Universidad Autónoma de Madrid

Técnicas para la Reducción de Consumo en FPGAs

Proyecto de Investigación

Autor: Gustavo Sutter
Director: Eduardo Boemo Scalvinoni

Escuela Politécnica Superior
Junio de 2003

Resumen:

El uso de PLDs (*Programmable Logical Devices*) y específicamente las FPGAs (*Field Programmable Gate Arrays*) ha aumentando rápidamente debido a la posibilidad de realizar prototipos con tiempos de desarrollo reducidos y bajo costo. En tanto el consumo de potencia y la disipación desde hace años son un factor importante en el diseño de VLSI. Por otra parte el estudio del consumo y las técnicas de optimización en el área de lógica reprogramable esta aun muy por debajo de los requerimientos del mercado.

En este trabajo se observan las principales técnicas de reducción de consumo en la tecnología VLSI y su posible aplicación en FPGAs. Mucho se ha publicado sobre bajo consumo en diferentes tecnologías, y el objetivos aquí es clasificar cuales se adaptan mejor al marco técnologico

Se realiza una clasificación de las técnicas de reducción de consumo y realizan experimentos sobre ellas. Para el estudio de las técnicas de reducción de consumo se construyen circuitos y utilizan arreglos experimentales donde se miden los parámetros físicos del consumo.

El trabajo se sustenta sobre un exhaustivo trabajo experimental, en el cual se han validado diferentes circuitos utilizando la tecnología de Xilinx. Esto permite identificar las técnicas que mejor se adaptan a una FPGA con LUTs tipo k=4.

Índice

Resumen:.....3
Índice.....4
Lista de Figuras.....11
Lista de Tablas.....15
Lista de Acrónimos.....17

Capítulo 1: Introducción y Motivación

1.1 Reducción y Estimación del Consumo.....19
1.2 Marco tecnológico del proyecto.....20
1.3 Selección de Técnicas de Reducción de Consumo.....20
1.4 Organización de está Memoria.....21

Capítulo 2: Consumo en circuitos CMOS

2.1. Consumo en Circuitos CMOS.....23
2.1.1 Consumo Estático.....24
2.1.2 Consumo Dinámico.....25
2.2. Espacio de diseño para bajo consumo.....26
2.2.1 Voltaje de alimentación.....26
2.2.2 Capacidad Física.....27
2.2.3 Actividad del Circuito.....29
2.3. Temas destacados en el diseño para bajo consumo.....31
2.4. Nivel de Proceso y Tecnología.....32
2.4.1 Encapsulado (packaging).....32
2.4.2 Proceso de Fabricación.....32
2.4.2.1 Optimización de la Tensión Umbral (Threshold Voltaje-Vt).....32
2.4.2.2. Reducción en la tecnología (Technological scaling).....33

2.4.2.3 Trazado (Layout) ... 33
2.4.2.4 Lógica dinámica (Dynamic logic) ... 34
2.4.2.5 Dimensiones de los transistores (Transisitor sizing) ... 35
2.5. Nivel de implementación ... 36
2.5.2 Descomposición tecnológica y mapeo ... 36
2.5.3 Reordenar las entradas ... 36
2.5.4 Reducción de Glitches ... 37
2.5.5 Concurrencia y redundancia ... 38
2.6. Nivel de Arquitectura y Sistema ... 39
2.6.1 Procesamiento concurrente ... 39
2.6.1.1 Paralelismo ... 39
2.6.1.2 Pipeline ... 40
2.6.2 Manejo de Potencia (Power Managemet) ... 41
2.6.3 Particionado ... 42
2.6.4 Representación de los datos ... 42
2.7. Nivel Algorítmico ... 44
2.7.1. Algoritmos para bajo consumo ... 44
Complejidad ... 44
Precisión ... 44
Regularidad ... 45
2.7.2 Algoritmos para arquitecturas de bajo consumo ... 45
Concurrencia ... 45
Modularidad y localidad ... 45
2.8. Nivel Sistema ... 46
2.8.1 Optimización de memoria ... 46
2.8.2 Particionado Hardware-Software ... 46
2.8.3 Optimización a nivel de instrucciones ... 46
2.8.4 Manejo dinámico del consumo ... 46
2.8.5 Minimización del consumo en las interfaces ... 47
2.9. Otros conceptos en el Tratamiento del Consumo ... 47
2.9.1. Energía vs. Potencia. ... 47
2.9.2. Proyecciones para la tecnología de semiconductores ... 48

2.10. Resumen del Capítulo 49

Capítulo 3: Reducción de Consumo en Máquinas de Estado

3.1. Definición de Máquina de Estados Finitos 51
3.1.1 Procedimiento para el diseño de Máquinas de Estados Finitos 52
3.1.2 Máquina de Mealy y Moore 52
3.1.3 Diagrama de Estados para Máquinas de Mealy y Moore 53
3.1.4 Estados, Transiciones y Salidas en las Máquinas de Mealy y Moore 53
3.1.5 Comparación de las Maquinas de Mealy y Moore. 54
3.1.5. Máquinas de Mealy Síncronas 55
3.2 Estrategias para reducción de consumo en Máquinas de Estados 56
3.2.1 Minimización de estados 56
3.2.2 Asignación de estados (state assignament/encoding) 56
Enfoques Tradicionales para la codificación de estados 56
Aproximaciones para asignación de estado de bajo consumo 56
Asignación de estados en FPGAs 57
3.2.3 Descomposición de maquinas de estado (FSM partitioning o descomposition) 57
3.3 Experimentos sobre codificación de Máquinas de Estados en FPGAs 58
3.3.1. Experimentos 58
3.3.2. Resultados Experimentales 60
3.3.3. Conclusiones sobre Codificación de Máquinas de Estados en FPGAs 63
3.4 Partición de Máquinas de Estado en FPGAs 64
3.4.1 Alternativas de Bajo Consumo en FSMs 64
3.4.2 Decomposición o partcionado de FSMs 65
Modelo General de Partición de Máquinas de Estado 65
Partición ortogonal de máquinas de estado 66
3.4.3 Técnicas de descomposición para bajo consumo 67
3.4.4 Una arquitectura de descomposición de FSMs para FPGAs 68
3.4.5. Cálculo de probabilidades de transición en un STG 69
3.4.6. Particionando la FSM en submáquinas 71

3.4.7 Métodos para bloquear los datos 71
3.4.8 Síntesis de la máquina de estados 71
3.4.9 Experimentos con partición de máquinas de estado 72
3.4.10 Resultados experimentales de la partición de máquinas de estado 73
3.5 Recomendaciones para la Reducción de Consumo en Máquinas de Estado Finitos 77

Capítulo 4: Experimentos a nivel topológico

4.1 Introducción 79
4.2 Relación entre Velocidad y Consumo en FPGAs 80
4.2.1 Circuitos de Prueba 81
4.2.2 Resultados Experimentales de la Relación Velocidad - Consumo 81
Correlación Velocidad-Consumo. Efecto de las Diferentes Topologías 81
Correlación Velocidad-Consumo para Implementaciones dentro de la Misma Topología 83
Importancia de los Glitches 84
Correlación Área-velocidad-consumo (Area-Time-Power) 85
4.2.3 Conclusiones de la Relación Velocidad - Consumo. 86
4.3 Conmutación de los Datos de Entrada (Propiedad Conmutativa y Diseño de Bajo Consumo) 87
4.3.1 Introducción 87
4.3.2 Circuitos de Pruebas Familia 4K 88
4.3.3 Resultados Experimentales Familia 4K 89
Esquema de Medición 89
Resultados 90
4.3.4 Circuitos de Pruebas Familia Virtex 93
4.3.5 Resultados Experimentales Familia Virtex 93
Relación Retardo-Consumo 94
Uso de Herramientas de Estimación de Consumo 94
4.3.6 Conclusiones de la Conmutación de Datos 96
4.4 Efecto de la Segmentación en el Consumo. 97
4.4.1 Medidas sobre XC4K 97
4.4.2 Mediciones sobre Virtex 98

Uso de herramienta de estimación de consumo 99
4.4.3 Conclusiones sobre la Segmentación 100
4.5 Observaciones en la Disminución del Consumo Registrando las Entradas y Salidas 102
4.5.1 Experiencias sobre la familia XC4K 102
4.5.2 Experiencias sobre la familia Virtex 103
4.5.3 Conclusiones del Registro de Entradas y Salidas 104
4.6 Conclusiones del Capítulo 105

Capitulo 5: Experimentos a Nivel Algorítmico

5.1 Introducción 107
5.2 Multiplicación Modular 107
5.2.1 Introducción 108
5.2.2 Algoritmos 108
A. Multiplicación y reducción (Multiply and Reduce) 108
B. Sumas y desplazamientos (Shift and Add) 109
C. Multiplicación de Montgomery 111
5.2.3 Detalle de la Síntesis 112
A. Multiplicación y reducción 112
B. Sumas y desplazamientos 113
C. Multiplicación de Montgomery 114
Comparación 114
5.2.4 Realización secuencial 116
5.2.5 Consumo de Potencia 117
Implementaciones combinacionales 117
Implementaciones secuenciales 118
5.2.6. Conclusiones Multiplicación Modular 118
5.3. Sumadores de alta velocidad. 119
5.3.1 Introducción 119
5.3.2 Sumador Ripple-Carry 119
5.3.3 Sumador Carry-Skip 120

5.3.4 Resultados Experimentales del Sumador Carry-Skip 122
Resultados en Área - Velocidad 122
Resultados en Consumo 124
Justificación de los Resultados en Consumo 125
5.3.5 Conclusiones Algoritmos Sumadores 126

Capítulo 6: Conclusiones y Futuros Trabajos

6.1 Conclusiones y Aportes 127
Consumo en CMOS y Clasificación de las Técnicas de Reducción de Consumo 127
Recomendaciones para la Reducción de Consumo en FSMs 128
Consejos a Nivel Topológico 128
Resultados a Nivel Algorítmico 130
Conclusiones Multiplicación Modular 130
Conclusiones Algoritmos Sumadores 131
Conclusiones Herramientas de Estimación de Consumo 131
6.2 Futuros Trabajos 131
6.3 Publicaciones Relacionadas con éste Trabajo 132

Capítulo 7: Referencias

Apéndice A: Placa de prueba XC4000

A.1. Introducción 145
A.2 Características de las placas de prueba 145
A.3 Conexión de la placa de pruebas 148
A.4. Método de Medición de Consumo 149
A.5. Generador de vectores 150

Apéndice B: Placa de prueba AFX

B.1. Introducción 151

B.2 Características de la Placas de Prueba AFX 151

B.3 Conexión de la Placa de Pruebas 152

B.4 Metodología de Medición de Consumo 154

Apéndice C: Traductor Kiss2VHDL

C.1. Introducción 155

C.2 El formato KISS 155

C.3 Bancos de pruebas (Benchmarks) 156

C.3 Traductor Kiss2VHDL 157

Apéndice D: Particionador de máquinas de estado: Part_FSM

Figura D.1. Declaración de la parte privada de la clases state y transition. 162

Figura D.2. Código para la partición de máquinas de estados. 163

Figura D.3. Diagrama de transición de estados de la máquina de estados DK27. 164

Figura D.4. a) Código KISS2 de entrada. b) Información de la herramienta con las probabilidades calculadas. 164

Figura D.5. Información de las particiones generadas. 165

Figura D.6. Partición generada para el circuito dk27 165

Lista de Figuras

Figura 2.1. a) Transistor CMOS b) Esquema lógico de funcionamiento del inversor CMOS 23
Figura 2.2. Consumo estático en un inversor CMOS 24
Figura 2.3. Consumo dinámico en un cambio de estado del inversor 25
Figura 2.4. Relación energía - voltaje y retardo - voltaje respectivamente 26
Figura 2.5. Capacidad en los dispositivos CMOS 27
Figura 2.6. Capacitancias de interconexión 29
Figura 2.7. Interpretación de la actividad en circuitos síncronos: Glitches 30
Figura 2.8. Jerarquía en el espacio de diseño 31
Figura 2.9. Esquema de la lógica dinámica 34
Figura 2.10. Compuerta NAND de 3 entradas en lógica estática y dinámica 35
Figura 2.11. Dependencia del orden de entrada para la reducción de actividad 36
Figura 2.12. Dependencia del orden de entrada para la reducción de actividad 37
Figura 2.13. Estructuras de cascada y balanceadas 38
Figura 2.14. Reducción de voltaje y paralelismo para bajo consumo 40
Figura 2.15. Reducción de voltaje y pipeline para bajo consumo 41
Figura 2.16. Ejemplos de cambio de signo en complemento a dos (C2) y signo magnitud (SVA).. 43
Figura 2.17. Gráfica del consumo y la energía 47
Figura 2.18. Características típicas del consumo en microprocesadores. 48
Figura 2.19. Proyecciones para la industria de semiconductores. 49
Figura 3.1. Máquina de estados de Moore 53
Figura 3.2. Máquina de estados de Mealy 53
Figura 3.3. Diagrama de estados para el controlador de la ruta de datos del MCD. 54
Figura 3.4. Maquina de estados de Mealy Síncrona 55
Figura. 3.5. Código KISS y resumen del código generado. 58
Figura 3.6. Ahorro de consumo en función de la cantidad de estados. a) One-Hot respecto de Binario. b) One-Hot respecto de "Out-oriented" 61
Figura 3.7. Consumo por nro de estados: a) Binary; b) One-Hot; c) Out-oriented; d) Two-Hot.... 61
Figura 3.8. Relación retardo- consumo para las FSMs 62
Figura 3.9. Relación área – consumo en las máquinas de estados 62

Figura 3.10. Descomposición de Máquinas de estados. A) paralela. B) Cascada. C) General 64

Figura 3.11. Diagrama de estados de una máquina de estados y diagramas de las particiones en la aproximación tradicional. 65

Figura 3.12 Diagrama de estados y partición ortogonal de una FSM 66

Figura 3.13. Arquitectura I para descomposición de máquinas de estado en FPGAs. 67

Figura 3.14. Arquitectura II para descomposición de máquinas de estado en FPGAs..... 68

Figura 3.15 Diagrama de transición de estados y grafos de probabilidades..... 70

Figura 3.16. a) Diagrama de transición de estados (State Transition Graph - STG), b) Probabilidad estática y probabilidad total de transición..... 70

Figura 3.17. Gráfica de frecuencia, donde se observa el impacto negativo de los registros de bloqueo..... 75

Figura 3.18. Área y retardo de la partición ortogonal respecto de las arquitectura 1 y 2..... 75

Figura 3.19. Consumo de la partición ortogonal respecto de las arquitecturas 1 y 2 76

Figura 4.1. Relación ancho de banda-consumo para multiplicadores segmentados [Boe96]..... 80

Figura 4.2. Consumo dinámica de los multiplicadores medidos..... 82

Figura 4.3. Relación velocidad-consumo para la misma topología (multipl. VHDL comportamental) 82

Figura 4.4. Relación velocidad-consumo dentro de la misma topología para el multiplicador Guid. 83

Figura 4.5. Estudio de las fluctuaciones en el consumo para circuitos de similares velocidades (Multiplicador Guid). 83

Figura 4.6. Transiciones a la salida de los multiplicadores seleccionados. Obtenida por Simulación 84

Figura 4.7. Transiciones a la salida de los multiplicadores seleccionados por Medición. 84

Figura 4.8. Relación entre actividad a la salida de los multiplicadores y consumo para implementaciones con similar velocidad..... 85

Figura 4.9. Relación entre actividad a la salida de los multiplicadores y consumo para implementaciones con similar velocidad..... 86

Figura 4.10. Arreglo segmentado Hatamian-Cash de 4 bits con entradas b locales y a globales..... 88

Figura. 4.11. Reducción de consumo. Arrays Hatamian-Cash. XC4010PC84 90

Figura 4.12. Reducción de consumo en función de la profundidad de lógica de los circuitos. XC4010PC84..... 90

Figura 4.13. Efecto de las líneas globales. XC4010PC84..... 91

Figura 4.14. Reducción de consumo para diferentes topologías. XC4010PC84..... 92

Figura 4.15. Reducción de consumo para diferentes circuitos familia XC40XXPC84 (multiplicador Hatamian-8) 92

Figura 4.16. Relación retardo-consumo para los multiplicadores de 32 bits. 94

Figura 4.17. Diferencia en el consumo estimado y medido multiplicadores de 32 y 16 bits 96

Figura 4.18. Consumo normalizado respecto de la profundidad lógica en XC4K 98

Figura 4.19. Consumo normalizado respecto de la profundidad lógica en Virtex 99

Figura 4.20. Consumo, área, retardo normalizados respecto de la profundidad lógica en Virtex 99

Figura 4.21. Consumo medido y estimado en función de la profundidad lógica. 100

Figura 4.22. Distribución de la línea global de reloj. a)Flip-flop en las patas de entrada salida; b) usando los Flip-Flop de los CLBs 102

Figura 4.23. 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 103

Figura 4.24. Consumo normalizado con registros en IOBs y CLBs. 103

Figura 5.1. Reductor modulo M. a) Celda elemental. b)Implementación combinacional. 113

Figura 5.2. Multiplicador shift and add. a) Celda de cálculo. b) arreglo combinacional. 113

Figura 5.3. Celda de cálculo del multiplicador de Montgomery 114

Figura 5.4. Esquema de la implementación secuencial del algoritmo de sumas y desplazamientos. 115

Figura 5.5. Celda que utiliza la cadena de acarreo en las FPGAs Xilinx Spartan2 / Virtex 120

Figura 5.6. Grupo sumador de s bits para el carry-skip 120

Figura 5.7. Multiplexores de salto de acarreo (carry-skip multiplexers) 121

Figura 5.8. Generación de la condición de propagación del acarreo. 122

Figura 5.9. Sumador carry-skip (n = 4.s) 123

Figura 5.10. Emplazamiento relativo para un Carry-Skip con N=128, S=16. 123

Figura 5.11. Estructura de los circuitos a medir. 124

Figura 5.12. Estructura de los acumuladores utilizados para medir el consumo 125

Figura A.1. Esquema de la placa de prueba de la familia XC4000. 147

Figura A.2. Esquema de conexión del arreglo experimental 148

Figura A.3. Fotografía del arreglo experimental 148

Figura A.4. Esquema del generador de patrones 150

Figura A.1. Esquema de la placa de prueba de la familia XC4000. 141
Figura A.2. Esquema de conexión del arreglo experimental 142
Figura A.3. Fotografía del arreglo experimental 142
Figura A.4. Esquema del generador de patrones 144
Figura B.1. Detalle de la placa de prototipado AFX 145
Figura B.2. Fotografía de la tarjeta AFX y de la interfaz con el puerto paralelo 146
Figura B.3. Fotografía del arreglo experimental 147
Figura B.1. Detalle de la placa de prototipado AFX 151
Figura B.2. Fotografía de la tarjeta AFX y de la interfaz con el puerto paralelo 152
Figura B.3. Fotografía del arreglo experimental 153
Figura C.1. Un primer ejemplo de descripción Kiss2 156
Figura C.2. Ejemplo grafo de transición de estados y su especificación en Kiss2 156
Figura D.1. Declaración de la parte privada de la clases state y transition. 162
Figura D.2. Código para la partición de máquinas de estados. 163
Figura D.3. Diagrama de transición de estados de la máquina de estados DK27. 164
Figura D.4. a) Código KISS2 de entrada. b) Información de la herramienta con las probabilidades calculadas 164
Figura D.5. Información de la particiones generadas. 165
Figura D.6. Partición generada para el circuito dk27 165

Lista de Tablas

Tabla 3.1. 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)..... 72

Tabla 3.2. Consumo de potencia expresado en mW/MHz para la partición de máquinas de estado. 73

Tabla 3.3. Área expresada en CLBs para la partición de máquinas de estado. 74

Tabla 3.4. Circuitos donde no se logra mejora en el consumo debido a la alta probabilidad de transición entre submáquinas de estados..... 74

Tabla 4.1. Principales características de los circuitos de prueba. Relación Velocidad-Consumo..... 81

Tabla 4.2. Principales características de los circuitos de prueba. 89

Tabla 4.3. Descripción de los vectores de prueba..... 89

Tabla 4.4. Actividad de las entradas, salidas y totales para la operación A □B y B□A con vec máximo toggle-8 91

Tabla 4.5. Principales características de los circuitos de prueba. Familia Virtex..... 93

Tabla 4.6. Diferencia de consumo A×B vs. B×A en multiplicadores de 32 bits. 93

Tabla 4.7. Diferencia de consumo A×B vs. B×A en multiplicadores de 16 bits. 94

Tabla 4.8.a Relación medidas – estimaciones para multiplicadores de 32 bits y secuencia MaxTog. 95

Tabla 4.8.b Relación medidas – estimaciones para multiplicadores de 32 bits y secuencia AvgTog. 95

Tabla 4.8.c Relación medidas – estimaciones para multiplicadores de 16 bits y secuencia MaxTog. 95

Tabla 4.8.d Relación medidas – estimaciones para multiplicadores de 16 bits y secuencia AvgTog. 95

Tabla 4.9. Principales características de las diferentes topologías de prueba XC4K. 97

Tabla 4.10. Principales características de los circuitos de prueba para Virtex. 98

Tabla 4.11. Consumo medido respecto del estimado para multiplicadores segmentados. 100

Tabla 4.12. Diferencia de consumo dinámico dependiendo del tipo de registros. 104

Tabla 5.1. Número de CLBs y Retardo Máximo (ns) para los multiplicadores modulares secuenciales..... 114

Tabla 5.2. Número de CLBs y Frecuencia máxima (MHz) para multiplicadores modulares secuenciales..... 117

Tabla 5.3 Área, retardo y consumo (Area-Time-Power) de los multiplicadores modulares combinacionales 117

Table 5.4. Área, retardo y consumo de los multiplicadores modulares secuenciales 118

Tabla 5.5. Resultados de la implementación de circuitos carry skip: Retardos en ns. 124

Tabla 5.6. Resultados de la implementación de circuitos carry skip: Penalidad en área. 124

Tabla 5.7. Consumo sumadores Carry-Skip. 125

Tabla A.1: Conexiones del analizador en la placas de prueba. 146

Tabla A.2: Otros pines importantes 146

Tabla A.3: Componentes del consumo y forma de medición 149

Tabla B.1. Conexiones del generador de patrones a la FPGA 153

Tabla B.2. Conexiones del generador de la FPGA al analizador lógico 153

Tabla B.3: Componentes del consumo y forma de medición en Virtex 154

Lista de Acrónimos

ADS	Aritmética de Dígitos en Serie
ASIC	Application-Specific Integrated Circuit
ATP	*Área-Time-Power*
C2	Sistema de representación de enteros de Complemento a Dos
CAD	Computer Aided Design
CDSP	Custom Digital Signal Processor
DCT	Discrete Cosine Transform
DSP	Digital Signal Processor
EDA	Herramienta de Diseño Asistido (CAD en inglés)
FA	Full Adder
fc	Frecuencia máxima de funcionamiento o de reloj
FF	Flip-Flop
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Description Language
IDE	*Integrated Development Environment*
LSI	Large-Scale Integration
LUT	Look-Up Table
MSB	Bit más significativo (Most Significant Bit)
MSD	Dígito más significativo
MSP	Multiplicador S/P de precisión simple
PCB	Printed Circuit Board
PDSP	Programmable Digital Signal Processor
PSC	Conversor paralelo/serie (Paralell Serie Converter)
r2	Coeficiente de determinación en el análisis de regresión lineal
RCA	Ripple Carry Adder
S/P	Serie/Paralelo
SPC	Conversor serie/paralelo
STG	State Transition Graph
STG	State Transition graph - grafo de transición de estados
STT	State Transition Table – Tabla de transición de estados
SVA	Signo Valor Absoluto
SVA	Sistema de representación de enteros de Signo Valor absoluto o signo y magnitud
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
VLSI	Very Large-Scale Integration

Capítulo 1:

Introducción y Motivación

El objetivo general de este trabajo es dar soluciones en el diseño de bajo consumo en FPGAs: conocer qué circuitos poseen mejores características de consumo, respetando características de área y velocidad. Sin embargo, ésta es una meta demasiado ambiciosa y alcanzarla con éxito requeriría la evaluación de multitud de operadores, variables, arquitecturas y algoritmos. Este trabajo no pretende ser más que una primera aproximación a dicho objetivo y, por ello, se ha limitado el estudio de la optimización de consumo en máquinas de estados y al análisis de alternativas a nivel topológico y algorítmico.

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 semicustom, a menudo sobrepasa los límites de disipación de los encapsulados. Actualmente, el consumo medio de potencia de un chip se sitúa entre 1 y 10 vatios, con una tendencia a incrementarse en los próximos años al menos un orden de magnitud.

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. Por otro lado, teniendo en cuenta que todas las causas de falla de los circuitos integrados crecen exponencialmente con la temperatura, la reducción del consumo aumenta la fiabilidad y vida del producto. 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.

Uno de los aspectos más importantes 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 en

un factor del 0,35% por ºC. En consecuencia, existe una relación oculta entre diseño de bajo consumo y diseño de alta velocidad, reforzada por el hecho de que la capacidad de cada nodo afecta tanto al consumo como al ancho de banda. Así, la optimización del primer parámetro usualmente produce mejoras en el segundo.

1.2 Marco tecnológico del proyecto

Se utilizarán circuitos reprogramables (*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 mercados que tradicionalmente eran exclusivos de tecnologías ASIC y DSPs.

El diseño en FPGA exige una metodología propia, en 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 axiomas 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 ejemplo, en las arquitecturas basadas en LUTs y registros, puede suceder que segmentaciones con diferentes granularidades se traduzcan en circuitos con la misma profundidad de lógica. Por todo esto, la viabilidad de algunos de los circuitos y métodos de diseño deben reconsiderarse en función de las características de la FPGA elegida.

La justificación del fabricante elegido se basa en varios hechos: Xilinx es el "inventor" de las FPGAs, fundada en 1984 es líder del mercado y su arquitectura basada en tablas de lookup de cuatro entradas es la más difundida entre otros fabricantes.

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 (diseño físico o layout), 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.

La selección de las técnicas parte de un primer estudio bibliográfico y un refinamiento posterior para adecuarlo al marco tecnológico. Una gran parte de las técnicas descriptas en las publicaciones son a nivel teórico con resultados obtenidos por simulaciones, en tanto existen otras desarrolladas ad-hoc para otras tecnologías.

1.4 Organización de está Memoria

Esta memoria se organiza de la siguiente manera: En el capítulo 2 se realiza una descripción de las causas del consumo en los circuitos CMOS así como un repaso a las principales técnicas de reducción de consumo. El capítulo 3 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 bajo consumo. El capítulo 4 describe diferentes experimentos a nivel topológico, estableciendo relaciones entre velocidad y consumo, probando los efectos de la propiedad conmutativa y la influencia de la profundidad lógica en el consumo. En el capítulo 5 se describen experimentos realizados sobre algoritmos para la multiplicación modular, así como para sumadores del tipo *Carry-Skip*. Finalmente el capítulo 6 resume los resultados y presenta futuros trabajos.

1.1 Reducción y Estimación del Consumo 19

1.2 Marco tecnológico del proyecto 20

1.3 Selección de Técnicas de Reducción de Consumo 20

1.4 Organización de está Memoria 21

Capítulo 2:

Consumo en circuitos CMOS y su efecto en FPGA's

A fin de introducir en el tema del bajo consumo, se describe en esta sección 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.

2.1. Consumo en Circuitos CMOS

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

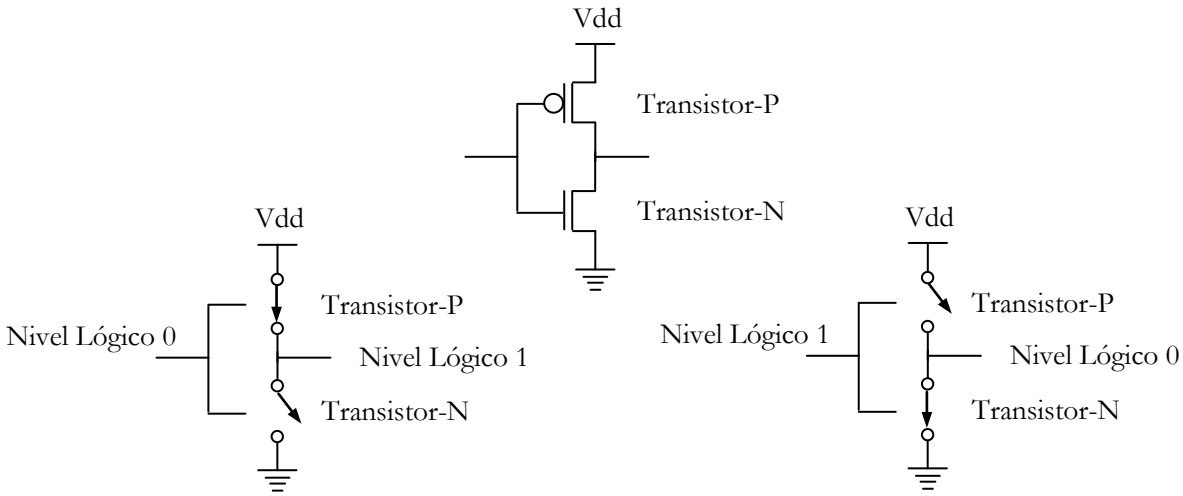


Figura 2.1. a) Transistor CMOS b) Esquema lógico de funcionamiento del inversor CMOS

El consumo estático se refiere a las corrientes que fluyen mientras no hay cambios de estado en los circuitos, esta es debido a las junturas PN, corrientes estáticas propias de los dispositivos y corrientes de fuga. El consumo dinámico que es el más importante en la tecnología CMOS depende de las cargas y descargas de las capacitancias en las transiciones.

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 2 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 voltaje en la puerta (*gate*) por debajo de la corriente de umbral

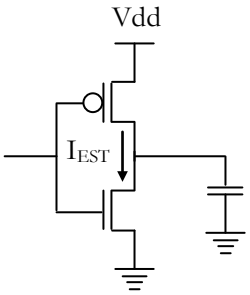


Figura 2.2. Consumo estático en un inversor CMOS

- Corriente standby (Standby Current) que es la que siempre fluye desde Vdd a tierra (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 el voltaje 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 V_{dd} y tierra. En general esta corriente es igual al producto entre V_{dd} y la corriente continua fluida entre la fuente y la tierra. La Potencia estática disipada (P_s) se puede modelar por la siguiente ecuación:

$$Ps = \sum (\text{corriente de fugas}) \times \text{Voltaje Suministrado} \qquad \text{(Ecuación 2.1)}$$

O bien como

$$P_s = V_{dd} \times I_{cf} \qquad \text{(Ecuación 2.2)}$$

Donde:

V_{dd} = Tensión de alimentación

I_{cf} = Corriente dentro del dispositivo (suma de corrientes de fugas como en ecuación 2.1)

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 tierra 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).

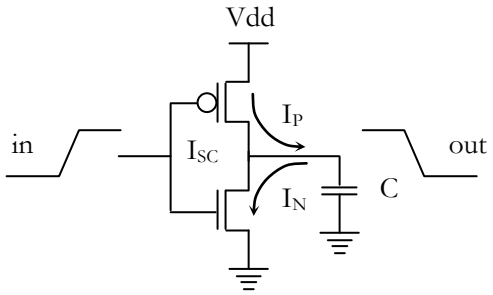


Figura 2.3. Consumo dinámico en un cambio de estado del inversor

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. La corriente máxima de cortocircuito fluye cuando no hay carga, esta corriente decrece con la carga.

Si se seleccionan los tamaños de modo que los tiempos de subida y bajada son aproximadamente iguales, el consumo por corriente de cortocircuito suele ser menor del 15% de la corriente dinámica [Vee84]. Tal lo expuesto, si bien no es despreciable, tampoco es un factor dominante en el consumo.

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:

$$P = \frac{1}{2} \cdot C_L \cdot V_{dd}{}^2 \cdot E(sw) \cdot f_{clk} \qquad \text{(Ecuación 2.4)}$$

Donde:

C_L = Capacitancia física a la salida de un nodo
V_{dd} = Tensión de alimentación
E(sw) = Actividad de conmutación (Switching activity), numero 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 es mucho menor que la dinámica, no obstante suele tener valores bastante mayores a los de un ASIC. Por el lado de la corriente estática se puede afirmar que depende de la tecnología del dispositivo, del tamaño (cantidad de bloques de cálculo, patas, etc). El porcentaje de ocupación no es desde el punto de vista del consumo estático relevante (ver experimentos capitulo 4). La corriente dinámica también aquí es proporcional al cuadrado de la tensión y a la frecuencia (ver experimentos).

2.2. Espacio de diseño para bajo consumo

Tal lo antedicho y expresado por la ecuación 2.4 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. Lamentablemente 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.

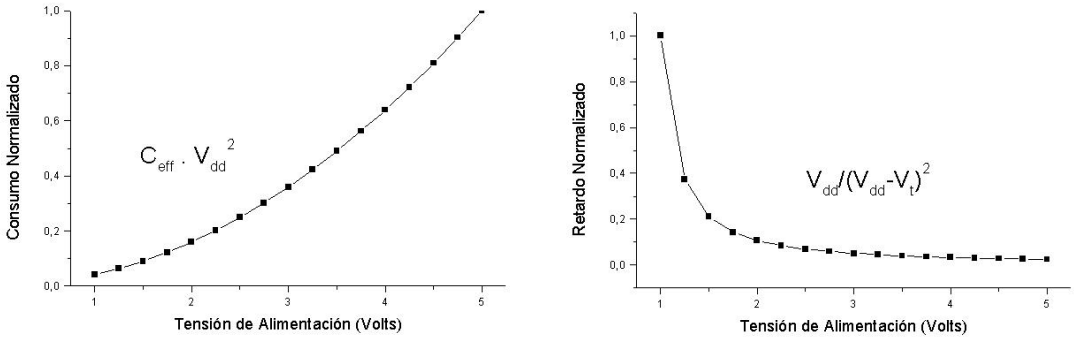


Figura 2.4. Relación energía - voltaje y retardo - voltaje respectivamente

2.2.1 Voltaje 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 (voltaje). Una simple reducción en un factor dos implica un ahorro de cuatro veces (figura 2.4.a) [lan94]. Muchos diseñadores están dispuestos a sacrificar capacidad en el circuito o actividad de los datos con tal de reducir el voltaje. Lamentablemente se paga un costo en velocidad muy alto cuando V_{dd} se acerca V_t (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:

$$T \propto V_{dd}/(V_{dd}-V_t)^2 \qquad \text{(Ecuación 2.5)}$$

La aproximación más atractiva a la hora de reducir voltaje sin pérdida de velocidad es la reducción de la tensión de umbral (V_t). El limite para la reducción de V_t esta dado por el margen de ruido

soportado y el aumento en la corriente de fuga de subumbral. Los límites tecnológicos para los circuitos CMOS a temperatura ambiente están alrededor de los 0.3 Volts de V_t.

Existen otros elementos limitantes para la reducción del voltaje fuera de la perdida de velocidad como son la compatibilidad e interoperabilidad con otros sistemas. La mayor parte de los componentes del mercado trabajan a 5V o 3,3 volts (más recientemente los estándares de 2,5 y 1,8 voltios) y a no ser que se diseñe un sistema completamente desde cero es altamente probable que el sistema deba comunicarse con otros componentes o sistemas los que operan a voltajes estándares. Este dilema es parcialmente solucionable con los conversores DC-DC de alta eficiencia que están apareciendo. Hoy en día existen muchos circuitos (sobre todo microprocesadores y otros con gran superficie dedicada al cálculo) que operan a una tensión menor que su entrada salida.

En cuanto a los circuitos de lógica programable, si bien existen familias operando a diferentes voltajes (5V y 3,3V tradicionalmente y los más nuevas como la familia Virtex a 2,5V y 1,8V), 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 voltajes [gar00], los que permiten medir la relación consumo tensión.

2.2.2 Capacidad Física

La reducción de la capacitancia ofrece una reducción lineal del consumo, de modo que la reducción es otra atractiva fuente de minimización. La capacidad en un circuito CMOS depende mayoritariamente de dos factores, por un lado los dispositivos y por otro las interconexiones.

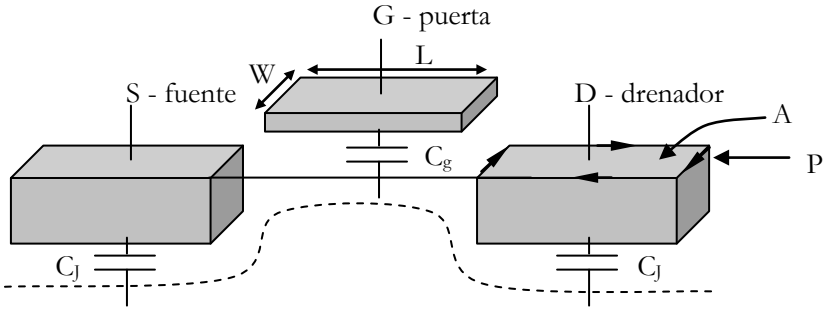


Figura 2.5. Capacidad en los dispositivos CMOS

Para los dispositivos las contribuciones más importantes provienen de los empalmes de las puertas. La capacidad más importante es la referida al oxido de la puerta (Gate), esta capacidad puede ser aproximada a la capacidad entre placas paralelas formada entre la capa y el substrato:

$$C_g = W \cdot L \cdot C_{ox} = W \cdot L \cdot \frac{\varepsilon_{ox}}{t_{ox}} \qquad \text{(Ecuación 2.6)}$$

Donde:

C_g = Capacidad debido a la puerta (G)
W = Ancho de la difusión de la puerta
L = Largo de la difusión de la puerta
C_{ox} = Capacidad por unidad de superficie (faradios/m^2)
ε_{ox} =

t_{ox} = Espesor del oxido

Adicionalmente tanto los empalmes de la fuente y el drenador contribuyen a la capacitancia total. Esta capacitancia tiene una componente dependiente del área y otra del perímetro y son no lineales con la tensión.

$$C_j(V) = A \cdot C_{j0}(1 - \frac{V}{\phi_0})^{-m} + P \cdot C_{jsw0}(1 - \frac{V}{\phi_0})^{-m} \quad \text{(Ecuación 2.7)}$$

Donde:

C_g = Capacidad debido a la fuente y drenador
A = Área de la fuente (F) o drenador (D)
V = Tensión de alimetación
P = Área de la fuente (F) o drenador (D)
C_{jo} = Capacidad de equilibrio para la pared inferior
C_{jo} = Capacidad de equilibrio para la pared lateral
ϕ_0 = Potencial de barrera del empalme
m = coeficiente de gradiente del empalme

Comúnmente esta capacidad no lineal es aproximada a la siguiente expresión.

$$C_{jeq} = \frac{\int_{V0}^{V1} C_j(V)dV}{V_1 - V_0} \quad \text{(Ecuación 2.8)}$$

Donde V_0 y V_1 son el rango de operación para el empalme.

En las tecnologías antiguas las capacitancias de los dispositivos dominaban sobre las interconexiones. Conforme las tecnologías siguen reduciéndose este deja de ser verdad y las capacitancias de las interconexiones empiezan a ser más importantes. Para las interconexiones, existe la capacitancia entre cada capa de metalización y el substrato, así como las capacitancias de acoplamiento entre las capas en si. Cada una de esas capacitancias posee dos componentes: Una componente de placas paralelas y una componente de borde.

$$C_w = W \cdot L \cdot C_p + 2 \cdot (W + L) \cdot C_f \quad \text{(Ecuación 2.9)}$$

Históricamente la capacidad de los planos paralelos, la que crece linealmente tanto con el ancho como con el largo de la pista, ha sido dominante. Las componentes de borde empiezan a ser significativas en tanto el ancho de las pistas se aproximan al grosor de las mismas [Bak90].

Entendiendo las fuentes de producción de capacitancias se puede considerar las formas de reducirlas. Claramente la forma más directa de reducción es a través de utilizar dispositivos mas pequeños y conexiones mas cortas. Naturalmente la optimización de las capacitancias no es una circunstancia independiente. Reduciendo el tamaño de los transistores no solo se reduce la capacidad si no que también la corriente manejada, con lo que el circuito opera más lento. La reducción del largo de las pistas es atractiva cuando se logra en base a una mejor distribución de los componentes.

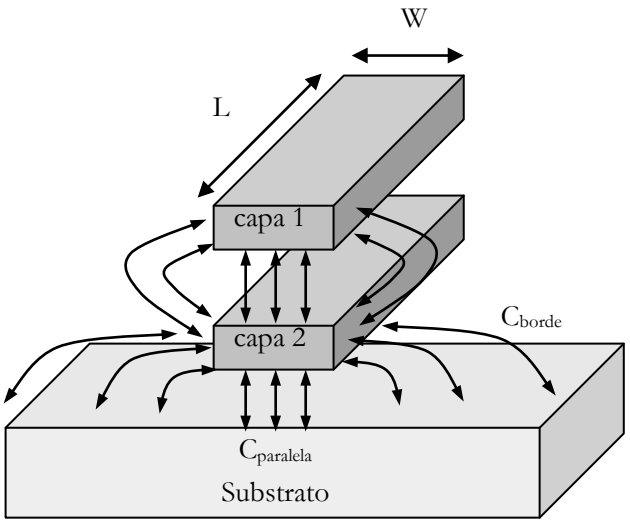


Figura 2.6. Capacitancias de interconexión

En las FPGAs es muy significativo la capacidad debido a las interconexiones (alrededor del 33% del consumo [gar00]) En ellas los canales de rutado son fijos y se debe pasar por uno o múltiples matrices de interconexión las que agregan importante cantidad de la capacitancia, consumo y retardo. La disminución del largo de pistas ofrece una importante fuente de ahorro de consumo en las arquitecturas con lógica programable. Herramientas optimizadas de emplazamiento y rutado (place and route) pueden ayudar en la disminución del consumo. Análogamente la disminución de elementos de cálculo puede generar un ahorro en la capacidad aunque se debería analizar el eventual aumento de recurso de rutado así como el aumento de actividad que esto podría provocar.

2.2.3 Actividad del Circuito

Al margen del voltaje y la capacidad la otra fuente de consumo dinámico es la actividad en el circuito. Un chip puede tener gran capacidad física, pero si no hay actividad el consumo será nulo. El concepto de actividad se refiere a cuan frecuentemente hay cambios de estado. Se puede considerar que la actividad del circuito depende de dos factores. Por un lado f_{ckl} 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. La actividad dentro de un multiplicador de 16 bits puede varias hasta 5 veces 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 sincrono y E(sw) que expresa cuantas transiciones se desarrollan en el lapso $1/f_{clk}$. 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 esta.

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 [Mar95].

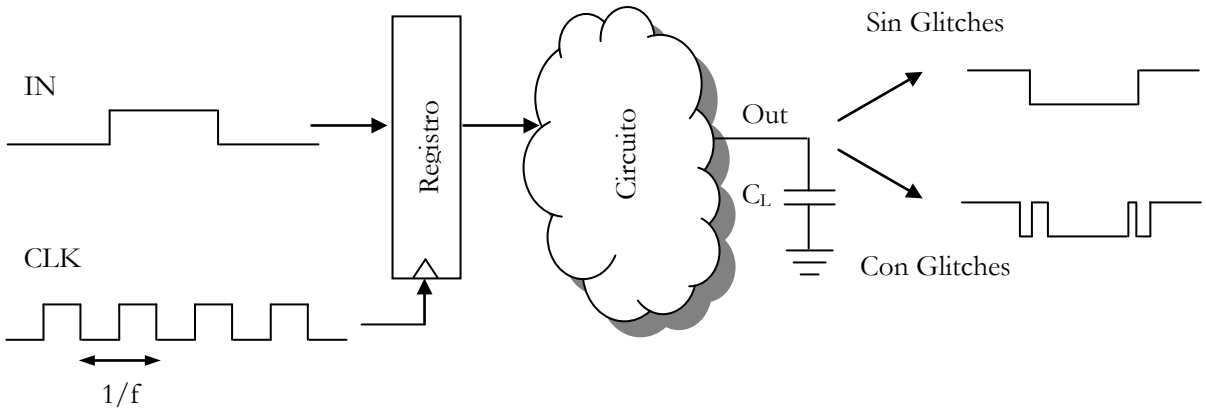


Figura 2.7. Interpretación de la actividad en circuitos síncronos: Glitches

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 periodo 1/f, transformado así la expresión del consumo en:

$$P = \frac{1}{2} \cdot C_L \cdot V_{dd}^{\ 2} \cdot E(sw) \cdot f_{clk} = C_{eff} \cdot V_{dd}^{\ 2} \qquad \text{(Ecuación 2.10)}$$

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.

Como en el caso del Voltaje 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 cilicio así como consecuentemente en capacidad del circuito. Como siempre el problema de optimización de las tres variables voltaje-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*.

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 simulación, ver sección experimental).

2.3. Temas destacados en el diseño para bajo consumo

La primera observación surge de la ecuación 2.4, donde se observa que el consumo es determinado por tres parámetros importantes como son: el voltaje, 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 del voltaje 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 (es decir quitar el reloj), 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 packaging (encapsulado) del circuito. En cuanto a las FPGAs estos factores ya son dados y existe poco por hacer, mas allá de elegir ciertos circuitos dentro de la gama de oferta de circuitos reconfigurables, los eventualmente operen a menor tensión, utilicen un proceso de fabricación más moderno, etc.

2.4.1 Encapsulado (packaging)

Generalmente una fracción importante del consumo total puede ser atribuida a las grandes capacitancias manejadas fuera del circuito 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. Dado que el consumo dinámico es proporcional a la capacidad de las pistas, circuitos con gran interacción de entrada/salida (I/O – input/output) pueden consumir hasta ¼ o ½ de la potencia total, haciendo necesario estudiar la reducción en los sistemas multi-chips. Dado el crecimiento en el poder de las FPGA's y el concepto de SoC (System-on-a-Chip, sistema en un chip) hace que cada vez más cálculo se integre en un solo chip, haciendo menos voluminosa la comunicación con otros dispositivos electrónicos.

La tecnología de encapsulados esta 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 la tecnología MCM (Multi-Chips Module) donde se integran múltiples circuitos sobre un mismo sustrato, se reduce considerablemente la capacidad de interconexión respecto del montaje sobre tarjetas normales. Además, nuevas tecnologías de encapsulados como los CSP (Chip-Scale Packages) reducen considerablemente la capacidad de los patillas (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 mas lógica lo que eventualmente redunda en una menor actividad de entrada salida.

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). Estas 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 en este nivel.

2.4.2.1 Optimización de la Tensión Umbral (Threshold Voltaje-Vt)

Cuando se pretende reducir el voltaje 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. Aunque idealmente los dispositivos están desconectados para tensiones bajo V_t, en realidad siempre existe cierta conducción para $V_{gs} < V_t$. Este hecho es especialmente importante para circuitos dinámicos para los cuales las corrientes subumbral pueden causar cargas y descargas erróneas de los nodos dinámicos.

Para asegurarse la correcta funcionalidad de los circuitos, la tensión subumbral debería limitarse a un mínimo de 0,3 – 0,5 V.

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, como se mencionó anteriormente, 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 el voltaje y la actividad de los datos se mantienen la potencia se disminuirá en un factor de *S* también.

$$P \propto \frac{1}{S} \qquad \text{(Ecuación 2.11)}$$

Existen no obstante otros factores negativos en las interconexiones, la resistencia de estas es proporcional al largo e inversamente proporcional 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.

$$R_w \propto \frac{L}{W \cdot T} \propto S \qquad \text{(Ecuación 2.12)}$$

$$C_w \propto \frac{L \cdot W}{T} \propto \frac{1}{S} \qquad \text{(Ecuación 2.13)}$$

$$t_{wire} \propto R_w \cdot C_w \propto 1 \qquad \text{(Ecuación 2.14)}$$

La reducción de dimensiones físicas no puede realizarse indefinidamente, ya que después de cierto punto, consideraciones de segundo orden como la resistencia de las interconexiones comienzan a influir negativamente en los beneficios de disminución de potencia.

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 separadas por una capa más gruesa de dióxido de silicio.

Dado que la capacidad física decrece linealmente con el incremento de t_{ox} (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.

Esta ha de ser un parámetro en cuenta para los fabricantes de circuitos reprogramables, No obstante para el usuario estos parámetros no son accesibles.

2.4.2.4 Lógica dinámica (Dynamic logic)

Es este apartado se analiza un caso que excede el tradicional modelos CMOS estático. En la lógica estática y complementaria el voltaje de la salida es mantenido por uno de los rieles de alimentación (VDD o GND). En contraste en la lógica dinámica existen periodos durante los cuales no hay caminos entre los rieles de alimentación y el voltaje se mantiene gracias a capacitancias en los nodos.

El funcionamiento se puede entender en base a la figura 2.9. Durante el periodo de CLK = 0, el condensador de salida (C) se precarga a V_{DD} voltios. Cuando CLK = 1 el condensador de salida se mantiene a VDD voltios si $Y_0 = 0$ (es decir si f = 1), o bien se descarga hasta 0 voltios si $Y_0 = 1$ (es decir f = 0). Por tanto, cuando CLK = 1, la salida del circuito es igual a f (x_1,x_2,...x_n).

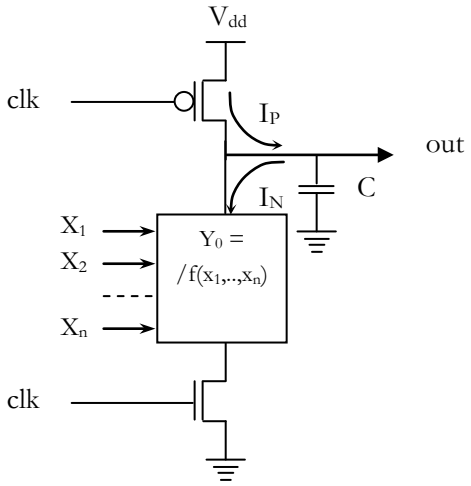


Figura 2.9. Esquema de la lógica dinámica

Históricamente, los diseños dinámicos fueron promocionados como inherentemente orientados al bajo consumo. Uno de los motivos es la menor cantidad de transistores necesarios, ya que todo el calculo es realizado por el arbol N-MOS en tanto el árbol P-MOS es reemplazado por un solo transistor de precarga (figura 2.10). Esta reducción en la cantidad de transistores tiene un impacto directo en la superficie ocupada por el circuito así como en la capacidad del circuito. Adicionalmente las compuertas dinámicas no disipan corriente de cortocircuito. Por otro lado esta garantizado que cada compuerta no realizará más de una transición por ciclo de reloj, a diferencia de la lógica estática que puede padecer de fenómenos de glitches.

No obstante en la práctica la lógica dinámica tiene varias desventajas. Para comenzar con ellas cada nodo de precarga debe ser manejado por una señal de reloj, esto implica una densa red de

distribución de reloj con su capacidad asociada. Adicionalmente con el crecimiento de la red de reloj los problemas de Skew comienzan a ser mas importante y difíciles de manejar.

En los circuitos actuales la señal de reloj es una señal de alta actividad (en la mayoría de los casos la de máxima actividad) y al estar conectada a la red de precarga (PMOS pull up network) puede introducir actividad innecesaria en el circuito. Para las compuertas tradicionales, la probabilidad que la salida realice un transición que produce consumo (de cero a uno) es siempre mayor en la lógica dinámica.

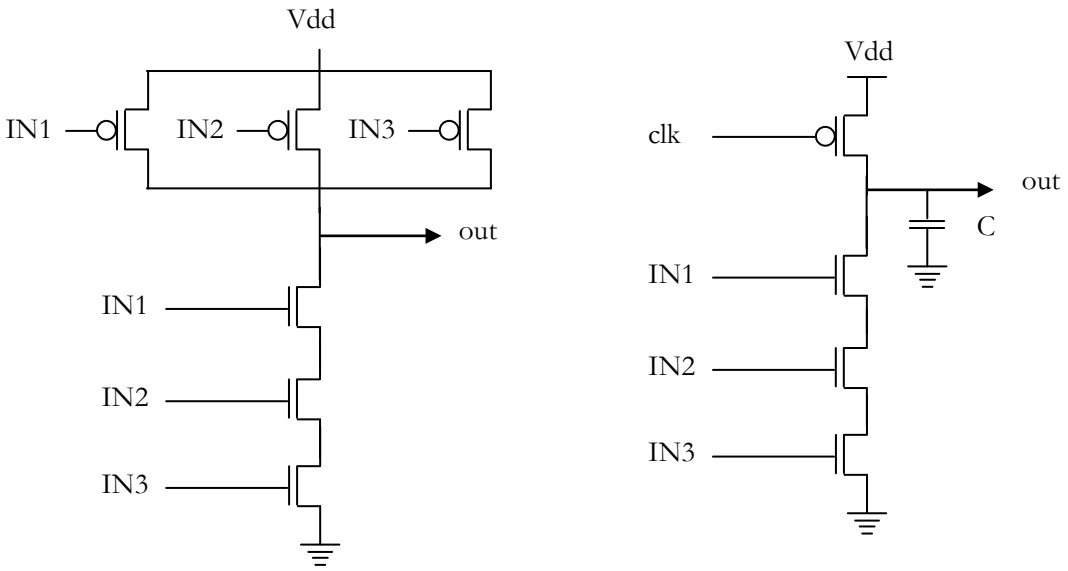


Figura 2.10. Compuerta NAND de 3 entradas en lógica estática y dinámica

En resumen lo lógica dinámica tiene ventajas y desventajas para la operación en bajo consumo. La clave es determinar que factor es el dominante para decidir su eventual utilización. Algunos casos exitosos son las PLAs dinámicas que logran funcionar a mayor velocidad y menor consumo, no obstante la mayoría de los circuitos se construyen en lógica estática.

2.4.2.5 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 aun 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.

2.5.2 Descomposición tecnológica y mapeo

Se denomina descomposición tecnológica al proceso de transformar una descripción booleana a nivel de compuertas 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 la funcionalidad de la descripción.

Por ejemplo en el caso de ASICs una compuerta NAND de tres entradas puede ser implementada por una simple compuerta 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.

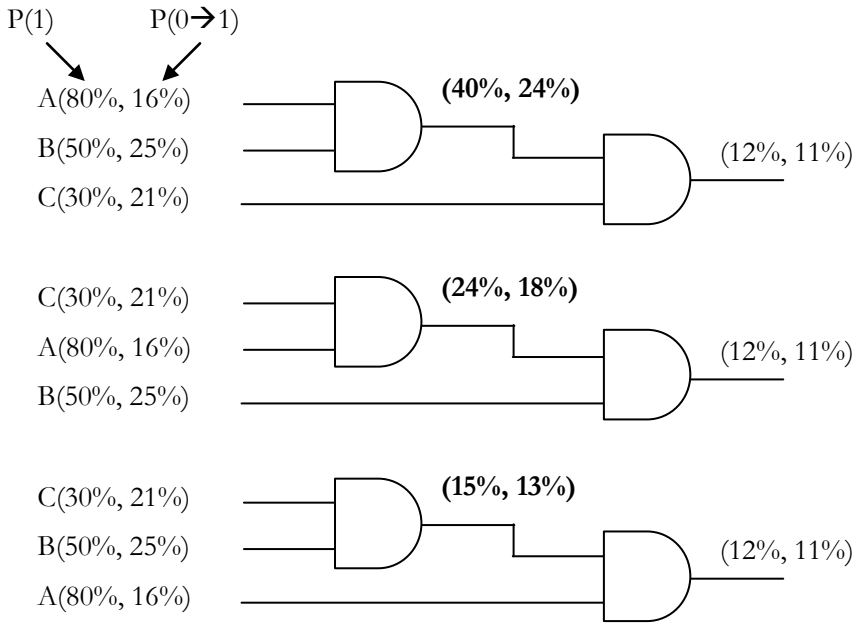


Figura 2.11. Dependencia del orden de entrada para la reducción de actividad

2.5.3 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 las cantidad de transiciones totales (figura 11 y 12). 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:

$$Tr = 2 \cdot p \cdot (1 - p) \qquad \text{(Ecuación 2.13)}$$

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).

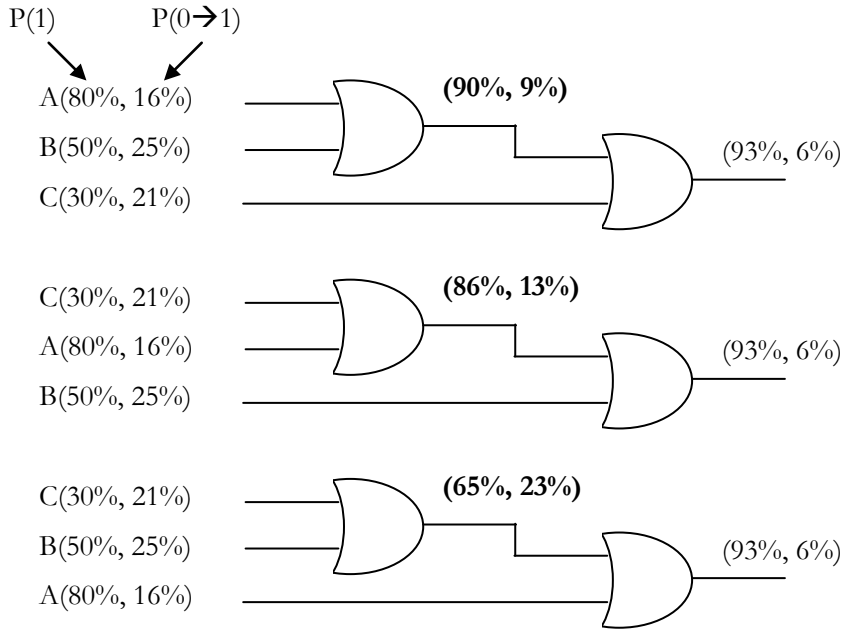


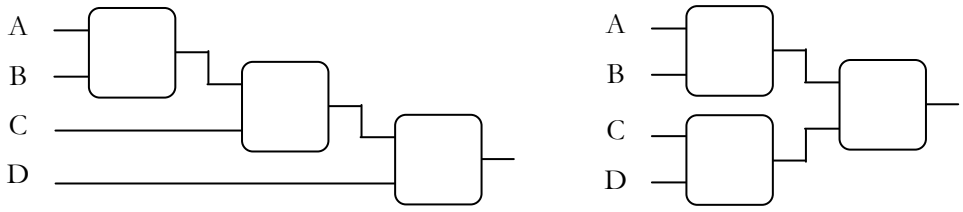
Figura 2.12. Dependencia del orden de entrada para la reducción de actividad

2.5.4 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 2.13 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, no desperdiciando consumo. 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.

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 2.13. 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.

2.5.5 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, el voltaje no puede ser modificado esta técnica pierde atractivo.

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 glitches. 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 el 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

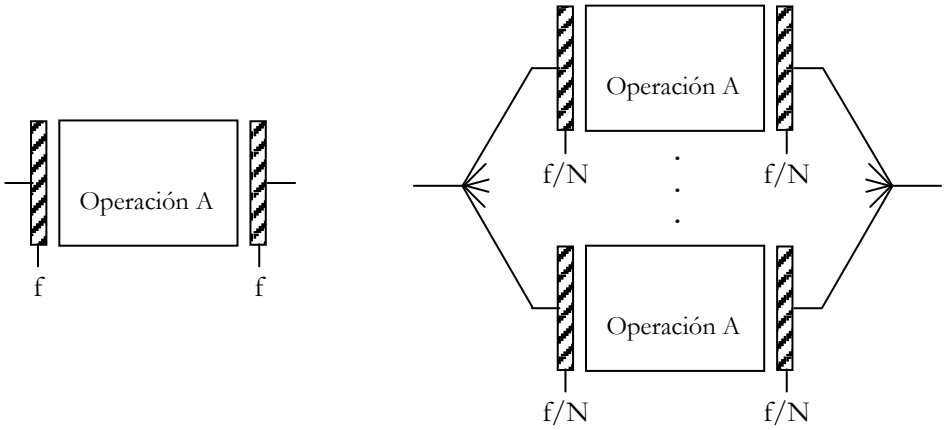
Se supone el ejemplo de la figura 2.14, donde un cálculo complejo A, se puede llevar a cabo en un cierto tiempo. Los registros de entrada y salida capturan a una frecuencia *f*. Si se supone que no existe dependencia de los datos de modo que se pueda paralelizar sin restricciones. Si se paraleliza el 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.

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 el voltaje 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:

$$P_n \propto C \cdot N \cdot \left(\frac{V}{N}\right)^2 \cdot \frac{f}{N} \propto \frac{P}{N^2} \qquad \text{(Ecuación 2.14)}$$

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 5, la velocidad de un circuito se puede considerar lineal a la tensión cuando $V_{dd} >> V_t$, más precisamente la expresión dice que $T \propto V_{dd} / (V_{dd}-V_t)^2$.



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.V²f**	**Power**	**~ P / N²**

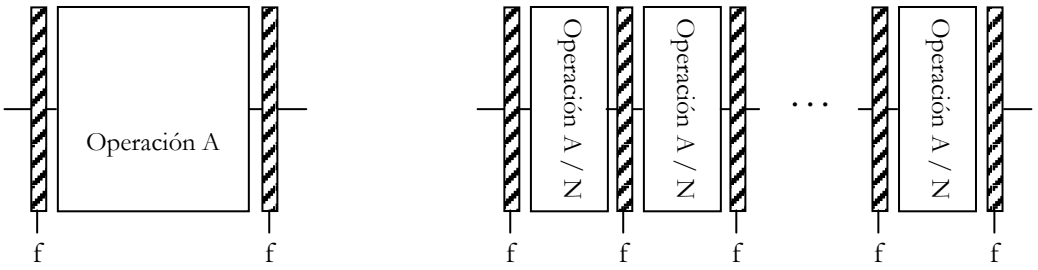
Figura 2.14. Reducción de voltaje y paralelismo para bajo consumo

2.6.1.2 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].



Circuito de Referencia

Capacidad = C
Frecuencia = *f*
Voltaje = V
Power = P ∝ C.V².f

Circuito con Pipeline

Capacidad ~ C
Frecuencia = *f*
Voltaje ~ V/N (despresiando Vt)
Power ~ P / N²

Figura 2.15. Reducción de voltaje y pipeline para bajo consumo

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 voltaje.

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 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].

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.

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 técnicas 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 essta 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 flotante-punto, 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 mas grandes de los requeridos por la aplicación. Estas decisiones apresuradas traen indiscutiblemente un aumento del área y el consumo.

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, signo-

magnitud, cero desplazado o la representación canónica con signo (canonical signed digit) son posibles representaciones aritméticas de datos. Sin dudas 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 (mas 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 15 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 2.16. 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 pasar por usar punto flotante. Otra alternativa es usar 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, podemos 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 (ver capitulo 3 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 2.15), se requiere conocer siempre el resultado anterior haciendo imposible (sin modificar el algoritmo) cualquier tipo de paralelización.

$$P_{i+1} := (P_i*2 + X(n-i)*Y) \qquad \text{(Ecuación 2.15)}$$

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 trasferencia 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 trasferencia entre módulos y el emplazado [Lid94][Geb97] [Lee95].

2.8.2 Particionado Hardware-Software

Es un concepto relacionado con el codiseño hardware software 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 optima [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 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 *Semiconductor Industry Association*.

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 voltaje y las dimensiones de las interconexiones, eventualmente la refrigeración de corto tiempo.

La energía consumida se expresa en julios (*joules*) e indica la potencia consumida en el tiempo, como lo muestra la ecuación 2.16. 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 2.17 muestra gráficamente esta ecuación.

Energy = power * delay (*joules* = *watts* * *seconds*) (Ecuación 2.16)

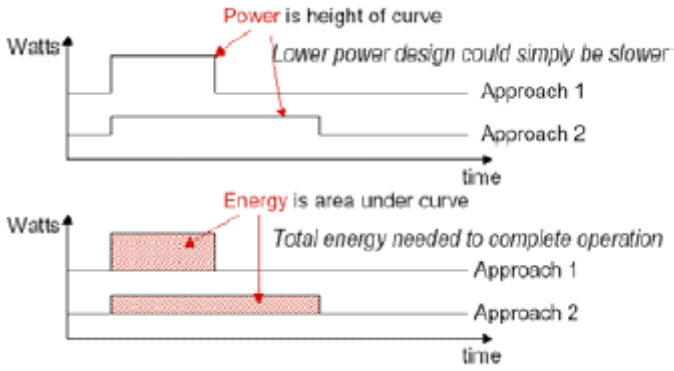


Figura 2.17. Gráfica del consumo y la energía

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 util de la batería (a menos que se cambie el voltaje de alimentación).

Métricas utilizadas

- 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].

  ETR = (Energy/operation)/ Throughput = Power/ Throughput^2 (Ecuación 2.17)

  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, en la figura 2.17. se pueden ver algunos valores caracterisiticos.

Processor Type	Characteristics	Energy Consumption
Conventional General Purpose processors (e.g. Pentiums)	is everything ... somehow we'll get the power in and back out	10-100 Watts, 100-1000 Mips **= 0 .01 Mips/mW**
Energy Optimized but General Purpose	Keep the generality, but reduce the energy as much as possible - e.g. StrongArm	5 Watts, 160 Mips **= 0.3 Mips/mW**
Energy Optimized and Dedicated		**= 100 Mops/mW**

Figura 2.18. Características típicas del consumo en microprocesadores.

2.9.2. Proyecciones para la tecnología de semiconductores

La asociación norteamericana de la industria de semiconductores [SIA03] realiza periódicamente proyecciones de la evolución de la industria (*Semiconductor Industry Association Roadmap*) de semiconductores. En base a sus informes y apoyado en los informes [Jew00][SIA97] se muestra la tabla con las proyecciones para los próximos años.

Capability	*1997*	*2001*	*2003*	*2006*	*2009*	*2012*
Line width (microns)						
DRAM lines	0.25	0.15	0.13	0.10	0.07	0.05
Logic lines	0.20	0.12	0.10	0.07	0.05	0.035
Memory						
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
Logic						
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
Memory						
Chip-to-package pads (cost performance)	800	1195	1460	1970	2655	3585
Chip-to-package pads (high performance)	1450	2400	3000	4000	5400	7300
Chip Frequency (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 performance	300	600	700	900	1200	1500
Chip-to-board, high performance	750	1400	1600	2000	2500	3000
Chip Size (mm^2**)**						
DRAM	280	445	560	790	1120	1580
Microprocessor	300	385	430	520	620	750

ASIC	280	445	560	790	1120	1580
Fabrication Process						
Number of on-chip wiring levels	6	7	7	7-8	8-9	9
Minimum number of masks	22	23	24	24-26	24-26	28
Maximum wafer diameter (mm)	200	300	300	300	450	450
Supply voltage (V) and power (watts)						
Logic supply voltage 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
Power (high perf. with heat sink)	70	110	130	160	170	175
Power (battery handheld	1.2	1.7	2.0	2.4	2.8	3.2

Figura 2.19. Proyecciones para la industria de semiconductores.

2.10. 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 el voltaje 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 el voltaje 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.

Capítulo 3:

Reducción de Consumo en Máquinas de Estado

Los circuitos secuenciales más importantes son sin duda las Máquinas de Estados Finitos (*Finite State Machine - FSM*). Si bien muchos problemas y algoritmos se pueden plantear en términos de máquinas de estados, y solo por ello tiene gran interés su estudio, es aún más importante el hecho de que prácticamente todos los circuitos digitales se implementan con una ruta de datos más una máquina de estados que la controla. En este capítulo se examina la reducción de consumo en máquinas de estados implementadas en FPGAs. El capítulo comienza con descripción y generalidades de las máquinas de estados (sección 3.1), luego un repaso de las estrategias utilizadas para la reducción de consumo en máquinas de estados (sección 3.2), más tarde se muestran resultados experimentales en la codificación de máquinas de estados (sección 3.3), . En la sección 3.4 se describe una técnica desarrollada para la partición de máquinas de estados, para finalizar en la sección 3.5 con recomendaciones para la reducción de consumo en máquinas de estado.

3.1. Definición de Máquina de Estados Finitos

El nombre proviene de que la secuencia lógica que implementa solo puede alcanzar una cantidad finita de estados. Las salidas y próximos estados de estas máquinas de estados finitos son combinaciones lógicas de las entradas y de su estado actual.

Más Formalmente, la definición estándar dice que una máquina de estado queda definida por una tupla $M = (\Sigma, A, Q, q_0, \delta, \lambda)$ donde:

Σ es el conjunto finito de símbolos de entrada.
A es el conjunto de símbolos de salida (debe ser $\neq \phi$).
Q es el conjunto finito de estados.
$q_0 \in Q$ es el estado inicial ("reset" state)
$\delta(q,a) : Q \times \Sigma \rightarrow Q$ es la función de transición de estados
$\lambda(q,a) : Q \times \Sigma \rightarrow A$ es la función de salida.

La especificación de una maquina de estados finita es equivalente a un grafo de transición de estados (State transition Graph - STG), donde cada estado corresponde con un nodo del grafo y cada arco entre dos estados q_i y q_j con una etiqueta *a / b* si existe $\delta(q_i,a) = q_j$ y $\lambda(q_i,a) = b$.

3.1.1 Procedimiento para el diseño de Máquinas de Estados Finitos

Se reconocen cinco pasos necesarios para la construcción de maquinas de estados:

Paso1: *Entender el problema:* Usualmente las especificaciones son ambiguas e incompletas y realizadas en texto o lenguaje informal. La primara tarea es entonces formalizar y entender la descripción.

Paso 2: *Obtener la representación abstracta de la FSM*: El formalismo más utilizado (aunque no el único) es el de diagrama de estados.

Paso 3: *Realizar minimización de estados*: Generalmente realizando la representación abstracta se logran FSM con demasiados estados y transiciones innecesarias pudiendo ser optimizado.

Paso 4: *Asignación de estados*: Los estados son derivados del contenido de Flip-Flops y una elección correcta simplifica la construcción. Otro criterio es minimizar las transiciones de los Flip-Flops con el objeto de reducir consumo.

Paso 5: *Implementar la maquina de estados*: Utilizando mapas de Karnaught o similares, minimizar funciones de transición y salida para luego describir el comportamiento en un lenguaje de descripción de hardware (HDL) u otro sistema de entrada en un EDA (State Editor [Ald99] StateCad [Sta02]).

3.1.2 Máquina de Mealy y Moore

Existen dos maneras básicas de organizar las redes secuenciales sincronizadas por reloj.

Máquinas de Moore: Las salidas solo dependen del estado actual (figura 3.1). Una red lógica combinacional genera el próximo estado a ser almacenado en los flip-flops en base a los datos de entrada y el estado actual. Las salidas se computan con un bloque combinacional solo función de las salidas de los flip-flops de entrada. Las salidas cambian sincrónicamente con la transición de estados y el flanco de reloj que lo produce. La mayor parte de las máquinas de estados son máquinas de Moore.

Máquinas de Mealy: Las salidas dependen del estado actual y del valor presente de las entradas (figura 3.2). Las salidas pueden cambiar inmediatamente tras el cambio de las entradas independientemente del reloj. Estas máquinas poseen salidas asíncronas.

Las salidas en las máquinas de Moore son síncronas con el reloj y cambian con los estados, en tanto las salidas en las máquinas de Mealy son asíncronas y pueden cambiar con cada cambio en los estados independientemente del reloj. Esto le da a las máquinas de Moore ventajas en términos de disciplina de sincronización, no obstante, existe una versión de Mealy con sincronismo en las salidas que posee interesantes características de bajo consumo.

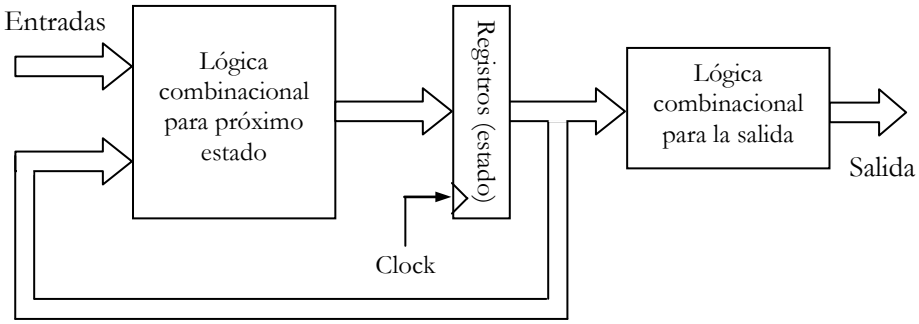


Figura 3.1. Máquina de estados de Moore

3.1.3 Diagrama de Estados para Máquinas de Mealy y Moore

Para el ejemplo del controlador para una ruta de datos que calcula el máximo común divisor se observan los diagramas de Mealy y Moore (figura 3.3). En el caso de Moore las salidas son asociadas a los estados, en tanto que para Mealy las salidas se asocian a los arcos de transición.

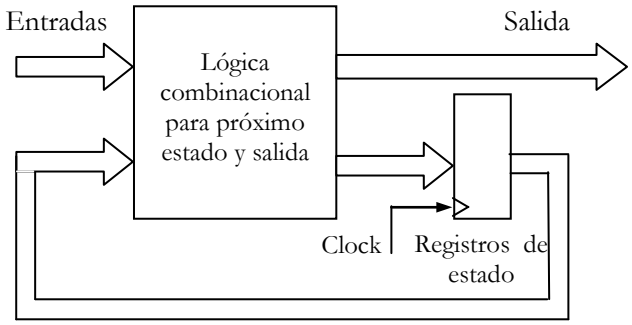


Figura 3.2. Máquina de estados de Mealy

3.1.4 Estados, Transiciones y Salidas en las Máquinas de Mealy y Moore

Se trata aquí de establecer los limites y complejidad que poseen las diferentes máquinas de estados. Para esto se supone una maquina de estados con M entradas, N salidas y L estados.

La primera cuestión es la cantidad de estados que se pueden codificar con *L* flip-flops, y esto es desde un mínima 1 a un máximo 2^L Estados.

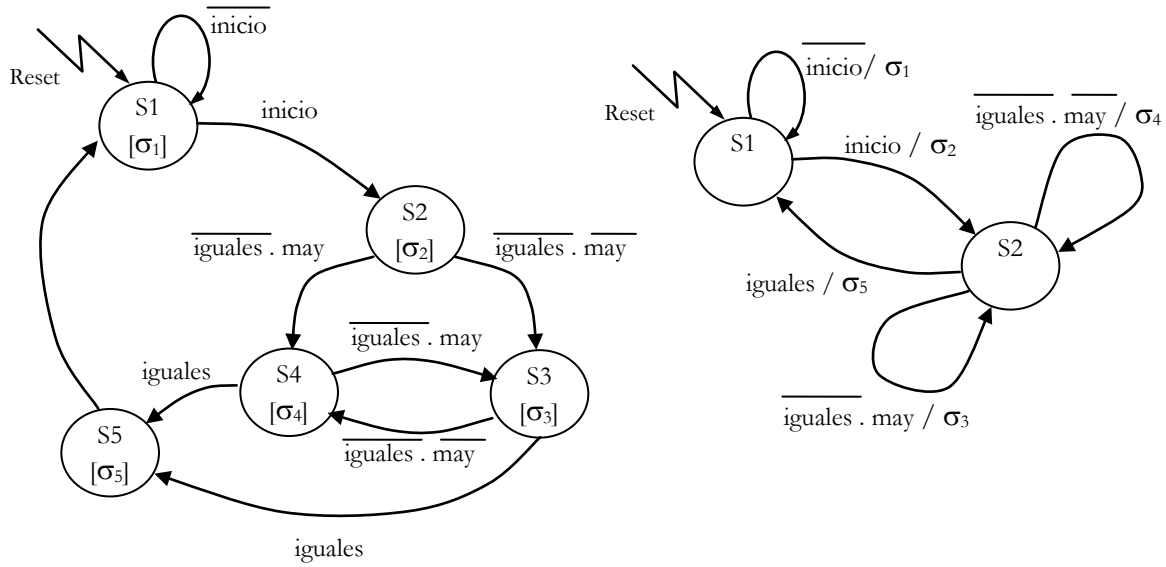
La cantidad de transiciones que pueden comenzar en un estado tiene que ver con la cantidad de posibles combinaciones de las entradas, es decir 2^M.

Una pregunta similar es la cantidad mínima y máxima de transiciones de estado que puede recibir un estado. Dado que existen estados de arranque que se alcanzan solo en la inicialización (a través de un "reset" por ejemplo) el mínimo es cero. Un estado puede recibir transiciones de cualquier otro estado incluyéndose el mismo y de cualquier combinación de las entradas, luego el máximo será $2^M * 2^L$ (el numero de todas las combinaciones entradas posibles multiplicado por todos los estados posibles).

Otra pregunta es la mínima y máxima cantidad de patrones que pueden ser observados a la salida de la máquina. El mínimo número de patrones de salida es naturalmente uno, cada estado y cada transición puede ser asociado con el mismo patrón.

El máximo número de patrones depende del tipo de máquina. En las máquinas de Mealy, dado que las salidas están asociadas a los arcos de transición, este valor será el mínimo entre la cantidad de transiciones posibles y la cantidad de patrones posibles Min($2^M * 2^L$, 2^N). Si el número de transiciones supera al número de posibles patrones de salida, algunos han de ser repetidos.

Para el caso de las máquinas de Moore, el máximo será lo menor entre el número de estados y la cantidad de patrones posibles Min(2^L , 2^N). Aquí nuevamente si el número de estados excede el número de patrones de salida, algunos de estos deberán ser repetidos.



a) Maquina de Moore b) Maquina de Mealy

Figura 3.3. Diagrama de estados para el controlador de la ruta de datos del MCD.

3.1.5 Comparación de las Maquinas de Mealy y Moore.

Dado que, se puede asociar salidas a las transiciones de estados en vez de solo a los estados, las máquinas de Mealy pueden ser más pequeñas que las de Moore. Como se mencionó anteriormente la cantidad de patrones de salida en las máquinas de Mealy pueden llegar a ser Min($2^M * 2^L$, 2^N), en tanto que en Moore Min(2^L , 2^N) de este modo potencialmente se pueden representar máquinas con menos estados en el modelo Mealy que en Moore.

Sea el siguiente ejemplo: Se desea controlar una ruta de datos que implementa el MCD (Máximo Común Divisor) con una máquina de estados. La maquina debe generar 5 ordenes distintas (σ_1, σ_2, σ_3, σ_4 y σ_5) en base a 3 señales de entrada (inicio, igual y mayor). Los detalles de la semántica de la máquina no se analizan aquí. Se muestran las diagramas de estados de ambas máquinas (Mealy y Moore). Obsérvese que la mínima cantidad de estados de la máquina de Moore es de 5 y están determinados por la cantidad de salidas que necesita la máquina, en tanto que la máquina de Mealy solo necesita 2 estados. La implementación de la primera requiere como mínimo 3 flip-flops($\lceil \text{Log}_2 5 \rceil = 3$) en tanto que la segunda solo necesita 1 ($\lceil \text{Log}_2 2 \rceil = 1$).

3.1.5. Máquinas de Mealy Síncronas

Los glitches que se pueden producir en las salidas de las máquinas de Mealy son inherentes a u construcción asíncrona. Los glitches no solo producen un gasto de consumo innecesario sino que además pueden producir comportamientos erróneos. Pero dado que las máquinas de Mealy pueden llevarse a cabo con menos estados, ahorrando de este modo registros, resulta atractivo usarlas.

Las máquinas de Mealy síncronas simplemente evitan la conexión directa de las entradas con las salidas interponiendo registros a la salida. La figura 3.4 muestra una forma de llevarlo a cabo, agregando flip-flops a las salidas. Los flip-flops son sincronizados con el mismo flanco que los registros de estados. Esto tiene el efecto de convertir una máquina de Mealy en una máquina de Moore, haciendo las salidas parte de la codificación.

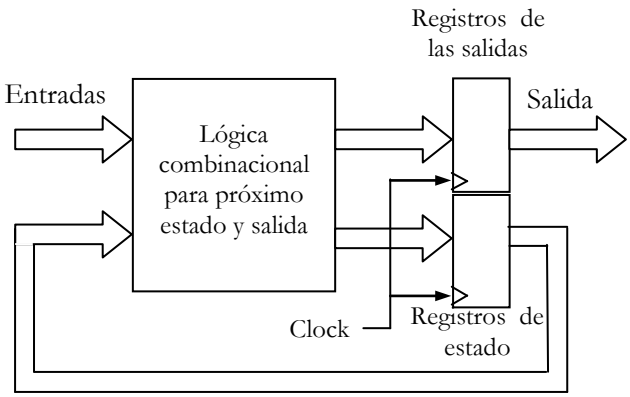


Figura 3.4. Maquina de estados de Mealy Síncrona

3.2 Estrategias para reducción de consumo en Máquinas de Estados

Las principales técnicas para la reducción de consumo en máquinas de estados se refieren a la minimización o reducción de estados, codificación de los estados y a la descomposición en varias submáquinas estados.

3.2.1 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 de superficie, 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].

3.2.2 Asignación de estados (state assignament/encoding)

Es una de las técnicas más extendidas para la reducción de consumo en máquinas de estados finitos. La principal idea es reducir la cantidad de transiciones entre estados. Se realiza a continuación una breve revisión del tema.

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 optima 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 sistema de síntesis de circuitos secuenciales SIS [Sen92] desarrollo por la Universidad de California, Berkley.

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.

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], a partir de esto varias estrategias han sido propuestas. 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 en que modelan el grafo de transición de estados (STG) como una cadena de Markov y resuelven el problema de asignación con minima 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].

Asignación de estados en FPGAs

Las investigaciones antes mencionadas fueron pensadas para circuitos integrados tipo *gate arrays* o *estandar cells*. Por otra parte, los fabricantes de FPGAs y de herramientas de síntesis utilizan One Hot (tantos registros como estados, con un único bit a '1' y los demás a cero) como la codificación por defecto para máquinas de estado [Xil00a][Exp99]. Este sistema de codificación permite crear FSMs más eficientes en FPGA en términos de área y profundidad lógica (por tanto velocidad).

Las FPGAs poseen muchos registros pero las generación de funcione 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 basadas en codificaciones binarias pueden tener mejores resultados.

3.2.3 Descomposición de maquinas de estado (FSM partitioning o descomposition)

Se trata de dividir máquinas de estados en otras más pequeñas comunicadas entre si. Maquinas más pequeñas han de ser más fáciles de asignar estados por separado que todas juntas, además si la división es correctamente realizada la mayor parte del tiempo solo una de las máquinas de estados pequeñas 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 3.4 se estudia en profundidad la aplicación de esta técnica en FPGAs.

3.3 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, luego 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] (más detalles en el Apéndice C).

```
# 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
```

```
entity EX5 is
  port(clk, rst: in std_logic;
    i : in std_logic_vector (1 downto 0);
    o : out std_logic_vector(1 downto 0));
end EX5;

architecture behave of EX5 is
type state is ( S0, S1, S2, S3);
attribute enum_encoding of state: type is
       "1000 " & -- S0
       "0100 " & -- S1
       "0010 " & -- S2
       "0001 " ; -- S3

signal machine, next_state : state;
signal o_i : std_logic_vector(1 downto 0);
begin
st_machine: process (rst, clk)
begin
  if rst = '1' then
     machine <= S0;
  elsif rising_edge(clk) then
     machine <= next_state;
  end if;
end process st_machine;
```

```
next_st_prc: process(machine,i)
begin
case machine is
   when S0 =>
      if i = "00" then
         next_state <= S0 ;
         o_i <= "11";
      elsif i(0) = '1' then
         next_state <= S2 ;
         o_i <= "00";
      elsif . . .
      . . .
      end if;
   when S1 =>
      . . .
   when others =>
      . . .
end case;
end process next_st_proc;

o <= o_i;

end behave;
```

Figura. 3.5. Código KISS y resumen del código generado.

3.3.1. 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 codificaciones que se consideran densamente codificadas (por el uso que realizan de las posibles codificaciones sobre los registros usados) en tanto que One-Hot y Two-Hot son las consideradas codificaciones dispersas o poco densas.

circuits	Original Machine				Minimized Mach.			
	inp	outp	rul	#st	inp	outp	rul	#st
bbara	4	2	60	10	4	2	42	7
bbsse	7	7	56	16	7	7	208	13
bbtas	2	2	24	6	2	2	24	6
beecount	3	4	28	7	3	4	20	4
cse	7	7	91	16	7	7	91	16
dk14	3	5	56	7	3	5	56	7
dk15	3	5	32	4	3	5	32	4
dk16	2	3	108	27	2	3	108	27
dk17	2	3	32	8	2	3	32	8
dk27	1	2	14	7	1	2	14	7
dk512	1	3	30	15	1	3	30	15
donfile	2	1	96	24	2	1	4	1
ex1	9	19	138	20	9	19	233	18
ex2	2	2	72	19	2	2	56	14
ex3	2	2	36	10	2	2	20	5
ex4	6	9	21	14	6	9	21	14
ex5	2	2	32	9	2	2	16	4
ex6	5	8	34	8	5	8	34	8
ex7	2	2	36	10	2	2	16	4
keyb	7	2	170	19	7	2	170	19
kirkman	12	6	370	16	12	6	370	16
lion9	2	1	25	9	2	1	16	4
mark1	5	16	22	15	5	16	180	12
opus	5	6	22	10	5	6	29	9
planet	7	19	115	48	7	19	115	48
prep3	8	8	29	8	8	8	29	8
prep4	8	8	78	16	8	8	78	16

Table 3.2. Circuitos de prueba originales y minimizados en cantidad de estados.

Los experimentos usan 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 tiene entre cuatro y cuarenta y ocho estados, de uno a doce entradas y de una a diecinueve salidas. Las maquinas están descriptas en el formato KISS2 [Sen92] (figura 3.9).

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 3.5) y un top-lovel VHDL con buffers de tercer estados en las patas para poder separar la potencia de fuera del chip. La descripción de la herramienta KISS2VHDL, así como del formato de descripción KISS2 y las FSMs de los bancos de pruebas se pueden encontrar en el apéndice C.

Las máquina de estado de prueba fueron primeramente minimizadas (en cuanto a la cantidad de estados y transiciones) con STAMINA [Hac91] herramienta distribuida con el sistema de síntesis de circuitos secuenciales SIS [Sen92]. La cantidad de entradas, salidas, reglas de próximo estado y **cantidad de estados de las maquinas originales y las minimizadas se pueden ver en la Tabla 3.2.**

Circuits	FSM characterist.				Area Bin		Area OH		Area Out-O		Area T-H		Delay (ns)				Power mW/MHz			
	inputs	outpts	rules	states	CLBs	FF	CLBs	FF	CLBs	FF	CLBs	FF	Bin	OH	Out-O	T-H	Bin	OH	Out-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 3.3. Area-Time -Power para el conjunto de circuitos de prueba.

Luego se traducen las FSMs con la herramienta KISS2VHDL con las respectivas codificaciones. Se utilizo FPGA Express [Syn99] para la síntesis y las herramientas Xilinx foundation [Xil00b] para la implementación en el dispositivo destino, una XC4010XC84-4.

Todos los circuitos fueron implementados y medidos en idénticas condiciones, esto es, todas 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), secuencia aleatoria de vectores de entrada, frecuencia de reloj y sondas del analizador lógico. Se utilizo la misma secuencia aleatoria para estimular los circuitos, la salida de cada pata soporta la carga del analizador lógico digital, la que es inferior a 3 pf [Tek02].

Los circuitos fueron medidos a 100 Hz, 2MHz, and 4 MHz para extrapolar la potencia estática, todos los prototipos incluyes buffer triestadoss en las patas de salida para facilitar la medida de la potencia fuera del circuito. (Ver metodología de medidas Apéndice A).

3.3.2. Resultados Experimentales

En la tabla 3.3 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 critico. Finalmente la potencia dinámica es expresada en mW/MHz.

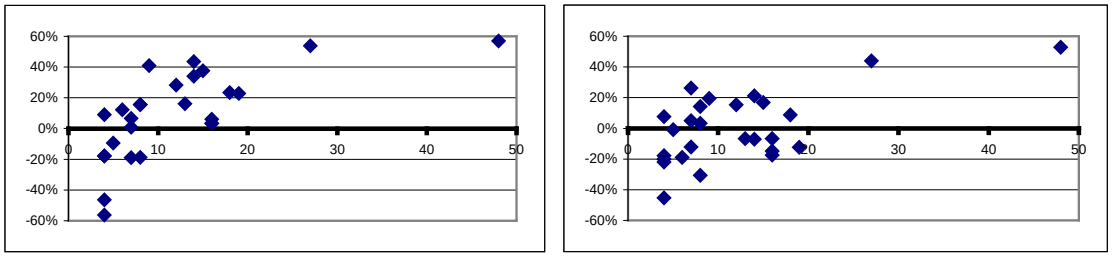


Figura 3.6. Ahorro de consumo en función de la cantidad de estados. a) One-Hot respecto de Binario. b) One-Hot respecto de "Out-oriented"

Ahorro de consumo: La Figura 3.6 muestra la comparación de ahorro de consumo: (a) OH (One Hot) vs. codificación binaria y en (b) OH vs. "Out-oriented". Los valores positivos indican reducciones obtenidas por la codificación OH. El eje de las abscisas (de las x) representa el número de estados de la FSMs. Las figura puede ser (conceptualmente) dividida 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.

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 puede llegar al 57% por la elección de la codificación correcta.

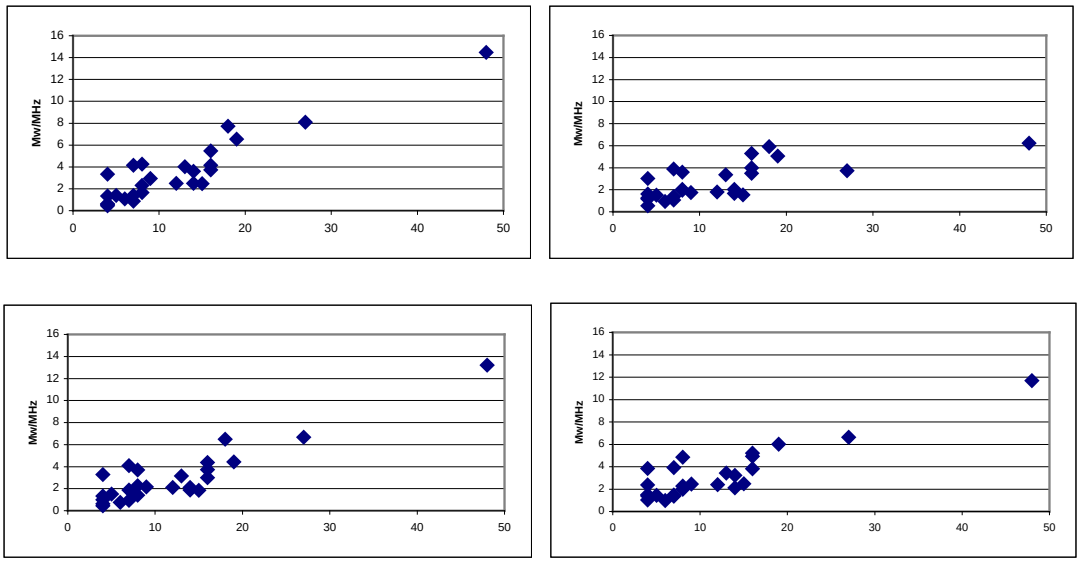


Figura 3.7. Consumo por nro de estados: a) Binary; b) One-Hot; c) Out-oriented; d) Two-Hot.

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 3.7). El consumo es aun 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 $r2 \cong 0.80$

Velocidad-Consumo: La relación se puede ver gráficamente en la figura 3.8, el coeficiente de correlación es $r2 \cong 0.7$. El experimento no sigue la regla general dentro del diseño de bajo consumo en FPGAs, que indica que, los circuitos más rápidos consumen menos potencia.

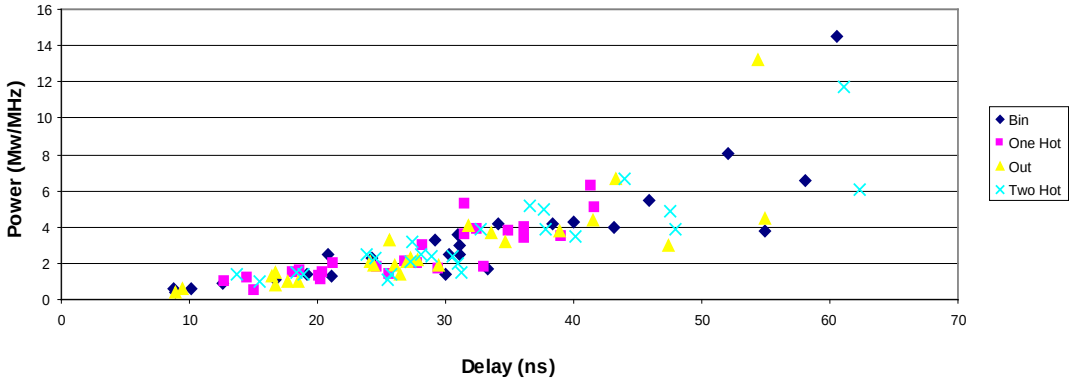


Figura 3.8. Relación retardo- consumo para las FSMs

Área-Consumo: La correlación es realmente importante ($r2 \cong 0.91$) y puede ser usada como una primera aproximación para decidir por un sistema de codificación. La figura 3.8 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.

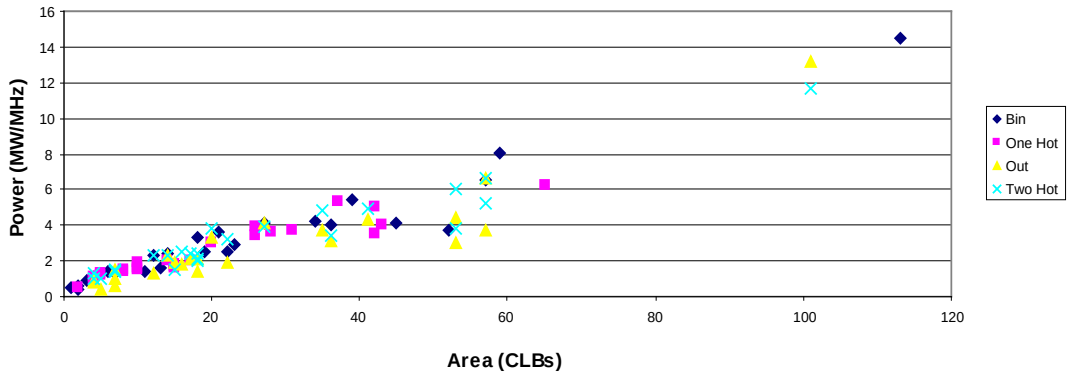


Figura 3.9. Relación área – consumo en las máquinas de estados

Otras correlaciones, como cantidad de estados-velocidad no son visibles (*r2* menor que 0,6). Correlaciones de área velocidad o consumo respecto a otros paramentos de la máquina de estados (cantidad de entradas, salidas, estados, reglas) y combinaciones de estos parámetros tampoco producen resultados significativos.

3.3.3. 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 importante 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 fácil de decodificar como One-Hot o Two-Hot muestran mejores resultados para máquinas de estados grandes (mas 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, una clara relación área-consumo se puede observar. Esta puede ser utilizada durante el ciclo de diseño para estimar el consumo a través de la información provista por la herramienta de síntesis.

3.4 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 optimas, se pueden lograr mejoras en área, velocidad y consumo en circuitos secuenciales realizando la interconexión de dos o más circuitos. Heurísticas para la asignación de estados y optimización lógica trabajan mejor sobre problemas pequeños que sobre grandes. Es mejor por tanto dividir grandes máquinas de estados. Desde el punto de vista del consumo, si la división es correctamente realizada, la mayor parte del tiempo solo una de las submáquinas de estados ha de estar funcionado pudiéndose desactivar las demás.

En este trabajo, la máquina de estados e dividida en dos submáquinas usando un criterio probabilístico. Solo una submáquina está activa a la vez mientras la otra esta inactiva para ahorrar consumo. Se han probado diferentes alternativas de desactivación en dispositivos Xilinx XC4K. Las alternativas fueron medidas en consumo de potencia usando circuitos de los bancos de prueba (benchmarks) MCNC [Lis88] y PREP [Pre00]. Con la técnica propuesta, que se adapta para grandes máquina de estados, se han logrado ahorros de consumo de hasta el 46 %.

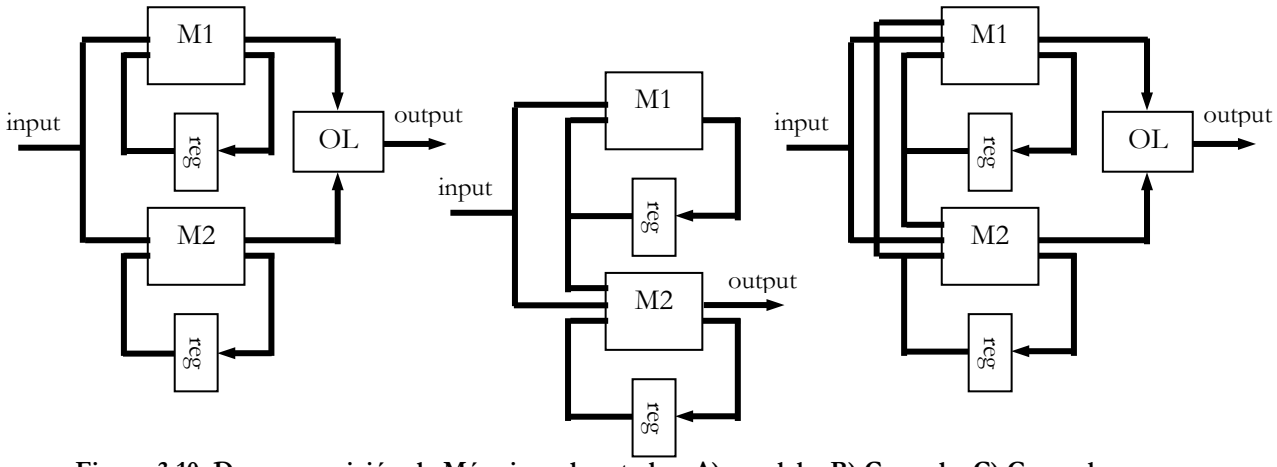


Figura 3.10. Descomposición de Máquinas de estados. A) paralela. B) Cascada. C) General

3.4.1 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 en la sección 3.3. 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 ncircuito 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, 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 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.

3.4.2 Decomposición o partcionado de FSMs

Muchos trabajos de investigación se han realizado en el tema de partición o decomposción de máquinas de estado. Trabajos sobre partición de máquinas de estados datan de los 60 [Har60]. El primer objetivo fue reducir la complejidad del bloque combinacional que se mapeaba en arquitecturas que poseían una cantidad limitada de lógica (PLAs, etc) [Ash91][Gei91]. 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 3.10), siendo estas últimas las más utilizadas en la práctica.

Modelo General de 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 3.11). Esta estrategia agrega un estado ocioso en cada submáquina.

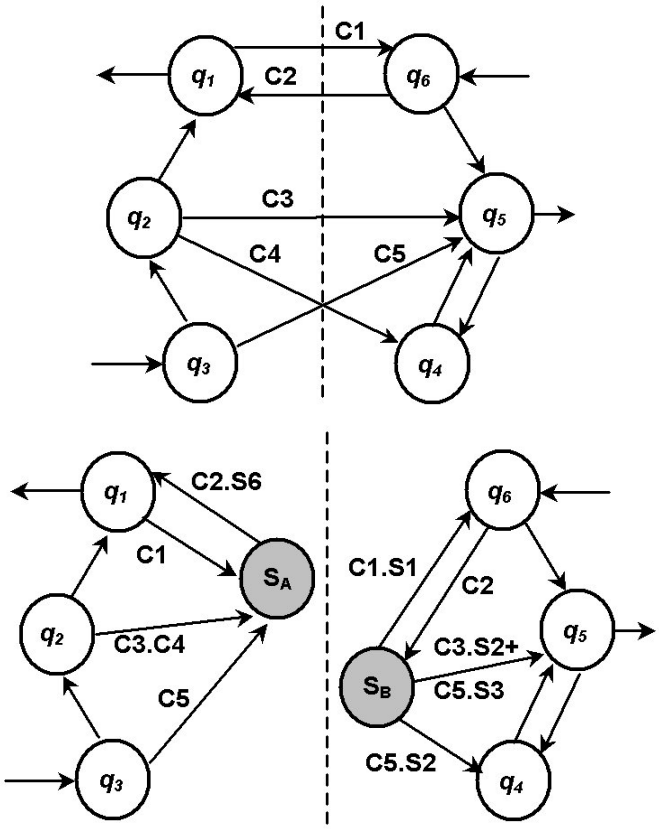


Figura 3.11. Diagrama de estados de una máquina de estados y diagramas de las particiones en la aproximación tradicional.

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 particiones no es $n_1+n_2 = n$, pero es aproximadamente $\sqrt{n}$ en cada partición. Si se considera Q = {q_1,q_2,…,q_n} el conjunto original de estados, luego dos particiones $\prod_A$ = {A_1, A_2, …, A_m} y $\prod_B$ = {B_1, B_2, …, B_k}del conjunto Q son *ortogonales* si, para todo i , j con $i \le m$, $j \le k$, se cumple que $A_i \cap B_j = \varnothing$ o bien $A_i \cap B_j = \{q_i\}$. De este modo, para representar un estado q_t de la máquina original se utiliza la combinación de un estado A_i y otro B_j

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 suceded 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).

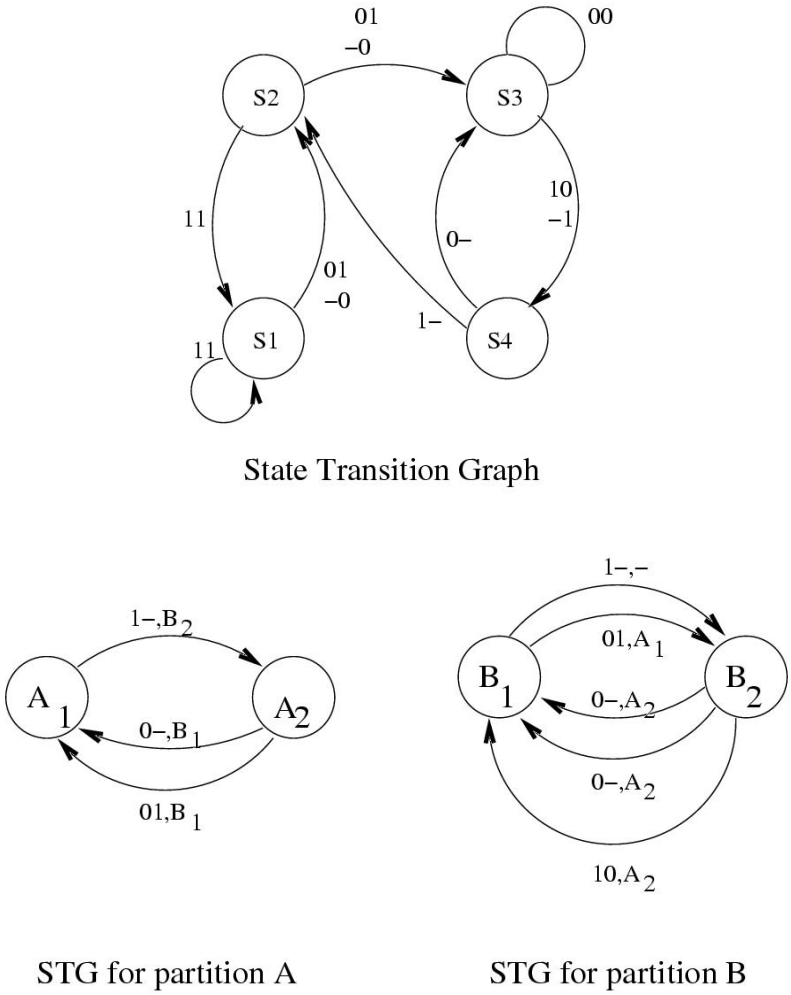


Figura 3.12 Diagrama de estados y partición ortogonal de una FSM

En la figura 3.12 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 tenemos: 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.

3.4.3 Técnicas de descomposición para bajo consumo

La idea principal para reducir consumo por descomposición de máquinas de estados es desactivar la parte inactiva de la FSM. La desactivación puede ser alcanzada o bien bloqueando las entradas, usando latches, puertas ANDs o buffer tri-estados o bien apagando la parte del circuito que no es usada (apagando 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 este 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.

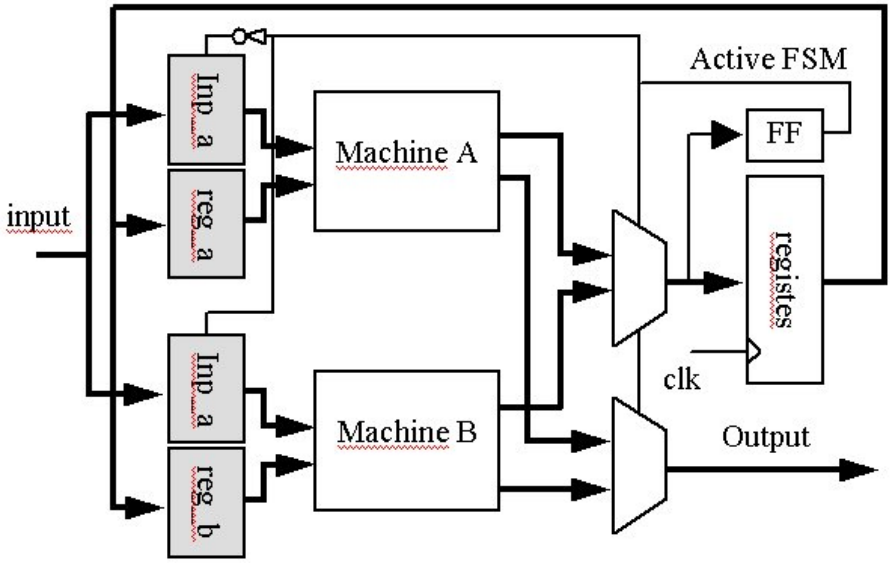


Figura 3.13. Arquitectura I para descomposición de máquinas de estado en FPGAs.

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 reducida.

Otra técnica utilizada para reducir consumo es utilizar la partición ortogonal descripta 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.

3.4.4 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 3.13. 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.

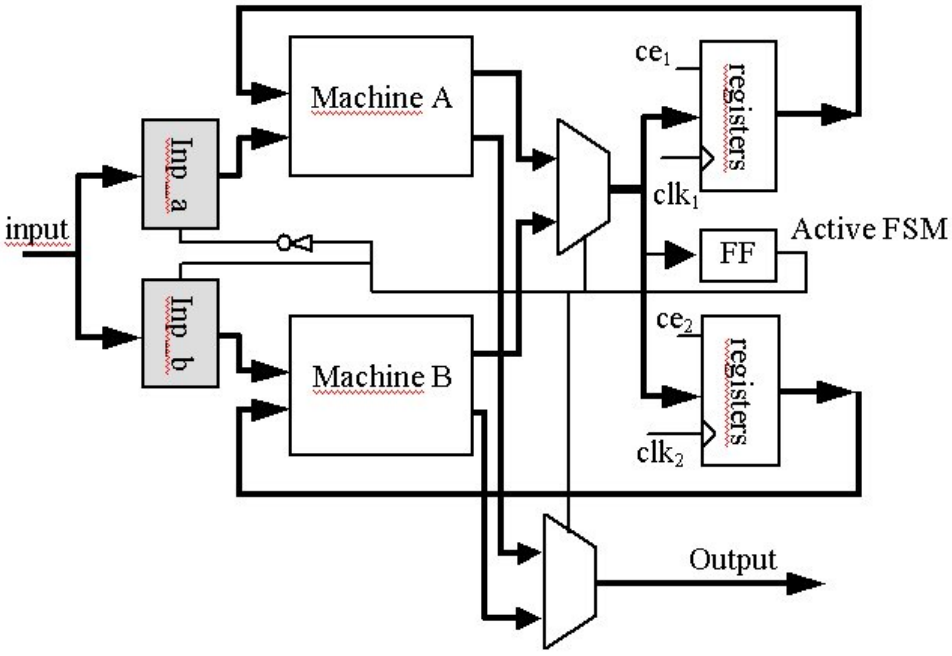


Figura 3.14. Arquitectura II para descomposición de máquinas de estado en FPGAs.

Una segunda arquitectura se puede ver el la figura 3.14. 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) o utilizando la técnica de *gated-clock* (aunque desaconsejado para el caso de FPGAs por los fabricantes).

Para ambas arquitecturas, el mismo ciclo de diseño debe ser llevado a cabo. Primero, un algoritmo para descomponer la FSM en dos submáquinas debe ser seleccionado. Luego, un método para bloquear los datos en la submáquina deshabilitado debe ser implementado. Finalmente, el código VHDL sintetizable debe ser escrito.

3.4.5. Cálculo de probabilidades de transición en un STG

Para llevar a cabo la partición de las máquinas de estado se lleva a cabo un modelo probabilistco [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:

$$v \cdot P = v \quad \text{y} \quad \sum_{i=0}^{n-1} P_i = 1; \quad \text{donde} \quad v = [P_0 P_1 ... P_{n-1}] \quad \text{y} \quad P = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ p_{n-1,0} & p_{n-1,1} & \cdots & p_{n-1,n-1} \end{bmatrix}$$

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 $\mathbf{P_{i,j}}$ puede ser calculada como $P_{i,j} = p_{i,j} \cdot P_i$.

Un ejemplo sencillo: La figura 3.15 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 3.15.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.

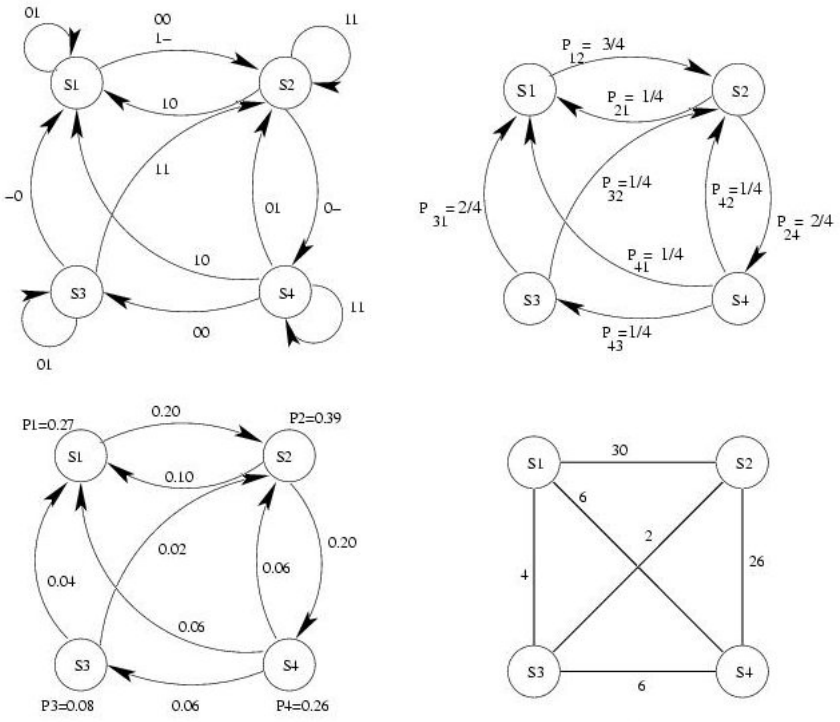


Figura 3.15 Diagrama de transición de estados y grafos de probabilidades.

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 3.15.c. Finalmente en la figura 3.15.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). Detalles de la implementación software en el apéndice D.

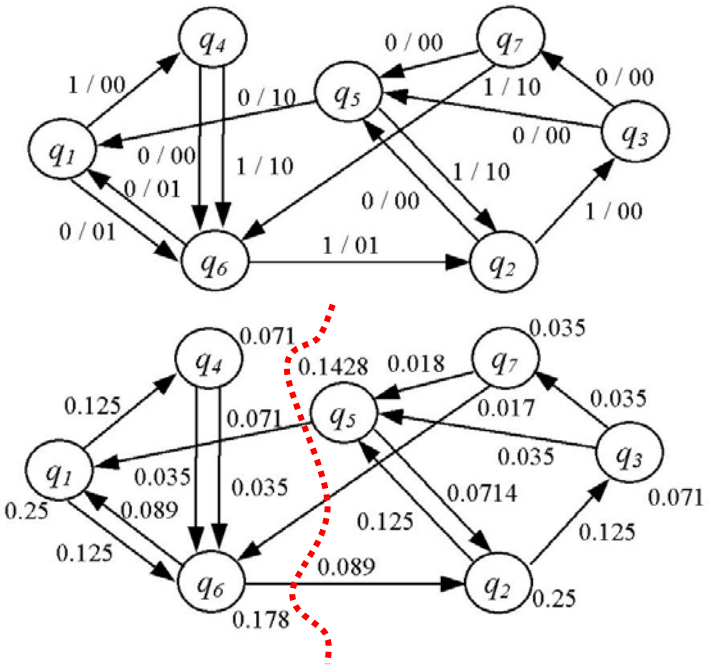


Figura 3.16. a) Diagrama de transición de estados (State Transition Graph - STG), b) Probabilidad estática y probabilidad total de transición

3.4.6. 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 3.16). 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 ejemplos en pocos segundos. Más detalles en el apéndice D.

3.4.7 Métodos para bloquear los datos

Para eliminar la actividad en la máquina de estado inactiva (bloques sombreados en figuras 3.13 y 3.14), 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. Ambos fueron testeados y descartados por su peor comportamiento en área, velocidad y consumo.

Para la arquitectura de partición de FSMs como la de la figura 3.14, 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 existir problemas por el *clock skewing*) y el no menos importante hecho que los recursos de reloj no sobran en las FPGAs y puede ser necesario usar líneas comunes para rutar el reloj con una importante perdida de calidad en el rutado (peor tiempo, líneas más largas y consecuentemente más consumo).

3.4.8 Síntesis de la máquina de estados

Se desarrollo 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 3.3.7 para finalmente generar código VHDL sintetizable. Otros parámetros son necesarios para el programa, tales como: tipo de arquitectura, método de bloqueo (blocking method), tipo de codificación de las submáquinas.

El código VHDL generado contiene la entidad de la máquina de estados y tres procesos (processes): uno para la lógica combinacional, otro para la circuitería de bloqueo de los datos y la última incorpora buffers tri-estado en las patas de salida para medir por separado la potencia fuera del chip (*off-chip power*).

3.4.9 Experimentos con partición de máquinas de estado

Los circuitos de prueba fueron implementados de diferentes maneras: 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 3.13) y otra a la *arquitectura II* (figura 3.14). Nuevamente para cada submáquina se aplico codificación binaria y One Hot. Adicionalmente sorber la arquitectura I se probaron diferentes técnicas de bloqueos de datos.

Por otra parte, se procedió a implementar la partición ortogonal (sección 3.4.2) en las máquinas de estado. Los resultados fueron muy deficientes en performance, las conclusiones y datos se muestran por separado.

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 3.1. 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

El código VHDL generado por el programa *part_FSM* (apéndice D) es luego compilado 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 ellos fueron construidos con look up tables (LUTs) actuando como memoria ram.

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 3.1. 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)

Todos los circuitos fueron implementados y medidos en idénticas condiciones, esto es, todas 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), secuencia aleatoria de vectores de entrada, frecuencia de reloj y sondas del analizador lógico. Se utilizo la misma secuencia aleatoria para estimular los circuitos, la salida de cada pata soporta la carga del analizador lógico digital, la que es inferior a 3 pf [Tek02].

Los circuitos fueron medidos a 100 Hz, 2MHz, and 4 MHz para extrapolar la potencia estática, todos los prototipos incluyes buffer tri-estados en las patas de salida para facilitar la medida de la potencia fuera del circuito [Tod00]. Ver detalles de la metodología en apéndice A.

3.4.10 Resultados experimentales de la partición de máquinas de estado

El consumo de corriente expresado en mW/MHz, se muestra en la tabla 3.5. Las primeras columnas muestran los valores para la máquina de estados original codificada en One Hot (OH) y binario (bin). Luego los resultados para el circuito particionado y codificado en tanto en One Hot como en binario se muestran de 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 compuertas ANDs (*Blk and*). En la última columna se observa un 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.

En la tabla 3.3 se muestra el área de los circuitos de la tabla 3.2 expresada en CLBs de la serie XC4000, además se muestra la utilización de registros flip-flop.

	Original FSM		Partitioned One Hot Encoded				Partitioned Binary Encoded				Power Improvement
Sample	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 3.2. Consumo de potencia expresado en mW/MHz para la partición de máquinas de estado.

Mejora de consumo: Para la mayor parte de las máquinas de estado se obtiene una reducción considerable de consumo llegando hasta un 42 % en las máquinas analizadas. 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 3.4).

Codificación Binario vs. One Hot en las submáquinas: De acuerdo con los resultados relacionados con el consumo en máquinas de estados sin particionar (sección 3.3), 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 mayoria de los casos. La mejora respecto de utilizar puertas AND (en realidad se "gasta" un CLB para esto) puede llegar a un 30 %. Solo en dos circuitos del banco de pruebas, el bloqueo con compuertas AND fue mejor y esto ocurrió debido a la escasa actividad de estos circuitos.

Sample	Original FSM				Partitioned One Hot Encoded								Partitioned Binary Encoded							
	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 3.3. 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 (la XC4000E no posee 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, en términos de área la *arquitectura I* posee peores características que la *arquitectura II* (lo contrario que en el consumo).

Sample	$\|\Sigma\|$	$\|\sigma\|$	$\|Q\|$	$\|\delta\|$	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 3.4. 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 3.17 muestra la máxima frecuencia en MHz para cada circuito implementado con One Hot. La influencia de los latches es remarcable, en tanto que las arquitectura que utiliza el bloqueo de los datos con compuertas ANDs muestra mejores performances.

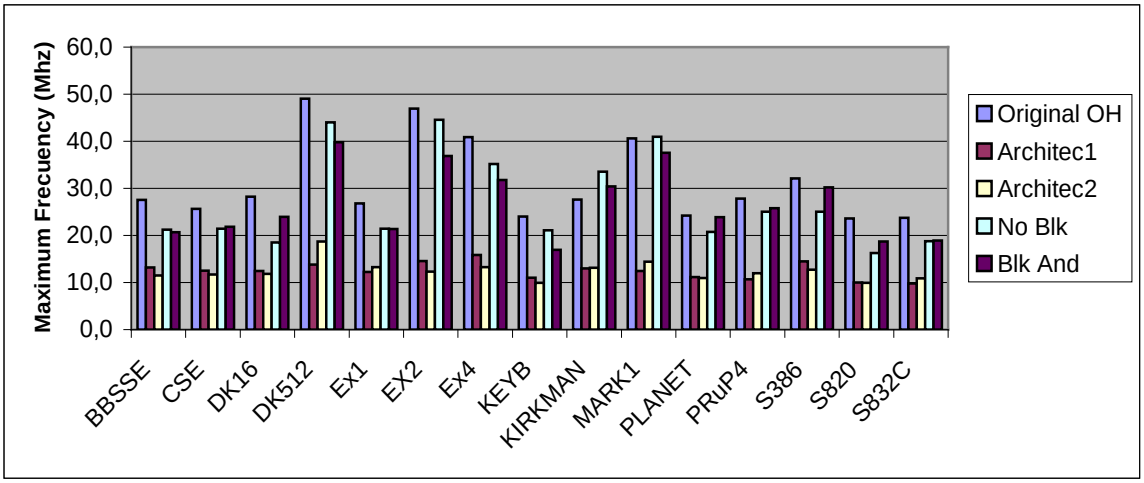


Figura 3.17. Gráfica de frecuencia, donde se observa el impacto negativo de los registros de bloqueo.

Particionado ortogonal: Se implemento el esquema de partición ortogonal como se explica en la sección 3.4.2. Los resultados fueron muy deficientes tanto en área y velocidad así como en consumo. En la figura 3.18 se muestra el área en CLBs y el retardo de las particiones en *ns* de los circuitos con *arquitectura 1* y *arquitectura 2* respecto de la partición *ortogonal*. En tanto, en la figura 3.19, se compara el consumo medido en mW/MHz para los mismos circuitos. Claramente y sin excepción la partición ortogonal muestra peores resultados.

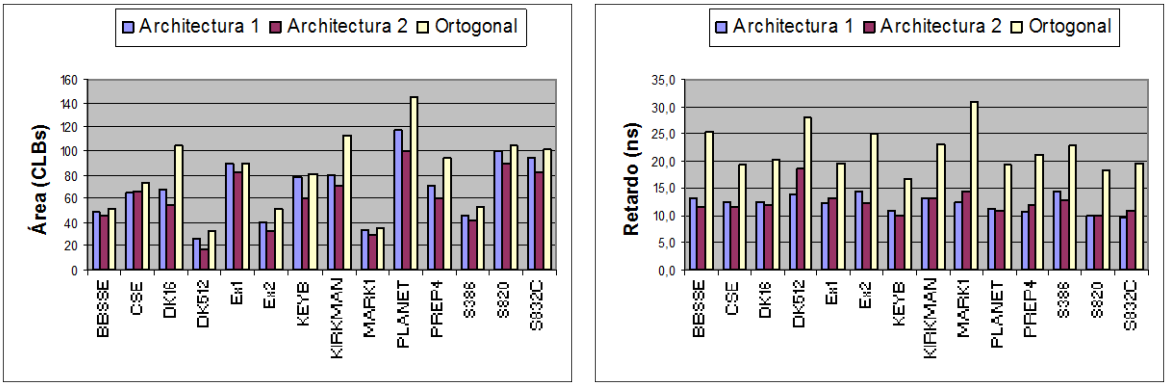


Figura 3.18. Área y retardo de la partición ortogonal respecto de las arquitectura 1 y 2

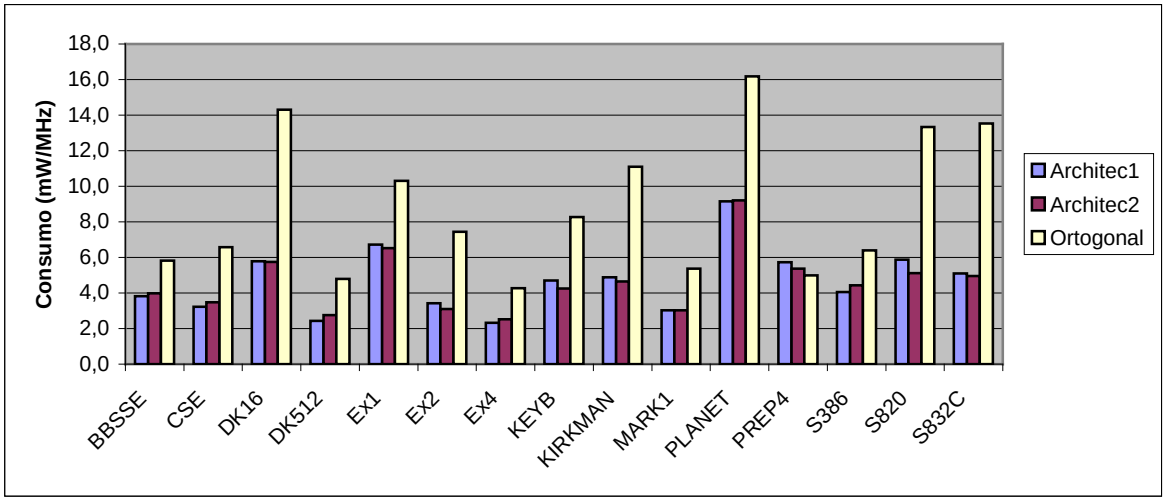


Figura 3.19. Consumo de la partición ortogonal respecto de las arquitecturas 1 y 2

3.4.11. 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. Importantes ahorros de energía (hasta un 46 %) se pueden obtener con esta técnica. Estos resultados pueden mejorar significativamente en dispositivos que disponen de latches nativos (Xilinx XC4000EX, Spartan2 o Virtex).

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, parte cabe señalar que los buenos resultados en cuanto al bajo consumo dependen de la actividad entre las máquinas de estados. Máquinas de estado donde no se puede generar particiones donde la actividad entre ellas sea pequeña hace esta técnica ineficaz.

3.5 Recomendaciones para la Reducción de Consumo en Máquinas de Estado Finitos

Los experimentos realizados en este capítulo, permiten generar la siguiente heurística para la minimización del consumo en máquinas de estados. La primer recomendación, aunque trivial, es llevar a cabo un diseño minimal de la máquina de estados. Existen innumerables programas (mayoritatiamente 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 sincronos 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 optima 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 usar 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. 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 57%

Capítulo 4:

Experimentos a nivel topológico

Este capítulo examina diferentes experimentos llevados cabo sobre el consumo en FPGAs con el objeto de determinar relaciones en el consumo, y "consejos" a nivel usuario respecto de las técnicas tener en cuenta. Otro objetivo implícito es el uso de herramientas de desarrollo como métricas indirectas para estimar o reducir el consumo. En éste capítulo se examinan la relación velocidad-consumo, la conmutatividad de datos y el efecto del pipeline y el uso de registros en general.

4.1 Introducción

Si bien los diseñadores de sistemas basados en FPGAs cuentan con herramientas para optimizar los circuitos en área y velocidad, pero 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 software 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, integrado en las últimas etapas de desarrollo. 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] a introducido una herramienta de estimación del consumo denominada XPOWER [Xpo02] 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 reportes que se generan para área y análisis de tiempo en los IDE de los fabricantes de FPGAs, son útiles también en diseño para bajo consumo.

4.2 Relación entre Velocidad y Consumo en FPGAs

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 de verificar experimentalmente esta proposición para FPGAs, lo cual abre una posibilidad para la estimación y optimización de consumo frente a la falta de herramientas EDA de diseño para bajo consumo para 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 *fanout*, la cantidad de CLBs o la profundidad de lógica, se optimiza 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. 4.1).

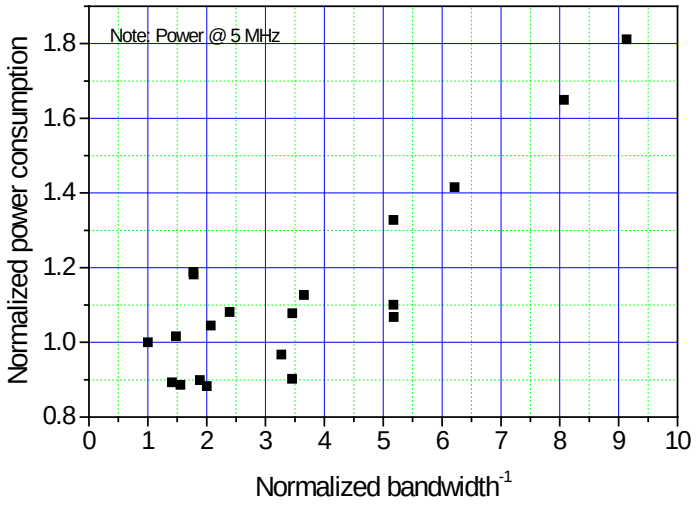


Figura 4.1. Relación ancho de banda-consumo para multiplicadores segmentados [Boe96].

La optimización de un parámetro de un IC como la velocidad, se puede atacar en diferentes niveles en el ciclo de diseño. A nivel de topología, se puede disponer de varias alternativas funcionalmente equivalentes. Por ejemplo para la adición binaria se tiene los sumadores *ripple-carry*, *carry-save*, *carry-skip*, *carry look-ahead*, *Brent & Kung*, 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 o segmentación. Por último, el circuito puede ser transformado a nivel físico: particionado manual, *floorplaning*, selección de un valor para el esfuerzo de rutado, restricciones de tiempo (*timming constraints*), etc.

En esta sección se exploran alternativas topológicas y físicas para multiplicadores implementados en FPGAs de la serie 4K 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.

4.2.1 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 [Wall64] [Hat86] [Gui69]. Las principales características de cada tipo de multiplicador se resumen en la tabla 4.1. En estos experimentos se incluye 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].

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.

Test circuit	Topology	Reference	Description language	Number of CLB	Logic depth in the 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 4.1. Principales características de los circuitos de prueba. Relación Velocidad-Consumo

4.2.2 Resultados Experimentales de la Relación Velocidad - Consumo

Los resultados más relevantes se muestran a continuación. Cabe destacar que el efecto en el consumo de las diferente topologías es (como es de esperar) mayor que la relación con la velocidad.

Correlación Velocidad-Consumo. Efecto de las Diferentes Topologías

Cada topología tiene sus propias características en cuanto a área y velocidad. Para obtener un circuito con bajo consumo, debe elegirse la topología que presente los menores valores de este parámetro. En la figura 4.2 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. 4.2, 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 síntesis. Es importante aclarar 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.

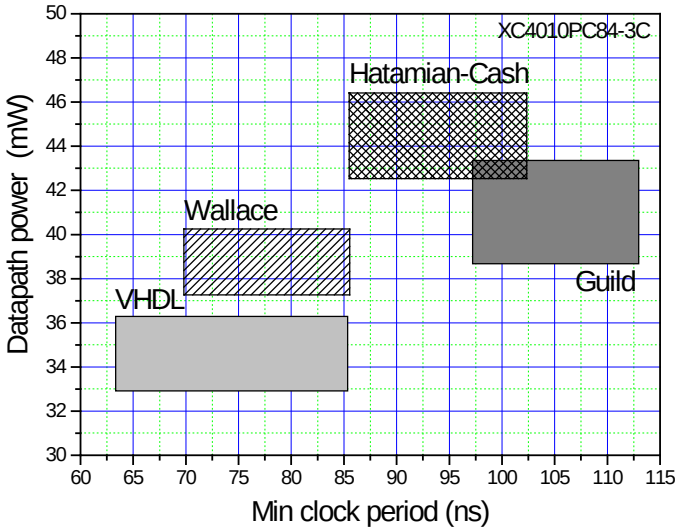


Figura 4.2. Consumo dinámica de los multiplicadores medidos.

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

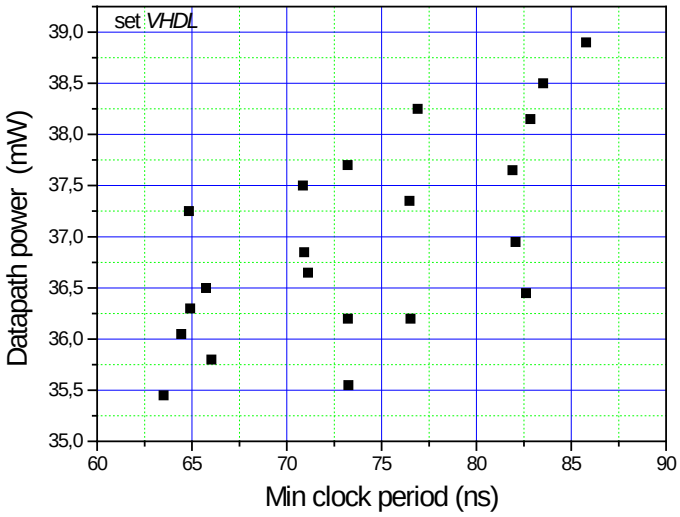


Figura 4.3. Relación velocidad-consumo para la misma topología (multipl. VHDL comportamental)

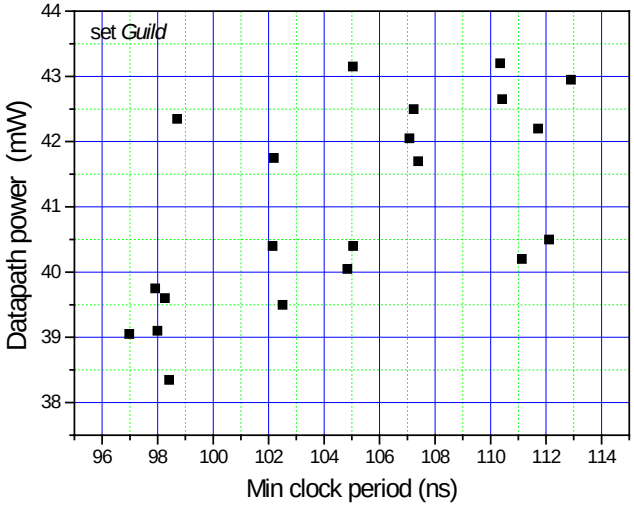


Figura 4.4. Relación velocidad-consumo dentro de la misma topología para el multiplicador Guid.

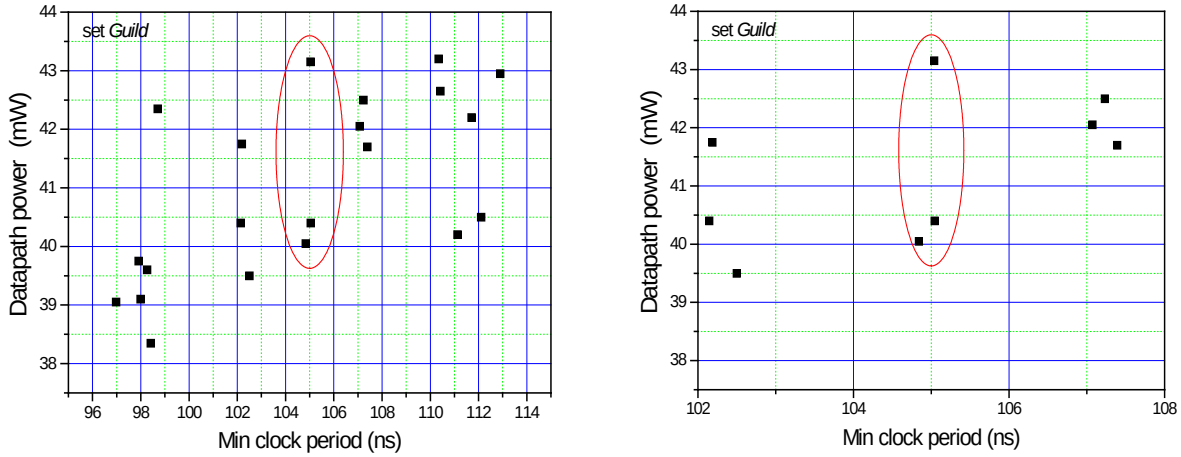


Figura 4.5. Estudio de las fluctuaciones en el consumo para circuitos de similares velocidades (Multiplicador Guid).

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 4.4 se muestra el resultados obtenido para el multiplicador Guild y en la figura 4.3 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.

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. 4.5.

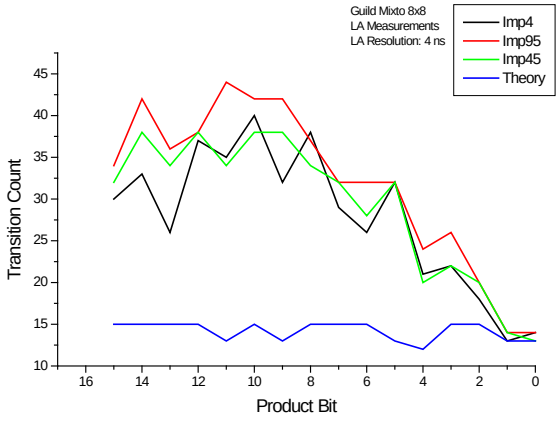


Figura 4.6. Transiciones a la salida de los multiplicadores seleccionados. Obtenida por Simulación

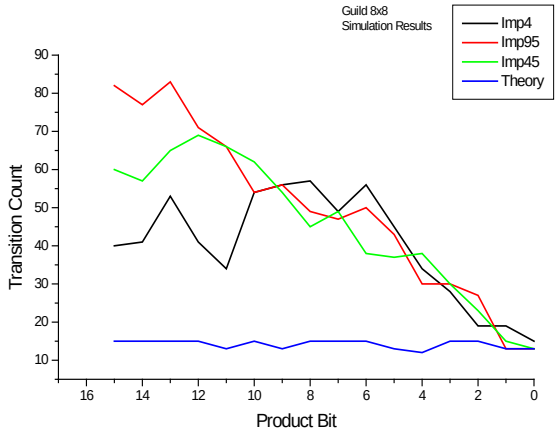


Figura 4.7. Transiciones a la salida de los multiplicadores seleccionados por Medición.

Como 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 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 se muestran en la Figura. 4.6 y 4.7.

Para resumir esta información se ha calculado la cantidad de transiciones por operación (TPO) según la Ecuación 4.1. Luego para la figura 4.8 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).

$$TPO = \frac{cantidad\_transiciones}{Numbero\_de\_oper} \qquad \text{(Ecuación 4. 1)}$$

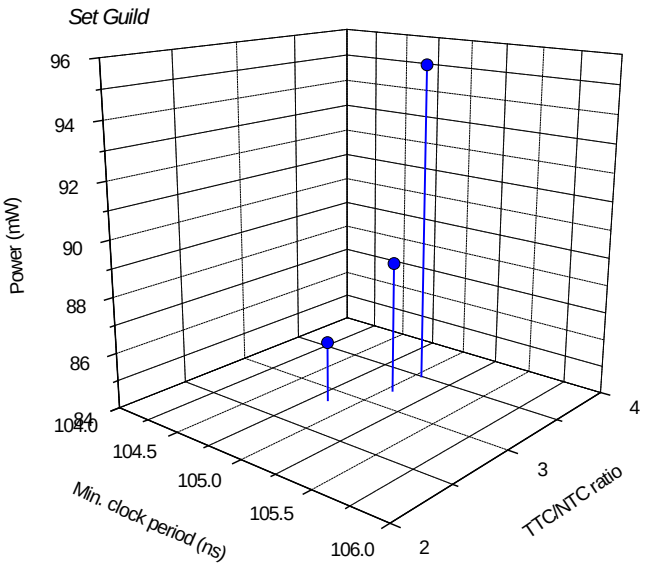


Figura 4.8. Relación entre actividad a la salida de los multiplicadores y consumo para implementaciones con similar velocidad.

En la figura 4.8 se observa que la cuenta de transiciones se puede usar 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 para el consumo.

Correlación Área-velocidad-consumo (Area-Time-Power)

No existe una clara correlación entre estos parámetros. Si se comparan los cuatro multiplicadores (figura 4.9). 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 medido 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

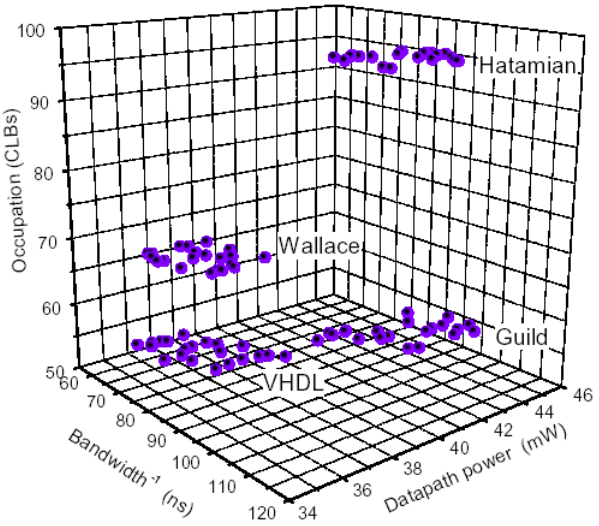


Figura 4.9. Relación entre actividad a la salida de los multiplicadores y consumo para implementaciones con similar velocidad.

4.2.3 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 reportes de temporización generados durante el ciclo de diseño para, indirectamente, mejorar el consumo. Estas recomendaciones se pueden complementar con las propuestas en un reciente trabajo para el nivel arquitectónico [Gar99]. 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 relación entre área y consumo no es tan clara como sucede en los circuitos basados en celdas. Algunas técnicas que ganan velocidad a expensas de mayor utilización de recursos (opciones del tipo duplicar hardware para disminuir fanout) contribuyen a reducir consumo sin ser significativo el aumento de consumo por la mayor utilización de CLBs.

4.3 Conmutación de los Datos de Entrada (Propiedad Conmutativa y Diseño de Bajo Consumo)

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 7 multiplicadores binarios utilizando FPGAS de la serie 4K de Xilinx, donde ha podido observarse reducciones de consumo cercanas al 8 %. Adicionalmente se han realizado medidas para multiplicadores implementados en la familia Virtex observándose diferencias en el consumo de hasta el 39 %.

4.3.1 Introducción

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 del *chip*.

La potencia en circuitos CMOS corresponde principalmente al consumo dinámico; es decir, la energía necesaria en cada ciclo para cargar y descargar la capacidad de cada nodo del circuito. Esta componente del consumo se modela con la expresión:

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. En un sistema síncrono correctamente diseñado, la actividad espuria no afecta a los resultados finales de cada ciclo de operación, pero contribuyen significativamente al aumento de la conmutación de los nodos del circuito (Ver sección 2.3.3).

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.

La asimetría del consumo se puede reforzar por un segundo mecanismo. Para ello, se deben combinar dos factores que aparecen a menudo en procesamiento de señal: la difusión (*broadcast*) de uno de los operandos se realiza a través de pistas globales y 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 4.10 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 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 este artículo se realizan varios experimentos sobre los argumentos anteriores. En la siguiente sección se resumen las características de los circuitos de prueba. En la sección 3 se resumen los métodos de medida y se muestran los principales resultados.

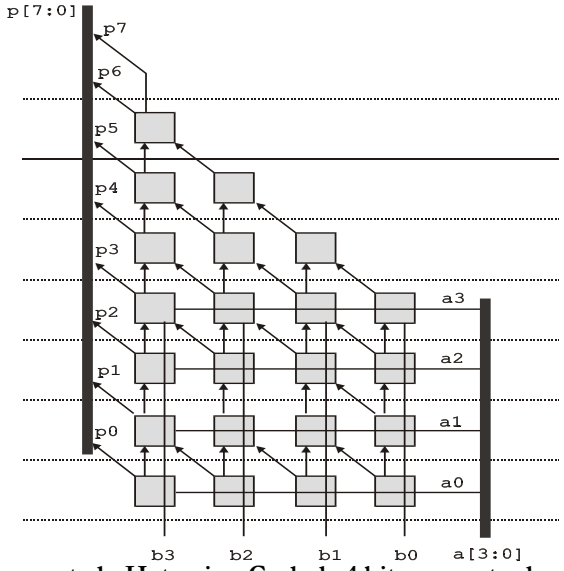


Figura 4.10. Arreglo segmentado Hatamian-Cash de 4 bits con entradas *b* locales y *a* globales.

4.3.2 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 Xilinx de la serie 4K. 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 4.2. La profundidad lógica se ha especificado siguiendo la forma adoptada por el fabricante; esto es, computando el nivel de lógica correspondiente al primer registro. 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 arrays segmentados con diferentes profundidades de lógica.

Topología	Ref.	CLBs	FF	Prof. de lógica (max.) LUTs	BW normalizado MHz
Guild-16	[8]	60	32	16	21.0
VHDL-12	[9]	56	32	12	32.1
Wallace-12	[10]	71	32	12	29.5
Hatamian-8	[11]	75	54	8	25.8
Hatamian-3	[11]	112	207	3	66.2
Hatamian-2	[11]	207	404	2	70.9
Xcore-9	[12]	52	96	9	78.1

Tabla 4.2. Principales características de los circuitos de prueba.

4.3.3 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:

$$PR = 100 * \frac{P_{original} - P_{modificación}}{P_{original}} \qquad \text{(ecuación 4.2)}$$

Esquema de Medición

Todos los prototipos fueron medidos en la misma placa de prueba, utilizando idéntica asignación de pines 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 pin de salida (para números aleatorios) y 0,18 mW/MHz para la secuencia de máxima actividad.

Se utilizó una FPGA XC3130 para generar cuatro juegos diferentes de vectores, cuyas características se resumen en la Tabla 4.3. 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 [14]. Los lectores interesado en otros métodos de medición de consumo pueden consultar las referencias [15] y [16]. El error de medición es menor que 0,3 %. Detalles del arreglo experimental 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 4.3. Descripción de los vectores de prueba

Resultados

En la figura.4.11 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.

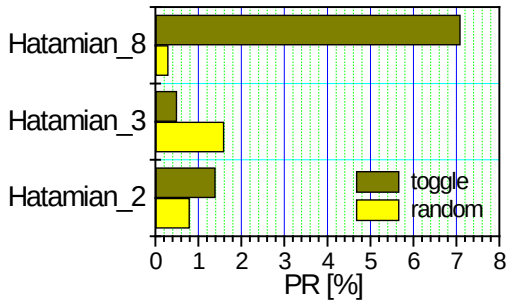


Figura. 4.11. Reducción de consumo. Arrays Hatamian-Cash. XC4010PC84

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 4.12, 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.

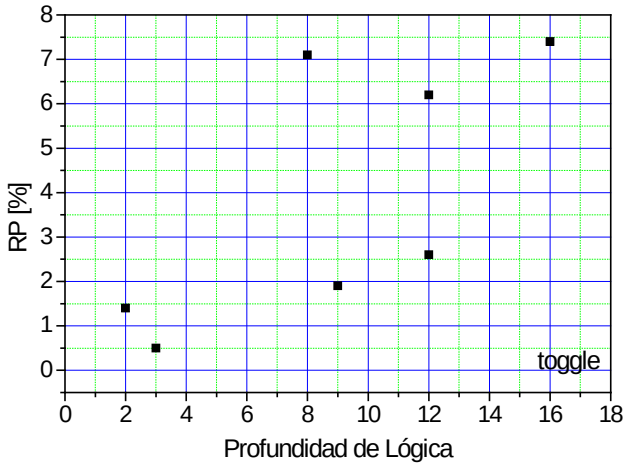


Figura 4.12. Reducción de consumo en función de la profundidad de lógica de los circuitos. XC4010PC84

Utilizando una herramienta informática [Tod02] se midió la cantidad de transiciones para un multiplicador Hatamian-8 con la secuencia toggle-8 donde se aprecia la diferencia de actividad (resumen en tabla 4.4). Esta 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	A×B	B×A
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	A×B	B×A
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	A×B	B×A	Nodo	A×B	B×A
P[15]	42	42	P[7]	71	71
P[14]	54	54	P[6]	69	69
P[13]	67	67	P[5]	68	68
P[12]	64	64	P[4]	61	61
P[11]	69	69	P[3]	48	48
P[10]	60	60	P[2]	81	81
P[9]	60	60	P[1]	49	49
P[8]	61	61	P[0]	38	38
			Total Pad_P	962	962

	A×B	B×A
Total nodos analizados	580	580
Cantidad Total de Transiciones	47476	52158
Promedio transición por nodo	81,86	89,93

Tabla 4.4. Actividad de las entradas, salidas y totales para la operación A×B y B×A con vectores de máximo toggle-8

El conjunto de arrays 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 4.13 se muestran los resultados. Se han elegido las medidas correspondientes al conjunto de vectores aleatorios, 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 array 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.

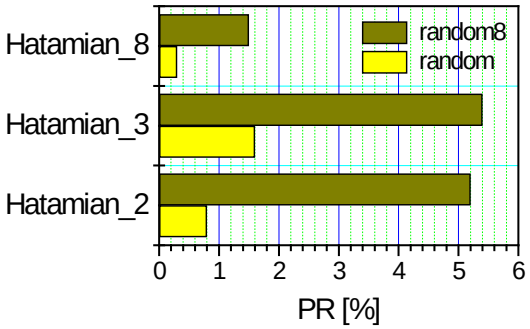


Figura 4.13. Efecto de las líneas globales. XC4010PC84.

Finalmente, en la figura 4.14 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.

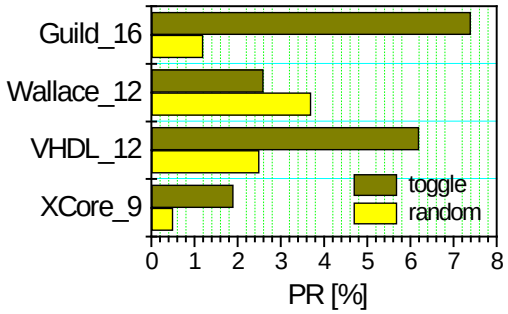


Figura 4.14. Reducción de consumo para diferentes topologías. XC4010PC84.

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 4.15 se observa las variaciones de consumo para el multiplicador Hatamian-8 para las secuencia toggle-8 y Random-1 en los diferentes circuitos.

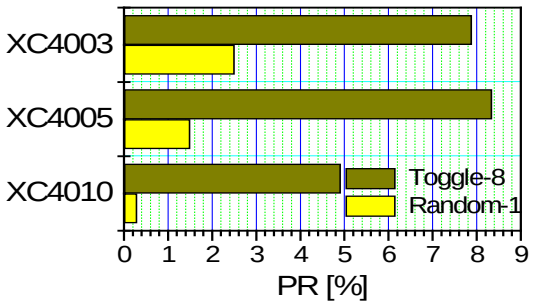


Figura 4.15. Reducción de consumo para diferentes circuitos familia XC40XXPC84 (multiplicador Hatamian-8)

4.3.4 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. Las pruebas se han realizado utilizando dos juegos diferentes de vectores, uno de máxima actividad en una entrada y prácticamente nula 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 4.5. Principales características de los circuitos de prueba. Familia Virtex

Cuatro de los circuitos de prueba son multiplicadores descriptos 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[Syn02]). 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 (ver sección 4.4). Las características de área y velocidad de los circuitos se resumen en la Tabla 4.5.

4.3.5 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 4.6 y 4.7 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 4.6. 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 4.7. 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 puede observar que el signo de la diferencia de consumo se mantine tanto para multiplicadores de 16 y 32 bits como para secuencias de máximo movimiento (MaxTog) y movimientos aleatorios (AvgTog)

Relación Retardo-Consumo

En la figura 4.16 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 4.2, donde se puede ver una clara correlación consumo promedio normalizado respecto del retardo para ambos conjuntos de patrones de entrada.

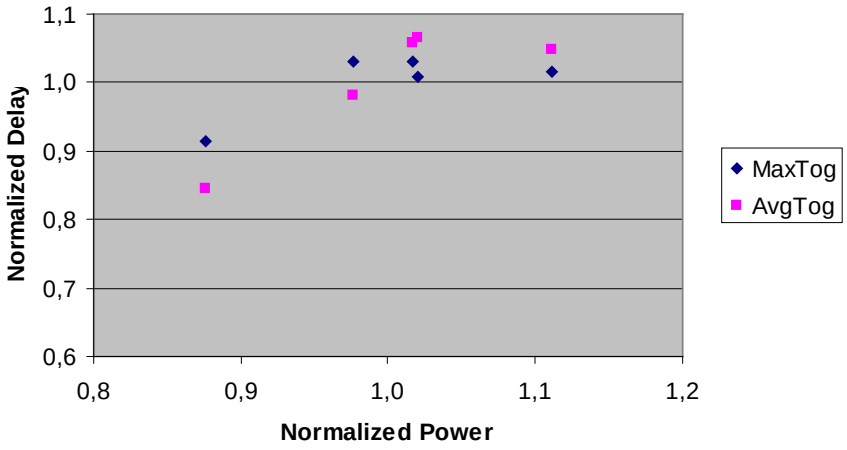


Figura 4.16. Relación retardo-consumo para los multiplicadores de 32 bits.

Uso de Herramientas de Estimación de Consumo

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 puede observar la forma de utilización de dicho software.

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 4.8.a 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 4.8.b 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 4.8.c 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 4.8.d Relación medidas – estimaciones para multiplicadores de 16 bits y secuencia AvgTog.

En las tablas 4.8 se puede ver el consumo expresado en mW/mHz de consumo dinámico tanto para las mediciones como para las estimaciones de Xpower. En las 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 4.17 muestra 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 via 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 4.17). Lo cual nos permite concluir que las estimaciones de consumo dinámico son de algún modo pesimistas. Solo el consumo por estimación del circuito sintizado por Synplify [Syn02] es menor que la media.

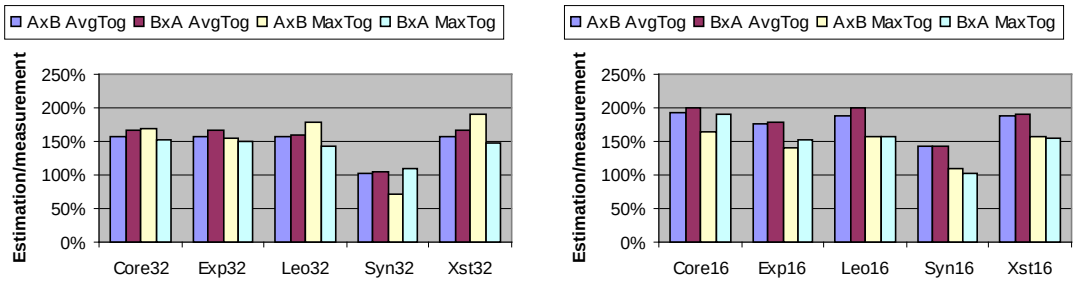


Figura 4.17. Diferencia en el consumo estimado y medido multiplicadores de 32 y 16 bits

4.3.6 Conclusiones de la Conmutación de Datos

Se ha analizado en detalle 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 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.

4.4 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. En los circuitos electrónicos la aplicación de las técnicas de *pipeline* se remonta a los inicios de esta disciplina. La relación del consumo con el *pipeline* proviene del hecho de que la segmentación reduce la actividad espuria (*glitches*).

Si bien los *glitches* no producen errores en los diseños síncronos bien diseñados pueden ser responsables hasta de un 70 % de la actividad del circuito [she92]. Como se describió en el capítulo 2, los *glitches* pueden se 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].

En esta sección se estudian los efectos de la segmentación en el consumo. En primer lugar se presentan resultados para multiplicadores de 8 bits sobre diferentes dispositivos de la familia XC4K, luego se analiza los resultados para multiplicadores de 32 bits implementados en la familia de dispositivos Virtex.

4.4.1 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 descriptos en el apartado 4.3.3 a los expuestos en la tabla 4.5. 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 4.18) 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 secuencial) 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 esta la componente más importante en un circuito altamente segmentado.

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	43,6	54,6	42,4
Hatamian-3	112	207	3	9	66,2	75,4	---
Hatamian-2	207	404	2	17	70,7	---	---

Tabla 4.9. Principales características de las diferentes topologías de prueba 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 de emplazamiento y por ello 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 empeora.

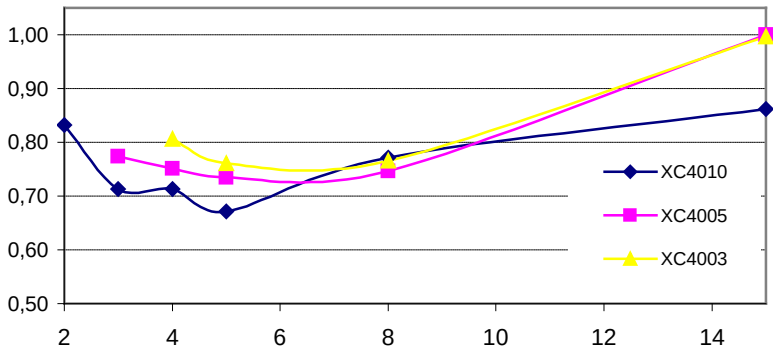


Figura 4.18. Consumo normalizado respecto de la profundidad lógica en XC4K

4.4.2 Mediciones sobre Virtex

Se han construido multiplicadores tipo *shift & add* de 32 bits con distintos niveles de segmentación. La tabla 4.10 muestra las principales características de los circuitos. Adicionalmente, los circuitos son registrados a la entrada y salida, lo que agrega dos ciclos de reloj para obtener el resultado. El nivel de segmentación es de cantidad de retardos menos dos.

Circuitos	Slices	Flip Flops	Prof. Lógica (LUTs)	Retardos (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 4.10. Principales características de los circuitos de prueba para Virtex.

Se utilizaron dos tipos de patrones de excitación, uno de actividad aleatoria y otro de máxima actividad. El consumo normalizado en función de la profundidad lógica se puede observar en la figura 4.19. 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 es mucho menos

significativo.

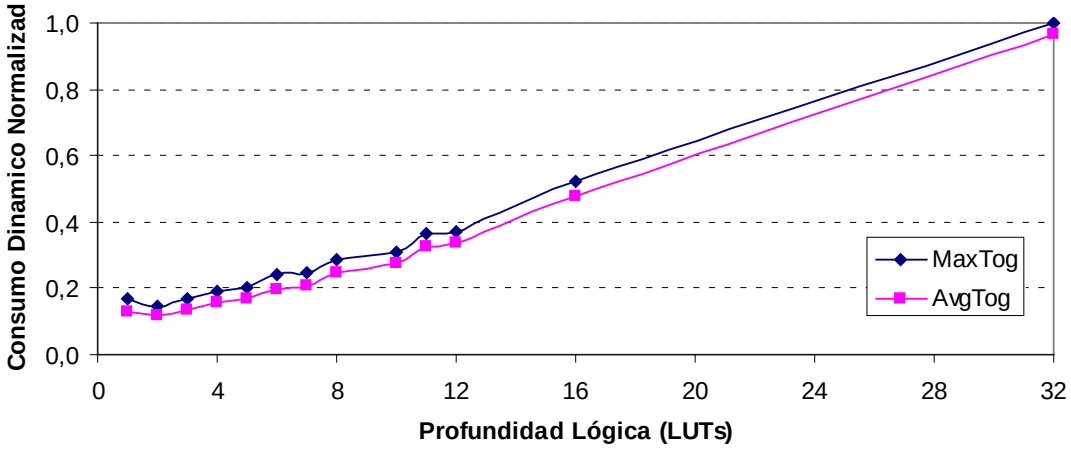


Figura 4.19. Consumo normalizado respecto de la profundidad lógica en Virtex

En la figura 4.20 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 (aumentar la segmentación) decae el retardo casi de forma lineal, al igual que 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 los 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.

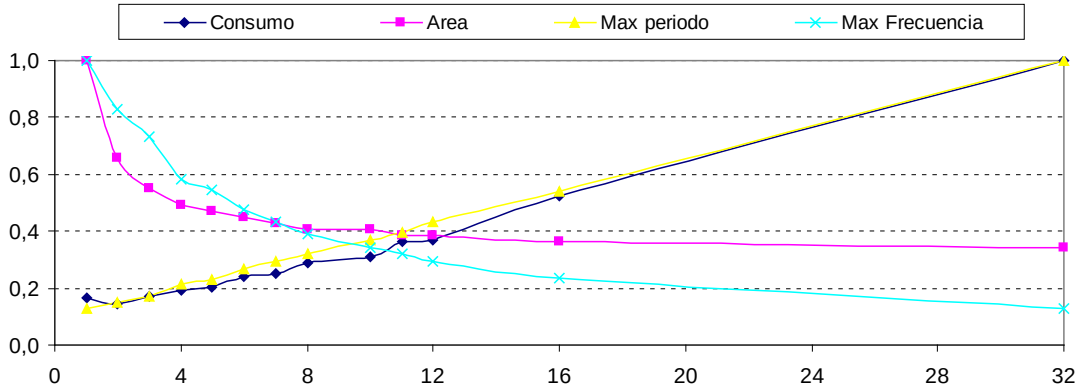


Figura 4.20. Consumo, área, retardo normalizados respecto de la profundidad lógica en Virtex

Uso de herramienta de estimación de consumo

Se ha utilizado la herramienta XPOWER [Xpo02] para estimar el consumo de los multiplicadores descriptos anteriormente. En la tabla 4.11 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 (consumo de sincronización), en tanto que la siguiente el porcentaje de consumo sincronización respecto al consumo aleatorio. En la figura 4.21 se observan la comparación de los datos de consumo medido respecto de los estimados.

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 4.11. Consumo medido respecto del estimado para multiplicadores segmentados.

En este caso se observa que la estimación con XPOWER no parece ser demasiado precisa, sobre todo en la versión totalmente combinacional (mult_pipe32).

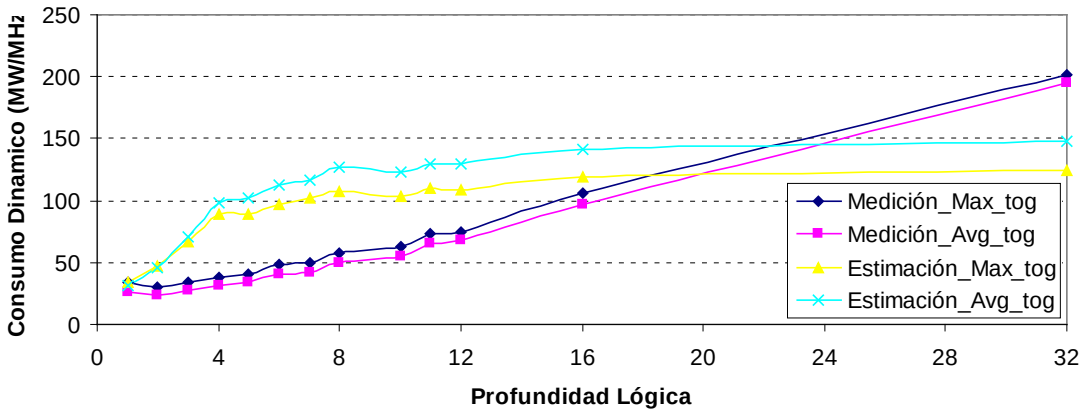


Figura 4.21. Consumo medido y estimado en función de la profundidad lógica.

4.4.3 Conclusiones sobre la Segmentación

El 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 es lo ideal, en tanto que para la familia Virtex una segmentación cercana al máximo posible es lo ideal.

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 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 este circuito - profundidad lógica 2) se puede reducir el consumo dinámico a un 12% del original. Finalmente se ha puesto a prueba la herramienta de estimación de consumo Xpower no obteniéndose resultados aceptables.

4.5 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.

4.5.1 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.

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 4.22 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 segunda figura se muestra la distribución de uno de los bits de los operandos (concretamente a<7>) y se ve la diferencia en recursos utilizadas.

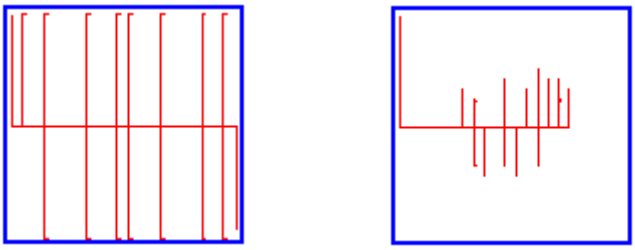


Figura 4.22. Distribución de la línea global de reloj. a)Flip-flop en las patas de entrada salida; b) usando los Flip-Flop de los CLBs

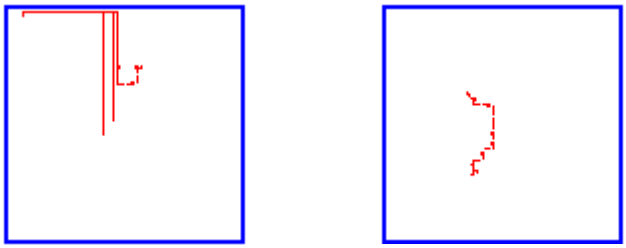


Figura 4.23. Distribución de la línea de entrada a<7> con fan-out de 16. a)Flip-flop en las patas de entrada salida; b) usando los Flip-Flop de los CLBs

En la figura 4.24 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.

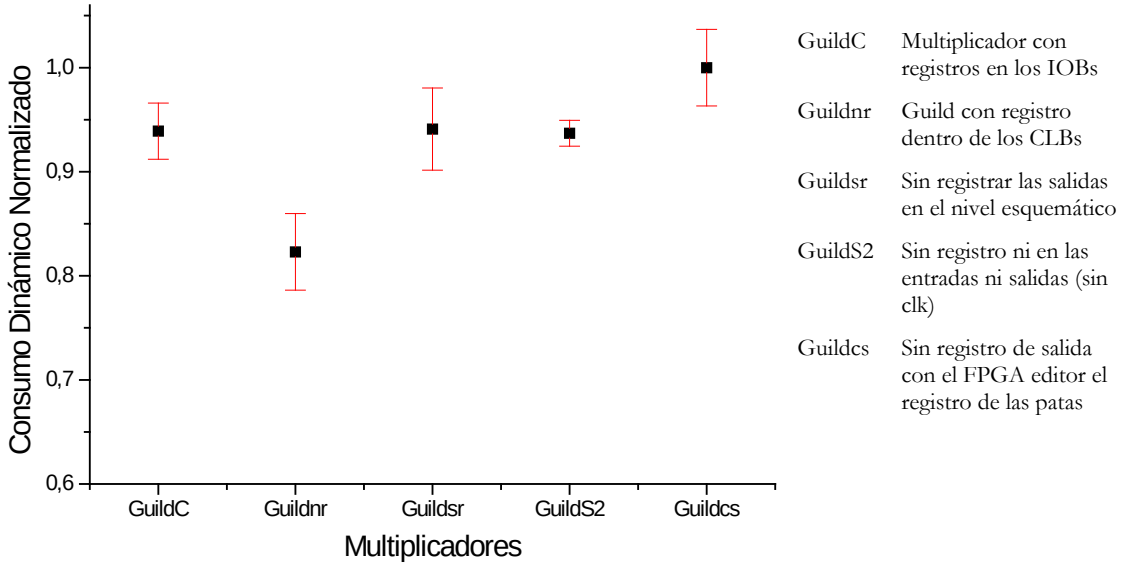


Figura 4.24. Consumo normalizado con registros en IOBs y CLBs.

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.

4.5.2 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 4.12 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	--
	Mult-ff-IOB	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	--
	Mult-ff-IOB	251	85	628	281	908	37%
	No-ff-out	255	179	638	591	1228	85%
	No-FFs	254	175	635	578	1213	83%

Tabla 4.12. Diferencia de consumo dinámico dependiendo del tipo de registros.

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). El 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.

4.5.3 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 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%.

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.

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 usarlo, no solo por la disminución de consumo sino también por la fiabilidad de los datos.

4.6 Conclusiones del Capítulo

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 usuario. En éste capítulo se examinó la relación velocidad-consumo, la conmutatividad de datos y el efecto del pipeline y el uso de registros en general. Las principales conclusiones se resumen a continuación:

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 relación entre área y consumo no es tan clara como sucede en los circuitos basados en celdas. Algunas técnicas que ganan velocidad a expensas de mayor utilización de recursos (opciones del tipo duplicar hardware para disminuir fanout) contribuyen a reducir consumo sin ser significativo el aumento de consumo por la mayor utilización de CLBs.

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 es lo ideal, en tanto que para la familia

Virtex una segmentación cercana al máximo posible es lo ideal. 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.

- En la familia Virtex 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 este circuito - profundidad lógica 2) se puede reducir el consumo dinámico a un 12% del original. Finalmente se ha puesto a prueba la herramienta de estimación de consumo Xpower no obteniéndose resultados aceptables.

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%.

- 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.

- 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 usarlo, no solo por la disminución de consumo sino también por la fiabilidad de los datos.

Capítulo 5:

Experimentos a Nivel Algorítmico

Este capítulo examina algunas alternativas a nivel algorítmico. Por un lado se examinan las opciones para la multiplicación modular, operación central en los cálculos criptográficos y por otro se presenta un experimento cuyos resultados no fueron los esperados desde el punto de vista del consumo como es la suma por el algoritmo de *carry-skip*.

5.1 Introducción

En la sección 2.7 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 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 usados por el algoritmo, así como la correlación de datos que se producen producto de los algoritmos 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 dan estimaciones de los costes correspondientes tanto para las versiones combinacionales como secuenciales. Por último se muestran los resultados prácticos para implementaciones combinacionales, así como para arquitecturas totalmente secuénciales utilizando dispositivos programables de la familia XC4K.

5.2.1 Introducción

La seguridad se ha convertido en un aspecto fundamental de los sistemas informáticos. Para hacer frente a esta necesidad se deben utilizar algoritmos criptográficos, tanto para el cifrado de los datos confidenciales como para la identificación de los emisores y receptores de los mismos. Para mejorar tanto las prestaciones como la seguridad física del sistema puede ser conveniente que los algoritmos criptográficos sean ejecutados por procesadores específicos, es decir, ASICs o dispositivos programables por el usuario (FPGAs). Con estos últimos se evitan los altos costes no recurrentes de los primeros. Por los motivos mencionados antes, la generación de modelos VHDL sintetizables, que permitan integrar las principales primitivas criptográficas en una FPGA y con la mejor figura de consumo posible.

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 $y^x \bmod m$, 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 tres algoritmos: multiplicación y reducción, sumas y desplazamientos, multiplicación de Montgomery.

5.2.2 Algoritmos

Dados tres números naturales x, y y m tales que

$$x < m, y < m \text{ y } m < 2^n,$$

Se proponen tres algoritmos – con los circuitos correspondientes – para calcular

$$z = x.y \bmod m.$$

A. 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 *shift and add*:

Algoritmo 1

```
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 es parecido al de división a mano:

Algoritmo 2

```
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) utiliza como primitiva de cálculo un sumador / restador cuya operación depende de una condición binaria previamente calculada:

Algoritmo 3

```
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.

B. Sumas y desplazamientos (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 4

```
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

```
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. 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 $k = m\text{-}y$ 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 6

```
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;
```

C. 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 7

```
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 $2^{2.n} \bmod m$, 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 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 $two\_m = 2^{2.n} \bmod m$ 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 9

```
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, es decir, multiplicación y reducción, sumas y desplazamientos, y producto de Montgomery, han sido sintetizados en forma de circuitos combinacionales e integrados en una FPGA XC4025. Para la síntesis de los algoritmos (1 y 2 o 3, 6 y 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 - Configurable Logic Blocks) 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 tablas (look-up tables). La conclusión sería bastante diferente en el caso de la versión Standard Cell de las mismas primitivas.

A. Multiplicación y reducción

La multiplicación (algoritmo 1) incluye n sumas condicionales. El coste correspondiente es igual a $n.(n/2 + 1)$ CLBs. Para ejecutar la reducción con el algoritmo 2 se necesitan n restadores condicionales y n multiplexores, mientras que el algoritmo 3 sólo incluye $n-1$ sumas / restas, una resta inicial y una suma condicional final. El coste correspondiente es $(n+1).(n/2 + 1)$ CLBs. La figura 5.1 muestra un esquema del reductor modulo m. El coste total es igual a:

$C_{\text{multiplicación y reducción}} = n^2 + 2{,}5.n + 1$ CLBs.

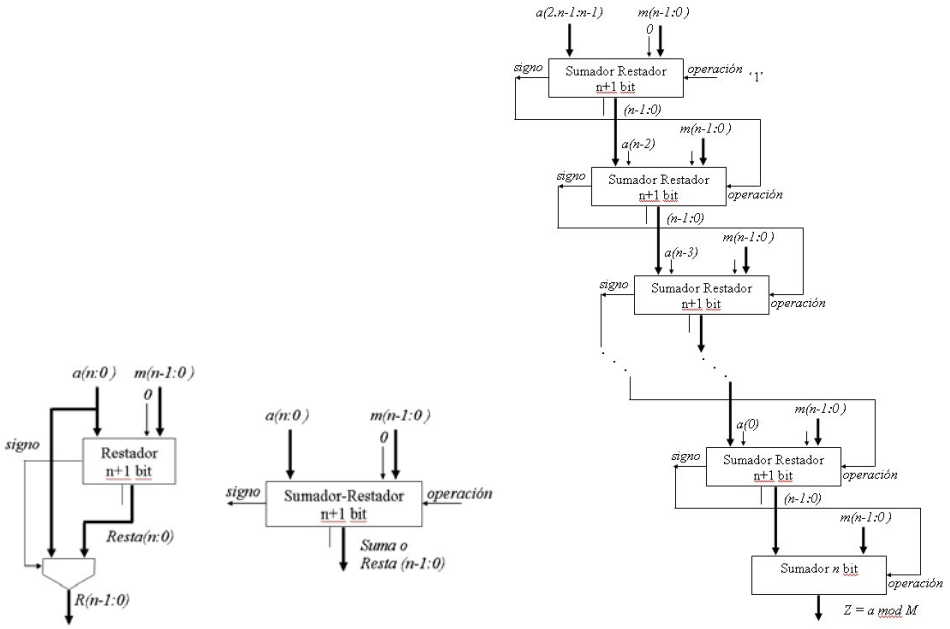


Figura 5.1. Reductor modulo M. a) Celda elemental. b)Implementación combinacional.

B. Sumas y desplazamientos

Para la ejecución del algoritmo 6 se necesitan:

un primer restador (cálculo de *k*),
n multiplexores (selección de *w*),
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

$$C_{\text{sumas y desplazmientos}} = 2.n^2 + 3{,}5.n + 1 \text{ CLBs.}$$

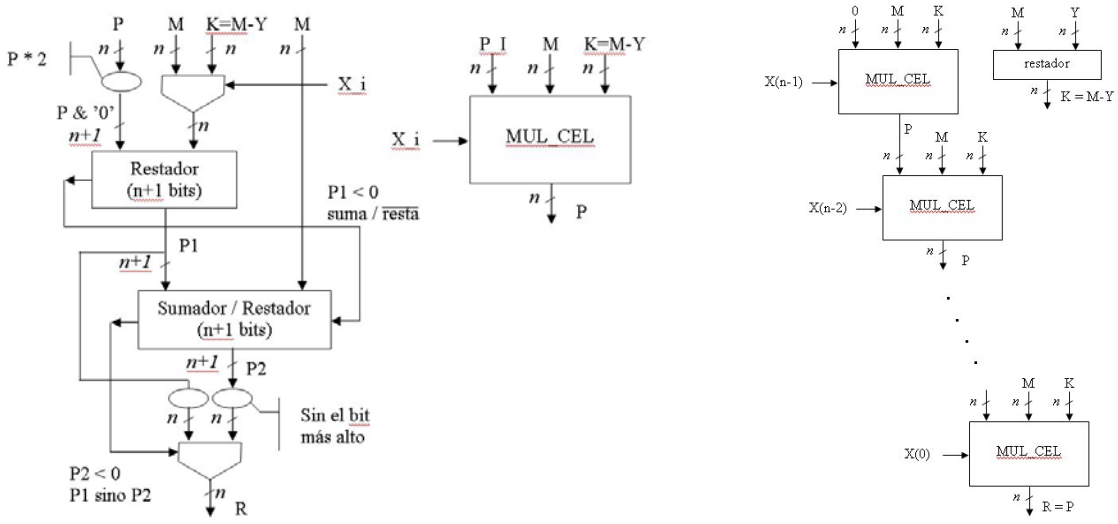


Figura 5.2. Multiplicador shift and add. a) Celda de cálculo. b) arreglo combinacional.

C. Multiplicación de Montgomery

El algoritmo 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.

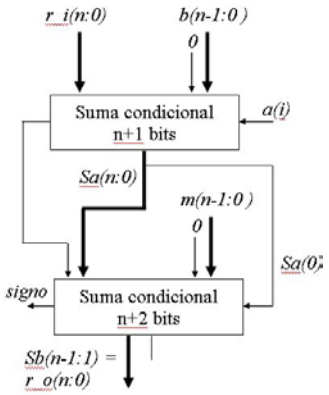


Figura 5.3. Celda de cálculo del multiplicador de Montgomery.

Comparación

Los tres tipos de multiplicadores modulares (*m_r*: multiplicación y reducción, *s_d:* sumas y desplazamientos, *mont.:* multiplicador de Montgomery) han sido integrados en una matriz programable XC4025E. La descripción inicial era 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 da el número de CLBs y el retardo máximo (en nanosegundos, sólo para 8 y 16 bits dado que los multiplicadores de 24 y 32 bits no caben dentro de la matriz elegida).

	Area (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 en una matriz 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 7) es más rápido.
2. El cálculo de la función exponencial con la multiplicación de Montgomery (algoritmo 9) necesita el cálculo previo de 2^n mod m y de $2^{2.n}$ mod 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 $z = x.y$ mod m sino $z'' = x.y.2^{-n}$ mod m. Para obtener z a partir de z'' es necesario haber calculado previamente el valor de $2^{2.n}$ mod m y se debe ejecutar un segundo producto de Montgomery dado que $z = z''.2^{2.n}.2^{-n}$ mod m. Por tanto el método de Montgomery es ineficiente para realizar una multiplicación modular aislada (cuando no sirve para calcular la función exponencial modular).

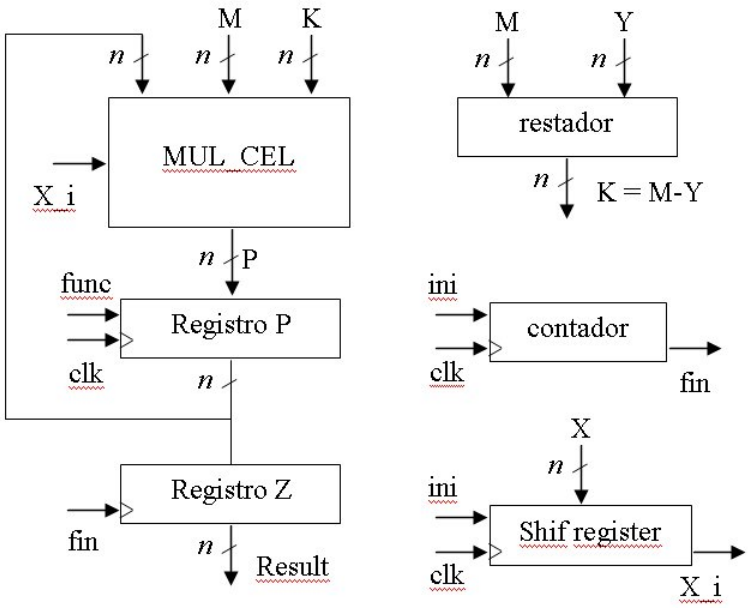


Figura 5.4. Esquema de la implementación secuencial del algoritmo de sumas y desplazamientos.

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 1 y 3 (*m_r*), 6 (*s_d*) y 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 matriz que antes (XC4025E).

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 1 y 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 6 (sumas y desplazamientos) se usan

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 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 tres tipos de multiplicador han sido sintetizados e integrados en una FPGA XC4025E. Los resultados de la integración se recogen en las tablas 5.2 (número de CLBs y frecuencia máxima en megahertzios). 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.

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

Para medir el consumo de los diferentes multiplicadores se utilizo el arreglo experimental y la metodología descripta en el apéndice A. Adicionalmente cada circuito fue medido a 100 Hz, 2, 3, 4 y 5 MHz para reducir los errores en la medición. El código VHDL fue sintetizado con FPGA express [Fpg99][Fpg01] y el entorno de desarrollo de Xilinx [Xil00] en un dispositivo XC4010EPC84-1.

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.*
Power (mW/Mhz)	96,0	186,4	92,7
Area (CLBs)	85	157	102
Time (ns)	186	201	167

Tabla 5.3 Área, retardo y consumo (Area-Time-Power) 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.

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 pueden 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.
Dynamic Power (mW/Mhz)	8,94	6,55	4,82
Synchronization Power (mW/Mhz)	5,85	3,28	3,40
Combinational Power (mW/Mhz)	3,09	3,27	1,39
Area (CLBs)	57	33	34
Flip - Flops	67	37	31
Time (Mhz)	25	17,2	32,1

Table 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 (potencia x tiempo) 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 de la acumulación de glitches. La potencia de sincronización en estos casos es mayor que la potencia combinacional.

5.2.6. Conclusiones 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_a*) 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) y del 54% en el caso secuencial (Montgomery vs. Multiplicar y reducir). Además la energía consumida en las versiones secuenciales es menor debida a la disminución de los glitches.

5.3. Sumadores de alta velocidad.

En esta sección se analiza la el algoritmo de adición de alta velocidad conocido como *carry skip*. De los algoritmos alternativos al del sumador *ripple-carry*, el *carry skip* parece por su estructura que ha de haber de tener menos *glitches* producto de una menor propagación del acarreo y consecuentemente menos consumo. Basándose en esta idea y en el hecho probado en el capítulo anterior que los circuitos más veloces consumen menos se construyeron circuitos cuyas mediciones negaron la suposición inicial.

5.3.1 Introducción

Esta sección se ocupa de la adición de operadores largos, es decir la primitiva básica en la mayor parte de las operaciones aritméticas. La suma con grandes operandos es de gran utilidad en las operaciones criptográficas.

Aunque las técnicas de *carry-look-ahead* (CLA) parecen ser más veloces que las de *carry skip* ([Kor02] [Par00] [Ober01]), en algunas tecnologías especificas esto no puede ser realmente explotado. De hecho la implementación del CLA en FPGAs no tiene buenos resultados. 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.

5.3.2 Sumador Ripple-Carry

Las FPGAs en general poseen recursos de computación para generar sumadores rápidos [Xil01] [Xil03], como ejemplo las familias Spartan2 y Virtex incluyen puertas lógicas y multiplexores que junto a las tablas de *look-up* de de propósito general permiten construir sumadores de tipo ripple-carry de forma eficiente. En la figura 5.5 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 *slices* 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*).

$$p(i) = x(i) \text{ xor } y(i)$$

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 computo de una *lookup table* 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:

$$C_{adder}(n) = n/2 \text{ slices}$$

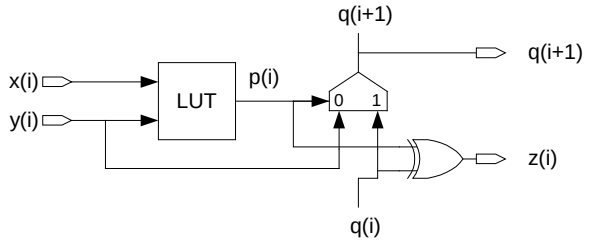


Figura 5.5. Celda que utiliza la cadena de acarreo en las FPGAs Xilinx Spartan2 / Virtex

5.3.3 Sumador Carry-Skip

Dado el hecho de que $t_{lut} >> t_{muxcy}$, usar 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 implementaciones eficientes se utilizan en el cálculo de los productos *P(i)*, así como, para seleccionar la salida de cada grupo de *s*-bits multiplexores dedicados.

Un grupo sumador del *carry-skip* para *s*-bits se muestra en la figura 5.6. El tiempo de computación y área utilizados son los tradicionales para un sumador *ripple-carry* normal.

$$T_{adder\_group} = t_{lut} + s.t_{muxcy} \quad \text{y} \quad C_{adder\_group} = s/2 \text{ slices.}$$

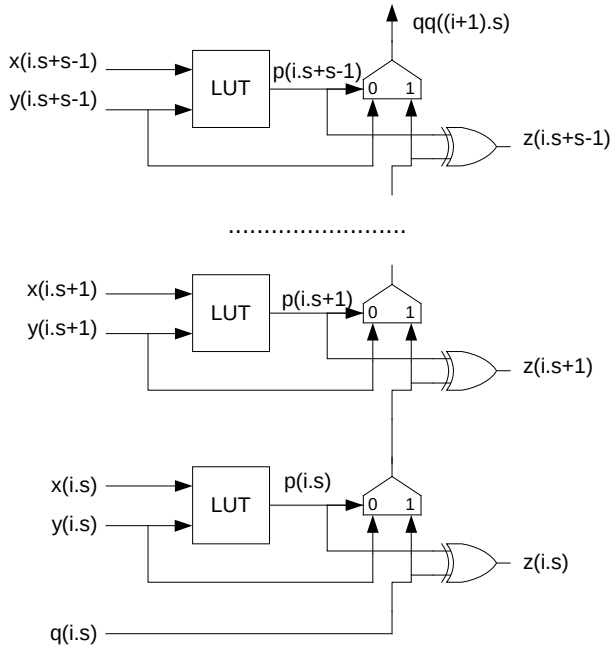


Figura 5.6. Grupo sumador de s bits para el carry-skip

El grupo de multiplexores que seleccionan la salida de cada grupo de sumadores pertenece al camino crítico del circuito, por ello es que se implementan utilizando multiplexores de acarreo como se puede ver en la figura 5.7. Obsérvese que para conectar P(i) a los multiplexores internos no hay otra forma más que conectarlo a traves de tablas de look-up. Los tiempos de propagación y área son:

$t_{lut} + (n/s - 2).t_{mux\text{-}cy}$ y $(n/s - 2)/2$ *slices*

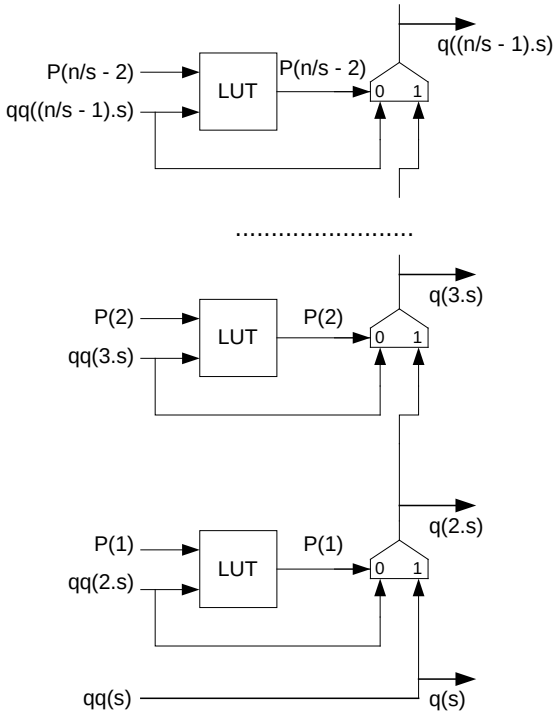


Figura 5.7. Multiplexores de salto de acarreo (carry-skip multiplexers)

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 5.8, cada look-up table 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:

$t_{lut} + (s/2).t_{mux\text{-}cy}$ and $s/4$ *slice*s.

La estructura completa de un sumador *carry-skip* (con $n/s = 4$) se puede ver en la figura 5.9. El tiempo de cálculo (parte sombreada del gráfico) es igual a:

$$T_{adder} = t_{lut} + max\ \{(2.s + n/s - 3).t_{mux\text{-}cy}., t_{lut} + (1{,}5.s + n/s - 3).t_{mux\text{-}cy}.\} + 2.t_{connection} + t_{xor2},$$

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.

$$s \cong (n/2)^{1/2} \text{ and } n/s \cong (2.n)^{1/2}, \text{ or } s \cong (n/1{,}5)^{1/2} \text{ and } n/s \cong (1{,}5.n)^{1/2}.$$

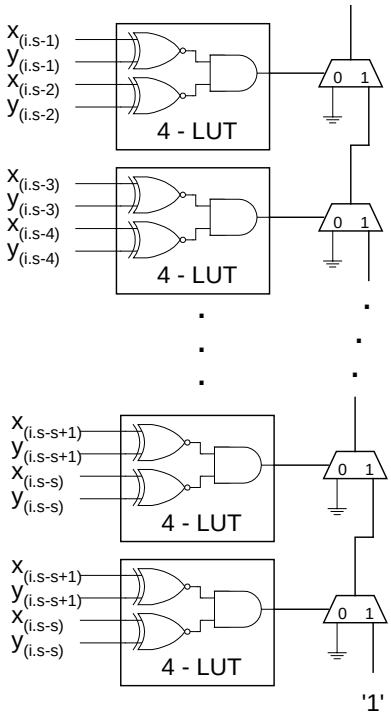


Figura 5.8. Generación de la condición de propagación del acarreo.

5.3.4 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.

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 el sintetizador XST (*Xilinx Synthesis Technology*)[Xil02] y la implementación física usando el Xilinx ISE (*Integrated System Environment*) [Xil03]. Para poder utilizar todos los recursos disponibles, el diseño instancia componentes de bajo nivel de la FPGA y usa además emplazamiento relativo (*relative placement & routing* - RPR) [Xil99] [Xil02b].

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 5.9 no utilicen recursos de rutado de proposito 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.

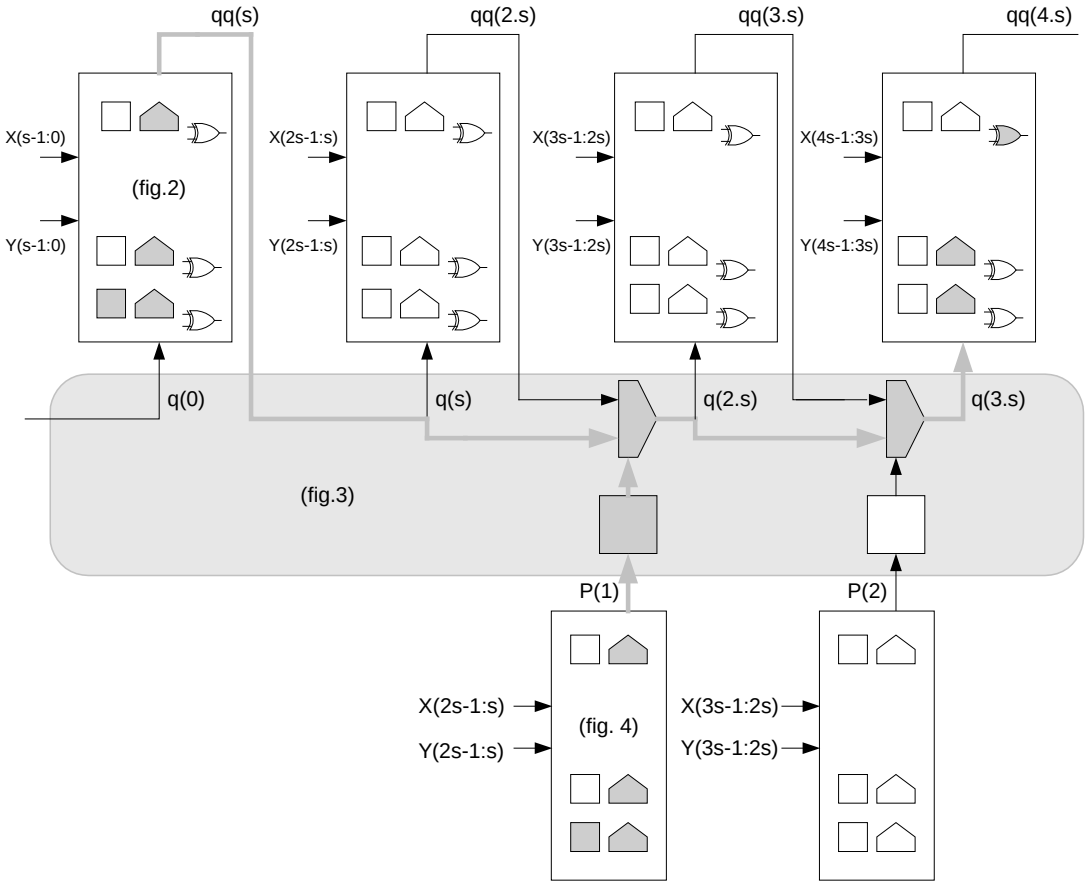


Figura 5.9. Sumador carry-skip (n = 4.s)

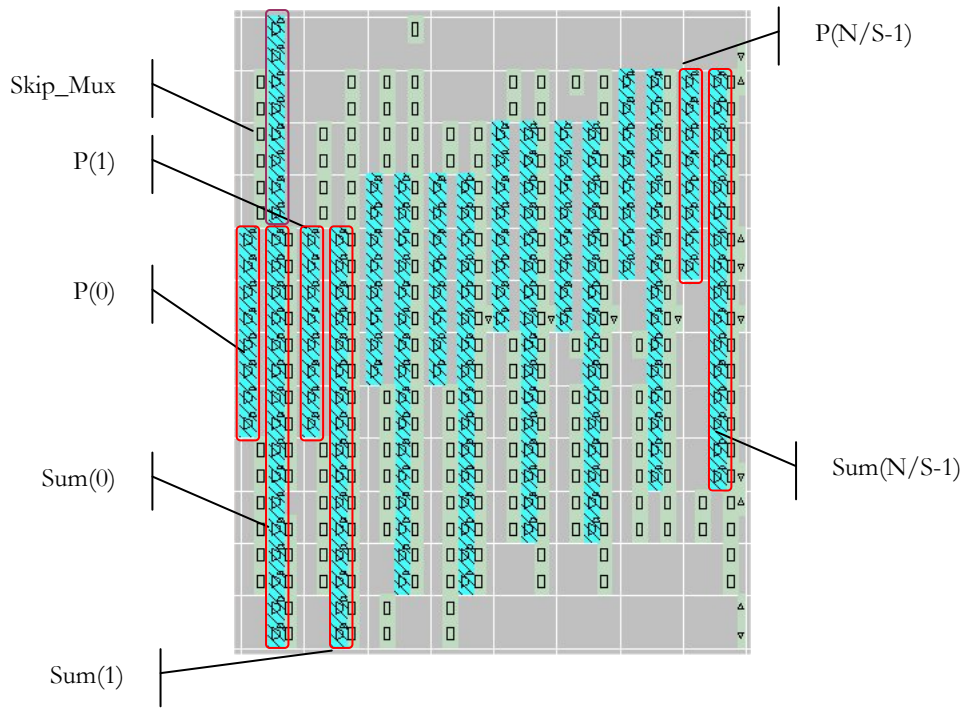


Figura 5.10. Emplazamiento relativo para un Carry-Skip con *N*=128, *S*=16.

Todos los circuitos son implementados con la estructura que se sugiere en la figura 5.11. 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.

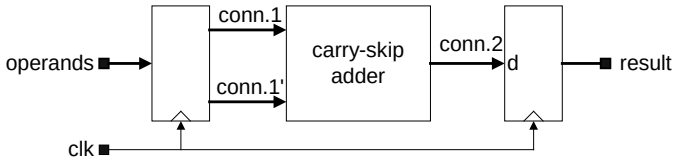


Figura 5.11. Estructura de los circuitos a medir.

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 penalidad u aumento de área producto del uso de la técnica.

n	*s*=*n*	*s*=8	*s*=16	*s*=32	freq. increase
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.

	Area in Slices				Area overhead		
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.

Resultados en Consumo

Dada la cantidad de datos de entrada-salida, resulta difícil de medir el consumo directamente. Por ello se utilizaron los sumadores dentro de un acumulador para medir su 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 5.12). Para controlar el funcionamiento se sacan como salida los 32 bits más altos y los 32 más bajos del acumulador como resultado.

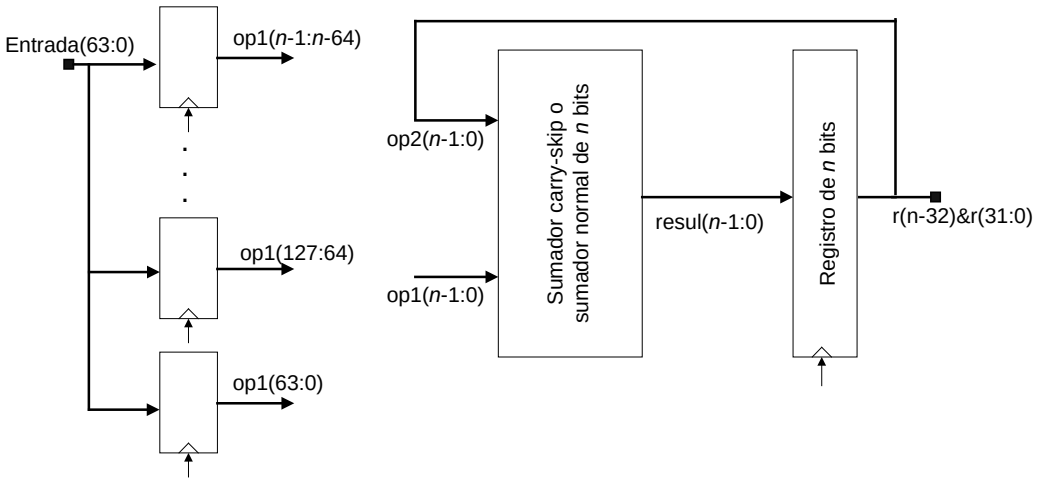


Figura 5.12. Estructura de los acumuladores utilizados para medir el consumo

Se han construido y medido acumuladores basados en sumadores normales y sumadores *carry-skip* para 256, 512 y 1024 bits. Los resultados a nivel simulación con Xpower dieron resultados 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.

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.

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.

Justificación de los Resultados en Consumo

Basados en resultados anteriores, donde los circuitos más rápidos menos consumen y que por la arquitectura del sumador *carry-skip* ha de haber una menor actividad en la cadena de acarreos es de suponer que éste último debe consumir menos. 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.

En general si se considera a un sumador compuesto por las funciones G-P (generate - propagate) donde:

g(i) = 1 iff x(i) + y(i) > B-1, en base B = 2 será: g(i) = x(i) . y(i)

p(i) = 1 iff 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 Algoritmos Sumadores

En esta sección se ha analizado el algoritmo de adición de alta velocidad *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 probado en el capítulo 4, 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 en un principio

Capítulo 6:

Conclusiones y Futuros Trabajos

Se resumen en este capítulo las principales conclusiones y aportes generados, los futuros trabajos que se desprenden de ésta publicación, así como una enumeración de las publicaciones generadas a raíz de este trabajo.

6.1 Conclusiones y Aportes

El objetivo general de este trabajo es dar soluciones en el diseño de bajo consumo en FPGAs: conocer qué circuitos poseen mejores características de consumo. No obstante, ésta es una meta demasiado ambiciosa y llevarla a cabo con éxito requeriría la evaluación de multitud de operadores, variables, arquitecturas y algoritmos. Como primera aproximación a dicho objetivo y, se ha comenzado con un estudio de las causas del consumo así como con una clasificación de las técnicas de reducción de consumo, luego se han estudiado las máquinas de estados como caso de estudio particular, siguiendo con alternativas a nivel topológico y algorítmico.

Consumo en CMOS y Clasificación de las Técnicas de Reducción de Consumo

En el capítulo 2 se brinda una introducción al diseño de bajo consumo en CMOS y su aplicación al marco tecnológico de los circuitos reprogramables. 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)ºEn 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 el voltaje 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 el voltaje 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.

Recomendaciones para la Reducción de Consumo en FSMs

Los experimentos descriptos en el capítulo 3, permiten generar la siguiente heurística para la minimización del consumo en máquinas de estados. La primer recomendación, aunque trivial, es llevar a cabo un diseño minimal de la máquina de estados. Existen innumerables programas (mayoritatiamente 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 usar 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. 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 57%

Consejos a Nivel Topológico

Durante el capítulo 4 se examinó diferentes experimentos llevados cabo sobre el consumo en FPGAs con el objeto de determinar relaciones en el consumo, y "consejos" a nivel usuario. En éste

capítulo se comprobó la relación velocidad-consumo, la conmutatividad de datos y el efecto del pipeline y el uso de registros en general. Las principales conclusiones se resumen a continuación:

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 relación entre área y consumo no es tan clara como sucede en los circuitos basados en celdas. Algunas técnicas que ganan velocidad a expensas de mayor utilización de recursos (opciones del tipo duplicar hardware para disminuir fanout) contribuyen a reducir consumo sin ser significativo el aumento de consumo por la mayor utilización de CLBs.

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 es lo ideal, en tanto que para la familia Virtex una segmentación cercana al máximo posible es lo ideal. 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.

- En la familia Virtex 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 este circuito - profundidad lógica 2) se puede reducir el consumo dinámico a un 12% del original. Finalmente se ha puesto a prueba la herramienta de estimación de consumo Xpower no obteniéndose resultados aceptables.

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%.

- 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.

- 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 usarlo, no solo por la disminución de consumo sino también por la fiabilidad de los datos.

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 y por otro se presenta un experimento cuyos resultados no fueron los esperados desde el punto de vista del consumo como es la suma por el algoritmo de *carry-skip*.

Conclusiones 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_a*) 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) y del 54% en el caso secuencial (*Montgomery* vs. Multiplicar y reducir). Además la energía consumida en las versiones secuenciales es menor debida a la disminución de los *glitches*.

Conclusiones Algoritmos Sumadores

Se ha analizado el algoritmo de adición de alta velocidad *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 probado en el capítulo 4, 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 en un principio.

Conclusiones Herramientas de Estimación de Consumo

Se ha utilizado paralelamente a las mediciones en los arreglos experimentales estimaciones del consumo con la herramienta XPOWER. Ante todo hay que destacar que es un proceso lento que implica una simulación *post-place & route* y un posterior análisis con la herramienta, lo que lo convierte en un proceso lento. Los resultados observados para circuitos secuenciales son bastante aceptables, coincidiendo las arquitecturas que más consumen en la realidad con la estimación aunque no en un porcentaje constante. Para el caso del análisis de la segmentación los resultados fueron bastante más decepcionantes, no coincidiendo las figuras del consumo medido con el estimado.

6.2 Futuros Trabajos

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 demasiado ambiciosa, se ha reducido la cantidad de experimentos a llevar a cabo. La ampliación de estos experimentos debería incluir entre otros:

- *Estudio de bloques aritméticos.* Si bien las dos operaciones fundamentales en el procesamiento de señal (suma y multiplicación) están muy estudiadas desde el punto de vista del consumo, sería útil estudiar otras primitivas de gran aplicación como son la división, la exponenciación y la raíz cuadrada. Otro aspecto interesante y de gran aplicación en el futuro es el tratamiento con operadores grandes (más de 128 bits) por sus aplicaciones en criptografía y de las operaciones en punto flotante.

- *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 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. Algunas de las técnicas a evaluar aun son las de precomputación, representación de los datos y manejo dinámico del consumo.
- *Herramientas de síntesis para bajo consumo.* Los sintetizadores tradicionalmente han optimizado en área y velocidad. La síntesis de bajo consumo tiene algunas herramientas en ASICs pero no en FPGAs, en base a los resultados obtenidos aquí se podría incorporar caracteristicas 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 herramientas que permitan estimar el consumo a un mayor nivel de abstracción.

6.3 Publicaciones Relacionadas con éste Trabajo

Se han generado publicaciones relativas a partes de este trabajo. Las publicaciones son las siguientes:

- J-P. Deschamps and G. Sutter, "FPGA Implementation of Modular Multipliers". XVII Conference on Design of Circuits and Integrated Systems (DCIS 2002) Santander, November 19-22, 2002
- G. Sutter, E. Todorovich, S. Lopez-Buedo, and E. Boemo, "Low-Power FSMs in FPGA: Encoding Alternatives", 12th Power and Timing Modeling, Optimization and Simulation Conference. (Patmos 2002), Sevilla - Spain, September 2002.
- G. Sutter, E. Todorvich, S. Lopez- Buedo, and E. Boemo, "FSM Decomposition for Low Power in FPGA", Proc of the 12th Field Programmable Logic and Application Conference (FPL 2002), Montpellier - France. September, 2002
- G. Sutter and E. Boemo "Low Power Finite state machines in FPGA: Bynary vs One hot encoding" submitted to VIII WORKSHOP IBERCHIP, Guadalajara, MÉXICO, 3-5 Abril 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.

Y artículos aún no publicados:

- G.Bioul, J.-P.Deschamps and G.Sutter "FPGA Implementation of High Speed Adders"
- G. Sutter, J-P. Deschamps and E. Boemo; "Area-Time-Power of Modular Multipliers implemented in FPGA"
- G. Sutter y E. Boemo; "Análisis de una herramienta de estimación de consumo".
- G. Sutter y E. Boemo; "Logic Depth and Low Power"

Capítulo 7:

Referencias

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

[Alc00] J. Alcalde, J. Rius and J. Figueras, "Experimental techniques to measure current, power and energy in CMOS integrated circuits", Proc. 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 VLSI, V.2,nº4,pp.426-435, Dec 1994.

[Ash91] P.Ashar, S.Devadas, and A.Newton. Optimum and heuristic algorithms for an approach to fsm decomposition. IEEE Trans.Computer-Aided Design, 10(3):296-310, March 1991.

[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.

[Ben95a] L.Benini and G. De Micheli. State Assignment for Low Power Dissipation. IEEE Journ. 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, ISLP'95 International Symposium on Low Power Design, ACM-SIGDA and IEEE-CAS, April 23-26, 1995.

[Ben96] L.Benini P.Siegel and G.De Micheli. Automatic synthesis of low-power gated-clock finite-state machines. IEEE Trans.on CAD of IC, vol.15, Issue6, June 1996, pp. 630– 643.

[Ben98] L. Benini, G. De Micheli, and F. Vermeulen. Finite-state machine partitioning for low power. In Proc. IEEE Int'l Symposium on Circuits and Systems (ISCAS '98), volume 2, pages 5-8, Monterey, California, May31-June3, 1998.

[Ben98] Luca Benini, Giovanni De Micheli, Enrico Macii, Donatella Sciuto, and Cristina Silvano; "Address Bus Encoding Techniques for System-Level Power Optimization"; Proceedings of DATE'98, February, 1998 in Paris, France

[Ben99c] Luca Benini and Giovanni De Micheli, "System-Level Power Optimization: Techniques and Tools", ISLPED99, San Diego, CA, USA 1999.

[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, April 1999, Adelaide, Australia.

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

[Boe96] E. Boemo, "Contribution to the Design of Fine-grain Pipelined VLSI Arrays", Ph.D. Thesis, 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.

[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.

[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

[Cha94b] A. Chandrakasan, R. Allmon, A. Stratakos, and R. W. Brodersen, "Design of Portable Sys-tems", Proceedings of CICC '94, San Diego, May 1994.

[Cho96] S-H.Chow, Y-C.Ho, and T.Hwang. Low Power Realization of Finite State Machines Decomposition Approach. ACM Trans on Design Aut. Elec. Systems, 315-340, July 1996.

[Dem85] 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 Trans. Computer-Aided Design 7, 12 (December), 1988.

[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

[Dun97] J.Dunoyer, F.Pétrot, L.Jacomme. Intrinsic limitations of logarithmic encodingsfor low power finite state machines. Mixed Des of VLSI Circ Conf, p 613-618, Pologne, 1997.

[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, November 2001, Porto, pp. 327 - 332.

[Flu02] Fluke Ibérica "Multímetros digitales de la serie 170" www.fluke.es

[Fpg01] FPGA Express page. Synopsis, inc.; www.synopsys.com/products/fpga/fpga_express.htm

[Gar99] A. Garcia, W. Burleson and J. Danger, "Power Consumption Model of Field Programmable Gate Arrays", Proc. FPL'99, in LNCS. Springer-Verlag, 1999.

[Geb97]C. Gebotys, Low Energy Memory and Register Allocation Using Network Flow, DAC 97, Anaheim, California. 1997

[Gei91] M.Geiger T.Müller-Wipperfürth, FSM Decomposition Revisited: Algebraic Structure Theory Applied to MCNC Benchmark FSMs. 28th ACM/IEEE DAC Conference, 1991

[Gol93] H. Goldstine, "The Computer. From Pascal to von Newman", Princeton University Press: New Jersey 1993.

[Gui69] H. Guild, "Fully Iterative Fast Array for Binary Multiplication and Addition", Electronic Letters, pp.263, Vol.5, Nº12, June 1969.

[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 The Proceedings of the European Conference on Design Automation, pages 184–191, 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 Proc. of the European Conference on Design Automation, pages 184–191, Amsterdam, Holland, Feb 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 Trans. on CAD, CAD-13(2):167{177, February 1994.

[Har60] Juris Hartmanis, Symbolic Analysis of a decomposition of information processing. Information Control, 3: 154-178, june 1960.

[Hat86] Hatamian M. 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

[Hen99] Jörg Henkel, "A Low Power Hardware/Software Partitioning Approach for Core-Based Embedded Systems". Design Automation Conference, 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] Mobile Intel Pentium III with SpeedStep.

[Ise01] ISE Xilinx 4, "ISE 4 Release Notes and Instalation Guide - 0401965", 2001, also available at www.xilinx.com

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

[Kan98] V.Kantabutra, P.Corsonello, S.Perri and M.Iachino, "Efficient, Practical Adder for FPGA", Circuit Cellar, issue 148, October 2002, pp.42 - 48

[Kor02] I.Koren, "Computer Arithmetic Algorithms", second edition, A.K.Peters, 2002

[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

[Lei85] J. Leiten, J. van Meerbegen and J. Jess, "Analisys and Reduction of Glitches in Synchronous Networks", Proc. 1995 ED&TC 1461-1464. New York: IEEE Press, 1985.

[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.

[Lin89] B. Lin and A.R. Newton. Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages. In Proc. of Internat. Conf.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.

[Lop03] Sergio López-Buedo, "Técnicas de Verificación Térmica para Arquitecturas Dinámicamente Reconfigurables"; Tesis doctoral Departamento de Ingeniería Informática; Universidad Autónoma de Madrid.

[Lop97] S. Lopez-Buedo, "Técnicas de diseño de Alta Velocidad y Bajo Consumo " Memoria Proyecto Fin de Carrera, Septiembre 1997.

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

[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, November 2001, Porto, pp. 100 - 105.

[Mar00] M. Martínez, M. J. Avedillo, J. M. Quintana, M. Koegst, ST. Rulke, and H. Susse: Low Power State Assignment Algorithm, DCIS'00 conf , pp. 181-187, 2000.

[Mar95] R. Marculescu, D. Marculescu and M. Pedram, "Efficient Power Estimation for Highly Correlated Input Streams", Proc. 32 DAC Conf. 1995.

[Mee95] J. Leijten, J. Meerbengen, and J. Jess, "Analysis of Reduction of Glitches in Synchronous Networks", Proc. EDAC95, IEEE Press.

[Men02] Mentor Graphics, "LeonardoSpectrum Bookcase v2002a", 2002. www.mentor.com

[Men96] A. Menezes, P. van Oorschot and S. Vanstone, "A Handbook of Applied Cryptography", CRC Press, 1996.

[Men99] L. Mengíbar, M. García, D. Martín, and L. Entrena, "Experiments in FPGA Characterization for Low-power Design", Proc. DCIS'99, Palma de Mallorca, 1999.

[Men99] L. Mengíbar, M. García, D. Martín, and L. Entrena, "Experiments in FPGA Characterization for Low-power Design", Proc. DCIS'99, Palma de Mallorca, 1999.

[Men99] L. Mengíbar, M. García, D. Martín, and L. Entrena, Experiments in FPGA Characterization for Low-power Design, Proc. DCIS'99, Palma de Mallorca, 1999.

[Mon85] P.Montgomery, "Modular multiplication without trial division", Mathematics of Computation, vol.44, pp. 519 - 521, April 1895.

[Mon98] J. Monteiro, A. Oliviera, Finite State Machine Decomposition for Low Power, Proceedings 35th Design Automation Conference , San Franscisco, 1998, pp. 758-763.

[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 Trans. on VLSI Systems, Vol.2, nº4, pp.446-455. Diciembre 1994.

[Nec98] NEC Electronics Inc, "Power Management Modes In the VR4100 Family of 64-Bit MIPS RISC Microprocessors", Application Note, July 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 CAD. December 1988.

[Not99] Winfried Nöth and Reiner Kolla. Spanning Tree Based State Encoding for Low Power Dissipation". In Proc of Date99, pp 168-174, Munich, Germany, March 1999.

[Obe01] S.F.Oberman and M.Flynn, "Advanced Computer Arithmetic Design", Wiley, 2001

[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

[Par00] B.Parhami, "Computer Arithmetic, Algorithms and Hardware Designs", Oxford University Press, 2000

[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.

[Ped96] M. Pedram, "Power Minimization in IC Design: Principles and Applications", ACM Trans. On Design Automation of Electronic Systems", vol.1, nº1, pp.3-56, January 1996.

[Ped96] M. Pedram, "Power Minimization in IC Design: Principles and Applications", ACM Trans. On Design Automation of Electronic Systems", vol.1, nº1, pp.3-56, January 1996.

[Ped96] Massoud Pedram, "Power Minimization in IC Design: Principles and Applications", ACM Trans. On Design Automation of Electronic Systems, Vol 1, No 1, Jan 1996

[Ped96] Massoud Pedram, "Tutorial and Survey Paper - Power Minimization in IC Design: Principles and Applications" ACM Trans. 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. 1997, pp 73-96

[Pre00] Programmable Electronics Performance Corporation (PREP) Benchmarks (Programmable Electronics Performance Company), see: http://www.prep.org.

[Pre00] Programmable Electronics Performance Corporation (PREP) Benchmarks (Programmable Electronics Performance Company), see: http://www.prep.org.

[Rab96] J.M.Rabaey, "Digital Integrated Circuits", Prentice Hall, 1996

[Rab96] Jan M. Rabaey, "Exploring the Power Dimension Proc. Of Custom Integrated Circuit Conference", May 1996, pp 215-220

[Rag96] Anand Raghunathan, Sujit Dey and Niraj K. Jha, "Glitch Analysis and Reduction in Register Transfer Level Power Optimization", 33rd Design Automation Conference ACM 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.

[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.

[San90] A. San Giovanni-Vincentelli, T. Villa. NOVA: State Asignament of Finite State Machines for Optimal Two Level Logic Implementation. IEEE Transaction on 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. Tech. Report Mem. No. UCB/ ERL M92/41, Univ. of California, Berkeley, 1992.

[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.

[She00] R.Shelar, H.Narayanan, M.Desai, Orthogonal Partitioning and Gated Clock Architecture for Low Power Realization of FSMs, IEEE Int ASIC/SOC conf, 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", Proc. ICCAD-92 Conf, pp.402-407. IEEE Press, 1992.

[She99] R.Shelar, M.P. Desai, H.Narayanan, "Decomposition of Finite State Machines for Area, Delay Minimization", IEEE ICCD99 Austin, 10-13 Oct. 99, pp. 620-625.

[Sia03] SIA Semiconductor Industry Association, http://www.semichips.org/

[Sia97] SIA Semiconductor Industry Association, "The National Technologies Roadmap for semiconductors: Technologies Needs", 1997.

[Sta02] StateCAD 5.03, a graphical entry tool for FSM. distributed with Xilinx ISE foundations tolls 4.x, www.xilinx.com

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

[Sta95a] M. Stan, W. Burlesson, "Bus-Invert Coding for Low-Power I/O", IEEE Trans. on VLSI Systems, vol.3, nº1, pp.49-58. Marzo 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. Mar. 1995.

[Sta96] Mircea R. Stan; Wayne P. Burleson, "Two-Dimensional Codes for Low Power", ISLPED 1996 Monterey CA USA.

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

[Sut02] G. Sutter and E. Boemo Low Power Finite state machines in FPGA: Bynary vs One hot encoding. VIII Workshop Iberchip, Guadalajara, Mexico, April 2002.

[Syn01] Synopsis Inc, "FPGA Express 3.6.1 User Guide", Agosto 2001. www.synopsis.com/fpga/

[Syn02] Synpicity Inc; "Synplify Pro 7.1 Online Documentation", April 2002. Available at www.synplicity.com

[Syn98] Synplicity, Inc. Synplify User Guide Release 5.0, Agosto 1998. www.synplicity.com

[Syn99] Synopsis, FPGA Express User Guide version 2.3, 1999. FPGA Express home page; http://www.synopsys.com/products/ fpga/fpga_express.htm

[Tek00] Tektronix TLA 704 Logic Analyzer user guide. www.tektronix.com

[Tek01] Tektronix inc; "TLA 704 Logic Analyzer user Guide". www.tektronix.com

[Tek02] Tektronix inc; "TLA7PG2 Pattern Generator Module"; available at www.tektronix.com

[Tek02] Tektronix inc., "TLA 700 Series Logic Analyzer User Manual", 2002, available at http://www.tektronix.com.

[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" 35 th Design Automation Conference 06/98 San Francisco, CA USA.

[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.

[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 (DCIS2000), 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.

[Tsu94a] C-Y Tsui, M.Pedram, A. Despain, Exact and Approximate Methods for Calculating Signal and Transition Probabilities in FSMs, 31st Design Autom. Conf., pp. 18-23, 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

[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.

[Vil90] T.Villa, A.Sangiovanni-Vincentelli, "NOVA: State assignment for finite state machines for optimal two-level logic implementation", IEEE TCAD, Vol.9-9, pp.905, Sept. 1990.

[Wall64] C. Wallace, "A Suggestion for a Fast Multiplier". IEEE Trans. on Electronic Computers, pp.14-17, February 1964.

[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

[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.

[Xch99] Xchecker Xilinx download cable. "Hardware User Guide: XChecker Cable" http://toolbox.xilinx.com/docsan/3_1i/

[Xil00] Xilinx Inc, "Software Manuals and documentation for Foundation Series 3.1i." http://toolbox.xilinx.com/docsan/3_1i/, 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, information available at www.xilinx.com/support/ library.htm

[Xil01] Xilinx inc; "Virtex™ 2.5 V Field Programmable Gate Arrays Product Specification" DS003-1 (v2.5) April 2, 2001

[Xil01] Xilinx, inc. "Spartan-II 2.5V FPGA Family: Functional Description"; March 2001.

[Xil02] Xilinx, inc. "Xilinx Synthesis Technology (XST) User Guide", 2002, available at www.xilinx.com

[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.

[Xil02b] Xilinx, inc "Constraints Guide – ISE5.1" section 2-14, 2-15 Relative Location (RLOC) and Relationally Placed Macros (RPMs)", 2002.

[Xil03] Xilinx, inc. "Carry Logic in Virtex and Spartan-II", The Programmable Logic Data Book available on the Xilinx web site, http://support.xilinx.com.

[Xil03a] Xilinx Inc; "Xilinx Parallel Cable IV Advance Product Specification"; DS097 (v1.4) March 8, 2002 available at http://toolbox.xilinx.com/docsan/

[Xil03b] Xilinx, inc. "ISE 5.1 documentation", 2003, available at support.xilinx.com

[Xil95] Xilinx Inc, "Power Considerations", in Technical Conference and Seminar Series, 1995.

[Xil99] Xilinx inc "Xilinx Prototype Platforms User Guide for Virtex and Virtex-E Series FPGAs" DS020 (v1.1) December, 1999

[Xil99] Mala Sathyanarayan, "XCELL Journal" Issue 32, Second Quarter 1999.

[Xin98] S.Xing and W.W.H.Yu, "FPGA Adders: Performance Evaluation and Optimal Design", IEEE Design & Test of Computers, January - March 1998, pp. 24 - 29

[Xpo02] Xpower, "Xpower getting started", available at support.xilinx.com.

[Yan91] Saeyang Yang, Logic Synthesis and Optimization Benchmarks User Guide Version 3.0 Technical report, MCNC, Research Triangle Park, North Carolina, January 1991.

Apéndice A:

Placa de prueba XC4000

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-Boedo [Lop9X]. 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 las placas 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.

Haciendo una breve descripción, las placas 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 placa esta realizada en circuitos impreso con tecnología de dos caras sin taladros metalizados. La cara superior esta usada prácticamente en su totalidad como plano de masa. La placa incluyen conectores duplicados de alimentación tipo banana 4 mm, así como un BNC para la señal de reloj. Uno de los conectores de alimentación se utiliza únicamente por la FPGA bajo prueba, facilitándose así la medición del consumo, mientras que el otro sirve para todos los demás recursos de la placa de prueba.

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 download de la XC4000 y la asignación de botones y conmutadores de la XC3000.

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: Otros pines importantes

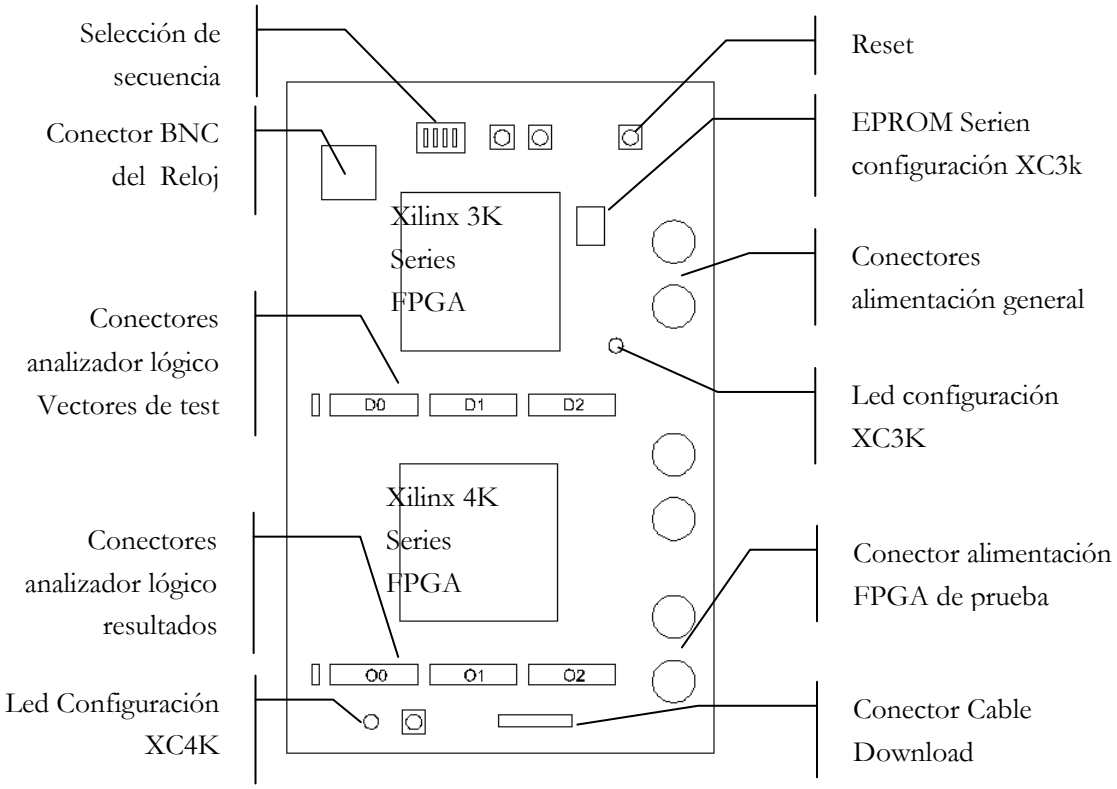


Figura A.1. Esquema de la placa de prueba de la familia XC4000.

A.3 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 el la FPGA a medir se colocan los elemento de medición (voltímetro y amperímetro) Figura A.2. Se conectan además el generador de reloj (Metrix GX245), el analizador lógico y el cable de download conectado al puerto serie del PC (Xchecker).

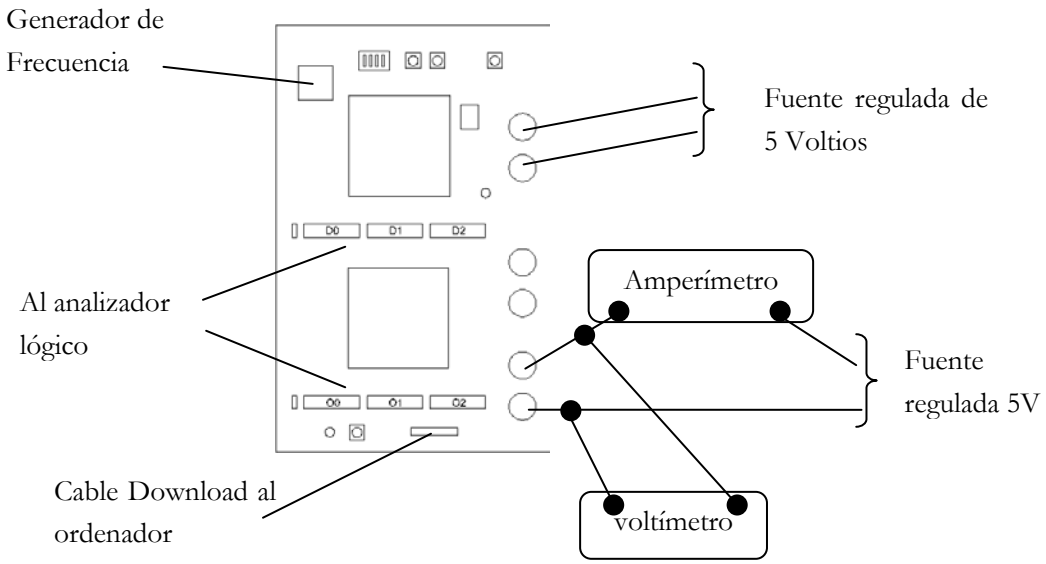


Figura A.2. Esquema de conexión del arreglo experimental

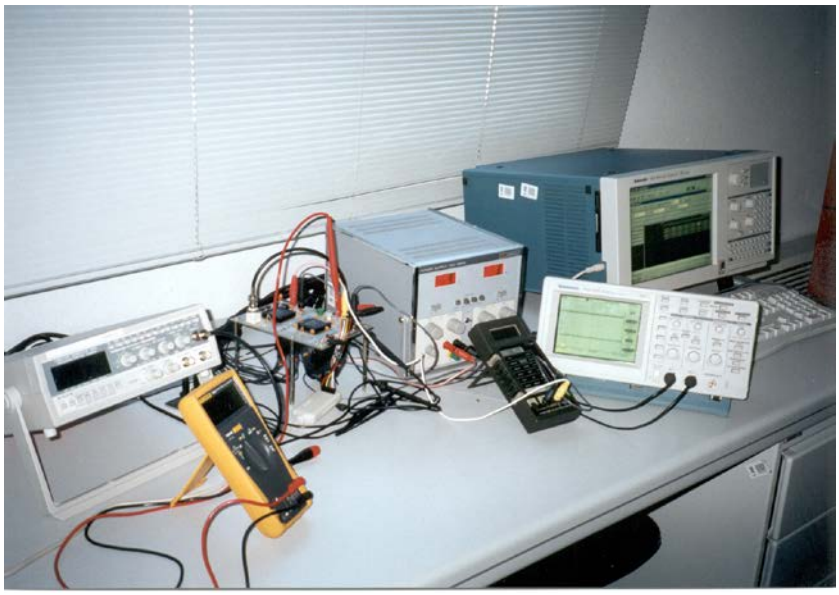


Figura A.3. Fotografía del arreglo experimental

A.4. Método de Medición de Consumo

La técnica usada 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. 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. En la tabla A.3 se resumen estos conceptos.

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.3: Componentes del consumo y forma de medición

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.

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 $I \times V$ por tanto tiene una precisión del orden del 1%.

A.5. 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.

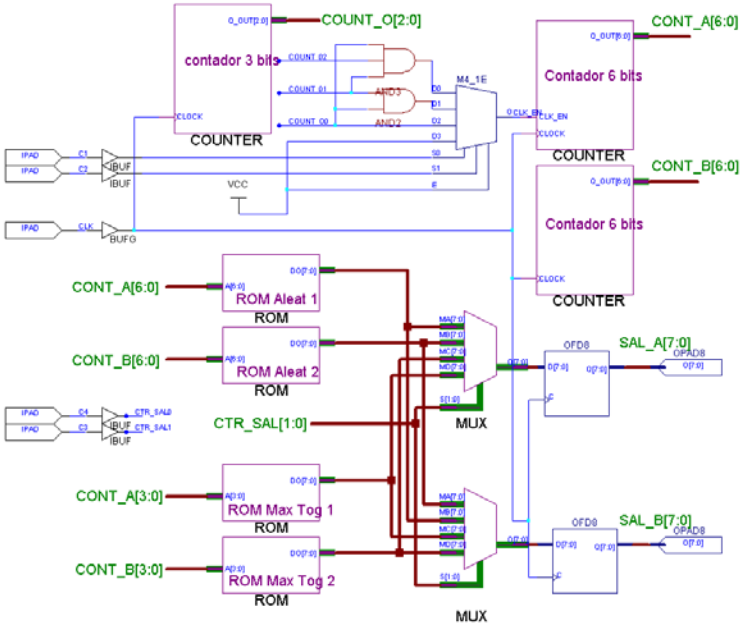


Figura A.4. Esquema del generador de patrones.

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.

Apéndice B:

Placa de prueba AFX

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].

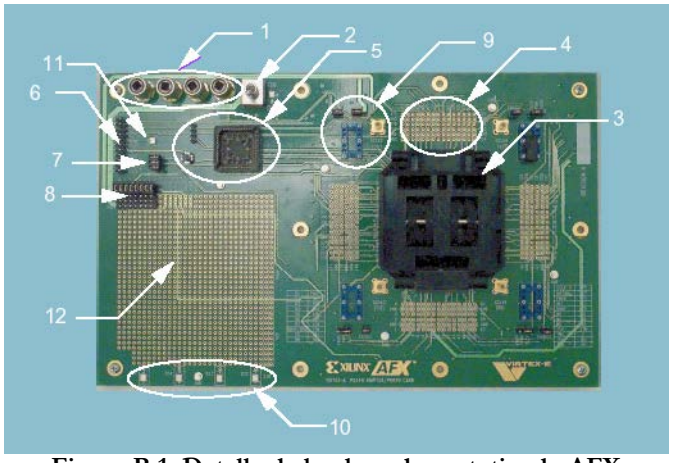


Figura B.1. Detalle de la placa de prototipado AFX

B.2 Características de la Placas de Prueba AFX

En la figura B.1 se puede ver una fotografía extraída de las notas técnicas de Xilinx. Básicamente, la tarjeta tiene unas 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 o bien cualquier cable de los provistos por Xilinx (por ejemplo paralell cable III [xil03a]) o bien en este caso se utiliza una interfaz 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 D.2

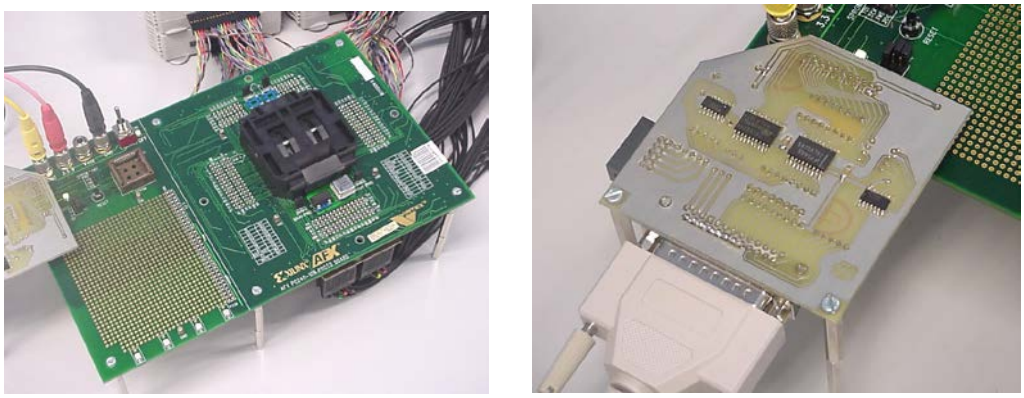


Figura B.2. Fotografía de la tarjeta AFX y de la interfaz con el puerto paralelo

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, 888,439 puertas equivalentes), fundamentalmente esta última FPGA se ha utilizado para la mayor parte de las medidas.

B.3 Conexión de la Placa de Pruebas

Las FPGAs de la familia Virtex se alimentan con 2,5 voltios el *core* del circuito en tanto que la periferia se alimenta con 3,3 voltios. Una fuente de alimentación doble alimenta por separado cada parte de la placa con las tensiones correspondientes. En el arreglo experimenta 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 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].

A la placa se han conectado 64 bits de entrada (procedente del generador de vectores de test) detalles de la asignación en la tabla D.1. 64 patas de salida de la FPGA se conectaron al analizador lógico, cuya asignación de patas se puede ver en la tabla D.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

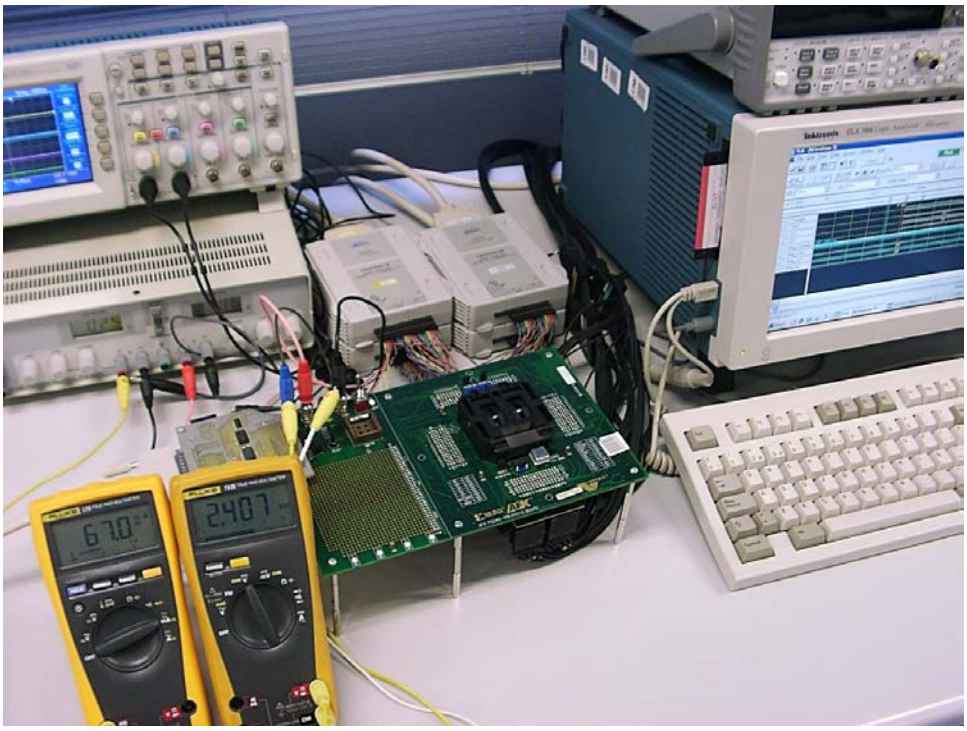


Figura B.3. Fotografía del arreglo experimental

B.4 Metodología de Medición de Consumo

La metodología de medición es similar a la expresada 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*) será alimentado por 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 D.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, pues 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 2,5V. No obstante se controla la tensión de 3,3 V para evitar errores en las medidas. En el rango utilizado el voltímetro posee un error 0,15% de la lectura, en tanto que como amperímetro el 1%.

Apéndice C:

Traductor Kiss2VHDL

C.1. Introducción

Se describe aquí la herramienta software construida para la realización de los experimentos de consumo de diferentes codificaciones en máquinas de estados finitos. Como prólogo se describen el formato de especificación de máquinas de estado KISS2 [Sen92] y los grupos de bancos de pruebas (benchmarks) MNCM [Lis88] y PREP [Pre00].

C.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 y se ha extendido su utilización en múltiples herramientas de asignación y minimización de estados.

En Kiss 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-combinacion 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 indica que puede ser indiferentemente 0 o 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

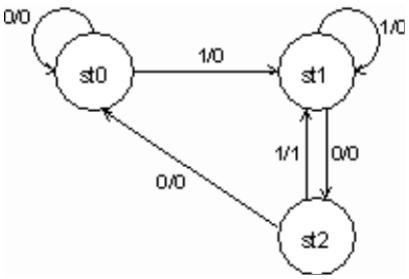
.i [d] Especifica la cantidad de variables de entrada.
.o [d] Especifica la cantidad de variables de salida
.p [d] Especifica la cantidad de reglas (opcional)
.s [d] Especifica la cantidad de estados (opcional)
.r [s] Especifica el estado de reset (opcional)

Luego se colocan las reglas en la tabla de estados, donde **current_state** y **next_state** son nombres simbólico representando 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

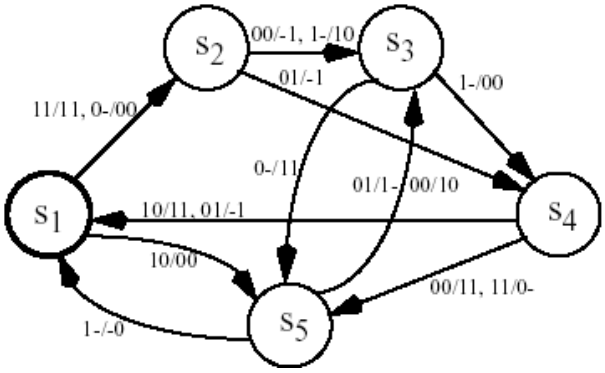
Un primer ejemplo: Un diagrama de transición de estados (STG) de una máquina de estados con un bit ded entrada y otro de salida, tres estados 6 reglas. Observese que la cantidad de estados y reglas son opcionales y no se han agregado.




```
.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 C.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
.0 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 C.2. Ejemplo grafo de transición de estados y su especificación en Kiss2

C.3 Bancos de pruebas (Benchmarks)

Para medir y comparar consumo en máquinas de estado se han utilizados los bancos de prueba de la 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

(**M**icroelectronics **C**enter of **N**orth **C**arolina) e incluye los conjuntos de bancos de prueba ISCAS'85 e ISCAS'89. El primer reporte de gran difusión (versión 2.0) data de 1988 [lis88] y ha sido ampliado en la versión 3.0 [Yan91], siendo 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 o Logica de dos niveles en Berkley PLA format (Como KISS pero sin estados) o SLIF. Se puede conseguir en la dirección de 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 www.prep.org.

C.3 Traductor Kiss2VHDL

El programa Kiss2VHDL codifica los estados de la máquina de estados 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 bufferss 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 parser para transformar el fichero KISS2 leído en una representación de diagrama (o grafo) de transición de estados, rutinas de asignación de estados 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;
```

Apéndice D:

Particionador de máquinas de estado: *Part_FSM*

D.1. Introducción

El programa *Part_FSM* es un programa C++ para la partición de máquinas de estados. Como entrada acepta máquina de estados descriptas en formato KISS2 [Sen92] (apéndice C), generando como resultado la partición de la máquina de estados según lo expresado en la sección 3.4. 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 3.4.4) u ortogonal (sección 3.4.3) además del tipo de método de bloqueo necesario.

D.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.

D.3 Cálculo de Probabilidad Estática

La probabilidad estática se calcula tal lo expresado en la sección 3.4.5. 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 sencillo como dividir la proporción de combinaciones que generan la transición respecto del total. Se puede ver el la figura 3.15.b las probabilidades calculadas, en tanto que en el ejemplo de la figura D.3 todas la probabilidades de transición son de 0,5.

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 muestran las figuras 3.15.c y 3.16.b se procede a invocar al algoritmo de partición.

```
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 D.1. Declaración de la parte privada de la clases *state* y *transition*.

D.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 $\Pi_B = \{S_{b1}, S_{b2}, \ldots S_{bn}\}$ 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. 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 D.2 se puede observar el código para la generación de las particiones.

```
// código del algoritmo de partición
void partitioner (state* STG, partition *P, float &minDistance, 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 estadoss
      //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) < minDistance){
        P->put(n_state,0);
        partioner(STG,P,minDistance,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) < minDistance){
              P->put(n_state,1);
              partitioner(STG,P,minDistance,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 < minDistance) { //si es la major sssolución se guarda
           minDistance = pd;
           P->print_partition();
           if (minP != NULL) delete (minP);
            minP = P->copy();
        }
      }
}
```

Figura D.2. Código para la partición de máquinas de estados.

D.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 D.3. En la figura D.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 D.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.

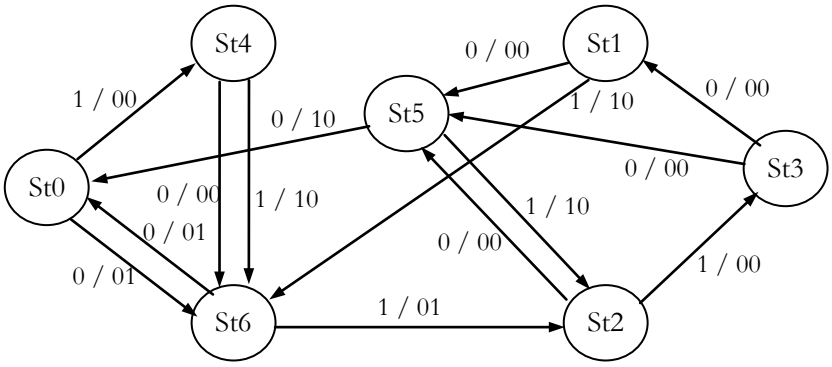


Figura D.3. Diagrama de transición de estados de la máquina de estados DK27.

```
.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 D.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 D.5. Información de las particiones generadas.

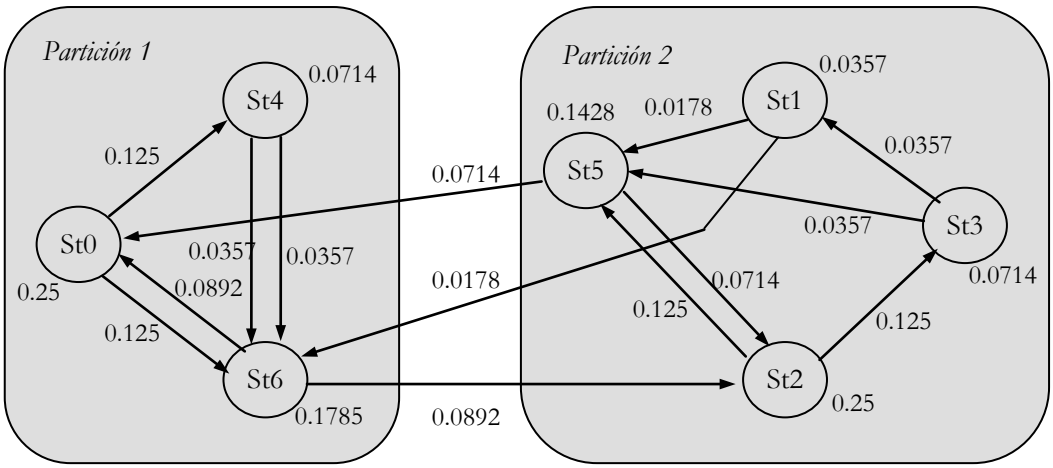


Figura D.6. Partición generada para el circuito *dk27*.

Por último, se muestra el código VHDL para el circuito dk27 con la arquitectura I, latches para bloquear los datos y buffer 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 la parte combinacional de las submáquinas de estado; y 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;
```