tal efecto generará un fichero en texto plano con la extensión *xpw* (o en formato html) en el directorio de trabajo del proyecto de Xilinx.

El tipo de reporte puede ser seleccionado entre los tipos (*standard, detailed ó advanced*) a través del menú "*edit → preferences*". En el modo *advanced* informará la corriente y potencia máxima de cada fuente de alimentación, así como la corriente estática. Información de la corriente máxima disipable por el dispositivo dependiendo de las condiciones de temperatura ambiente y refrigeración antes suministradas. También informará las señales, elementos de lógica, entradas y salidas que más consumen (la cantidad se define en las preferencias).

F.2.3 Uso de la línea de comando Xpwr

La herramienta de línea de comandos Xpwr es útil a la hora de sistematizar medidas repetitivas con procesos por lotes. El ejecutable está en la carpeta de instalación de Xilinx. Ejecutando xprw –h se obtiene la lista de comandos (Figura F.2). Un ejemplo de línea de comando utilizado en está tesis es:

```
xpwr my_design.ncd  my_design.pcf -v -a -s my_design.vcd -o report.pwr
```

Comando	Acción
-v [-a]	Modo verboso, [-a] avanzado
-l <limit>	Número máximo de lineas en el reporte
-x <userdata>	Lectura de un fichero XML previamente guardado
-wx <userdata>	Grabar un fichero de movimientos XML.
-s <simdata>	Especifica el fichero de simulación (fichero VCD)
-tb <num> [unit]	Unidades en el fichero de reportes basados en tiempo. Las unidades son ps (por defecto), ns, fs y us.
-o <reportfile>	Especifica el nombre del fichero de salida
-t <tclscript>	Especifica un script TCL para aplicar los settings

Figura F.2. Opciones en la herramienta de línea de comando Xpwr.

F.3 *Scripts* para el uso del Xpower

En el punto F.2.2 se mencionó la utilización de la herramienta de línea de comandos xpwr y sus opciones de línea de comando. No obstante, es útil no solo invocar a la herramienta Xpwr sino también a las herramientas de xilinx para generar un modelo simulable del diseño y al simulador (en este caso modelsim) para generar de forma automática el (los) fichero(s) VCD. También, dependiendo del caso, se puede llegar a independizar totalmente del entorno ISE e invocar a las herramientas de implementación de las FPGAs desde línea de comandos.

F.3.1 *Script* para la implementación del diseño

La implementación de un diseño implica llamar al sintetizador (en este caso XST [Xst03]), luego al *ngdbuild* [Ngd04] que traduce los ficheros EDIF y NGC en un fichero NGD con el diseño. Posteriormente, el *map* [map04] traduce el NGD en componentes específicos del dispositivo. Mas tarde, la herramienta *par* [Par04] realiza el emplazado y rutado. La herramienta *trce* [Trc04] realiza un análisis estático de tiempos generando un fichero con extensión *twr*, y por último, *bitgen* [Bit04] genera el fichero .bit para programar el dispositivo.

El *script* de la figura F.3 espera como parámetros el nombre del diseño así como el dispositivo donde se ha de implementar. Por ejemplo:

implement my_design xc2v1500-fg676-6

Script 1: implement (%1 es nombre diseño, %2 dispositivo)

```
xst -intstyle ise -ifn __projnav/%1.xst -ofn %1.syr
ngdbuild -intstyle ise -dd %1/_ngo -i  -p %2 %1.ngc %1.ngd
map -intstyle ise -p %2 -cm area -k 4 -c 100 -tx off -o
%1.ncd %1.ngd %1.pcf
par -w -intstyle ise -ol std -t 1 %1.ncd %1.ncd %1.pcf
trce -intstyle ise -e 3 -l 3 -xml %1 %1.ncd -o %1.twr %1.pcf
bitgen -intstyle ise -f %1.ut %1.ncd
```

Figura F.3. Script de implementación de un diseño.

F.3.2 *Script* para la simulación y generación del fichero VCD

Para realizar la simulación *post place & route*, primero se debe generar el fichero de simulación con la herramienta *netgen* [Net04] (en antiguas versiones eran 3 herramientas separadas: *ngd2ver*, *ngd2vhdl* y *ngdanno*). El modelo de simulación generado puede ser VHDL o Verilog, y adicionalmente esta herramienta genera un fichero SDF (*Standard Delay Format*) necesario en la simulación *place & route*. Desde el punto de vista teórico da lo mismo simular el modelo VHDL o Verilog, sin embargo se han encontrado diferencias en los resultados en las últimas versiones del software. El ejemplo de la figura F.4 genera el modelo de simulación Verilog.

Script 2: gen_vcd (%1 es el nombre diseño)

```
netgen -intstyle ise -s 4 -pcf %1.pcf -ngm %1.ngm -w -ofmt
verilog -sim %1.ncd %1_timesim.v
cmd /C vsim ..\script_sim.do
xpwr %1.ncd  %1.pcf -v -a -s %1_1.vcd -o %1_1.pwr -wx %1_1.xml
...
xpwr %1.ncd  %1.pcf -v -a -s %1_n.vcd -o %1_n.pwr -wx %1_n.xml
```

Figura F.4. Script de simulación e invocación a xpwr.

Tras esto, se debe invocar al simulador con el comando *vsim* [Mod04], con un fichero de comandos para el simulador (extensión .do). Un ejemplo de *script* para el simulador se muestra en la Figura F.5. El ejemplo carga primero un *package* en VHDL, luego el modelo del diseño *post place& route* (en verilog la carga del fichero SDF es implícita en tanto que en VHDL se debe hacer explícitamente), y por último se carga un fichero

testbench en VHDL. El comando *vsim* carga la simulación con precisión de 1 ps. El archivo de *testbench* del ejemplo lee ficheros de textos que estimulan el circuito, por ello a continuación se hace una llamada a la línea de comando (*cmd*) para copiar un fichero de texto al directorio actual. Luego se define el fichero VCD y se agregan las señales para luego simular hasta el final. El ejemplo continua reiniciando la simulación y ejecutando nuevamente el ciclo con otro grupo de valores.

Una vez ejecutado el simulador y generado el (los) fichero(s) VCD se puede invocar el xpwr con las opciones descritas en F.2.3.

Script 3: script_sim.do (escript para modelsim)

```
vcom -93 -explicit  mipack.vhd
vlog  "C:/Xilinx/verilog/src/glbl.v"
vlog  design_timesim.v
vcom -93 -explicit mi_test.vhd
vsim -t 1ps +maxdelays -L simprims_ver -lib work testbench glbl

exec cmd /c copy ..\\test1.txt test.txt
vcd file design_1.vcd
vcd add testbench/uut/*
run -all

restart -f
exec cmd /c copy ..\\test2.txt test.txt
vcd file design_2.vcd
vcd add testbench/uut/*
run -all
quit -force
```

Figura F.5. Script para el simulador.

F.4 Cometarios y conclusiones

La herramienta de estimación de consumo Xpower brinda una interesante vía para la estimación del consumo, no obstante desde el punto de vista práctico en diseños industriales plantea varios inconvenientes. Para un diseño normal del orden de millón de puertas, simulado por algunos milisegundos de tiempo real puede requerir tiempos de simulación del orden de horas y ficheros VCDs de decenas de *Gigabytes*. Añadido a esto, realizar un *testbench* de un circuito complejo lo suficientemente representativo de

la actividad media puede ser una tarea excesivamente complicada. En esta línea existen trabajos para poder predecir la actividad de circuitos complejos [Tod04] y de este modo evitarse el tiempo de diseño de *testbenches* significativos, las horas de simulación y los *Gigabytes* de datos intermedios

Otro aspecto a destacar de esta herramienta, es que es una herramienta en constante evolución y se puede observar que los datos informados por la herramienta sufren importantes cambios de versión a versión. En esta tesis se ha evaluado la herramienta desde la versión 4.1 del ISE [Xpo02] con modelsim 5.6 [Mod01], hasta la 6.2 [Xpo04] con modelsim 5.8 [Mod04], pasando por todas las intermedias y midiendo datos concretamente con la que acompaña a la versión 5.2 [Xpo03a] (modelsim 5.7 [Mod03]) observándose importantes diferencias en los resultados.

Más aun, en la última versión del software evaluada [Xpo04] se han encontrado diferencias en los resultados generando modelos post *place & route* en VHDL o Verilog.

A lo largo de esta tesis se presentaron algunos resultados comparativos con mediciones realizadas con los arreglos experimentales de los apéndices B y C utilizando los mismos valores de excitación con resultados bastante dispares. Una evaluación definitiva necesita de más casos de estudio

F.5 Referencias del apéndice

[Bit04] Xilinx inc, "Development System Reference Guide ver.6.X, Chapter 15: BitGen", January 2004, available at www.xilinx.com.

[Ise03] Xilinx inc, "ISE: Integrated Development Environment ver 6.1: What's New in Xilinx ISE 6.1i", 2003, available at www.xilinx.com.

[Map04] Xilinx inc, "Development System Reference Guide ver.6.X, Chapter 8: MAP", January 2004, available at www.xilinx.com.

[Mod02] Model Technologies "ModelSim® SE User's Manual Version 5.6e" October 2002, available at www.model.com

[Mod03a] Model Technologies "ModelSim® SE User's Manual Version 5.7d" May 2003, available at www.model.com

[Mod03b] Model Technologies "ModelSim SE Users Manual Ver.5.7d: chapt 15 - Value Change Dump (VCD) Files" May 03, available at www.model.com

[Mod04] Model Technologies "ModelSim® SE User's Manual Version 5.8c" March 2004, available at www.model.com

[Net04] Xilinx inc, "Development System Reference Guide ver.6.X, Chapter 23: NetGen", January 2004, available at www.xilinx.com.

[Ngd04] Xilinx inc, "Development System Reference Guide ver.6.X, Chapter 6: NGDBuild", January 2004, available at www.xilinx.com.

[Par04] Xilinx inc, "Development System Reference Guide ver.6.X, Chapter 10: PAR", January 2004, available at www.xilinx.com.

[Tod04] E. Todorovich, E. Boemo, F. Cardells, J. Valls, "Power Analysis and Estimation Tool integrated with XPOWER", *Twelfth ACM International Symposium on Field-Programmable Gate Arrays (FPGA'04)*. February 2004.

[Tra04] Xilinx inc, "Development System Reference Guide ver.6.X, Chapter 13: TRACE", January 2004, available at www.xilinx.com.

[Xil03b] Xilinx Inc, XST (Xilinx Synthesis Technologies) User Guide 4.0 distributed with ISE 6.X, available at www.xilinx.com, June 2003.

[Xpo02] Xpower, "Xpower getting started", Release version 4.2.03i, 2002, available at support.xilinx.com.

[Xpo03] Xpower, "Xpower getting started", Release version 5.1.03i, 2003, available at support.xilinx.com.

[Xpo04] Xpower, "Xpower getting started" Release version 6.2.03i, 2004, available at support.xilinx.com.