Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Industriales

Departamento de Automática, Ingeniería Electrónica e Informática Industrial

Máster Universitario en Electrónica Industrial

CARACTERIZACIÓN Y MEDIDA DE CONSUMO DE ENERGÍA EN SISTEMAS BASADOS EN FPGAs DE XILINX

Autora: Laura R. Gorostiaga Vázquez

Tutora: Teresa Riesgo Alcaide

Trabajo Fin de Máster

Agradecimientos:

Esta dedicatoria va para todas las personas sin cuyo aliento la culminación de este libro no hubiese sido posible.

A todos [1] ¡Gracias!

1 Solicitar clave de cifrado *al autor [Algoritmo (AES), función (CRC32), modo (ECB) y vector de iniciación automático].*

Indice General

Indice de Figuras

Indice de Tablas

Resumen

En este trabajo de fin de máster se ha estudiado el consumo de potencia de circuitos digitales para la placa BASYS2 y la eficacia y precisión de la herramienta de estimación de consumo Xpower Analyzer de XILINX. Los resultados de la estimación del consumo de potencia de circuitos digitales es contrastada con medidas físicas realizadas en laboratorio, dicha estimación de consumo de energía como ya dijimos se realiza con la herramienta XPA ISE14.7 de XILINX.

El objetivo principal del proyecto es analizar un circuito digital determinado variando la longitud de los vectores de entrada y salida del mismo, estudiando a la par también los parámetros más importantes a la hora de reducir el consumo de energía de estos sistemas digitales basados en FPGAs.

Son cuatro los términos matemáticos involucrados en la medida del consumo de potencia en circuitos digitales, pero el más importante a la hora de reducir el consumo es el de la potencia dinámica, que es la potencia relacionada a las conmutaciones en los nodos del circuito, tanto internamente como en las entradas y salidas.

Para la medida física del consumo de potencia en la placa BASYS2, existe una variedad de sistemas de adquisición de datos en este proyecto optamos por el más sencillo de todos, el cual no añadía consumo de más que el que solo queríamos medir que es el de la placa en cuestión, este sistema consiste en una resistencia de Shunt en serie con la fuente y la placa BASYS2, lo que medimos es la tensión e indirectamente con el valor de la resistencia calculamos la intensidad, teniendo estos datos calculamos fácilmente el consumo de potencia.

Para el estudio contamos también con un generador de números aleatorios LFSR, el cual también se detalla en los anexos de este libro específicamente el Apéndice A, se ha utilizado solo un bloque generador de números aleatorios para cada experimento con el circuito, a este código Generador de números aleatorios integramos un multiplicador genérico cuyo código esta también detallado en el Apéndice B, a la hora de variar las entradas y salidas y obtener así una medida de la lógica en el sistema.

En la placa BASYS2 realizamos varias pruebas, con el circuito multiplicador genérico, variamos las entradas salidas del mismo de la siguiente forma, para una entrada de 2 bits, la salida correspondiente es de 4 bits, para una entrada de 3 bits la salida correspondiente seria de 6 bits, y asi sucesivamente hasta probar entradas de N bits con salida de 2*Nbits.

Como comentamos al principio hemos utilizado las herramientas del ISE14.7 como ISIM para realizar las simulaciones, y la herramienta XPower Analyzer para estimar la energía consumida por los nodos de cada prueba.

Se han realizado en total 44 diferentes pruebas sobre la BASYS2, y un total de 44 simulaciones, y estimaciones de potencia con el XPower Analyzer de XILINX, utilizamos los circuitos internos de gestión de reloj llamados DCM del inglés Digital Clock managers, cabe mencionar que si generamos una frecuencia menor a 50MHz*(frecuencia de trabajo por defecto de la BASYS2)* con el DCM la potencia no será menor por ser la frecuencia menor precisamente si no que aumentara el consumo debido a la utilización de estos bloques DCM, utilizamos también estos bloques para realizar las pruebas a reloj parado que se detallan en el capítulo de metodología de este libro.

La conclusión a la que llegamos y que se puede ver a través de las pruebas realizadas es que al aumentar la longitud de los vectores de entrada/salida del circuito aumenta la potencia consumida, ya que se utilizan más recursos de la placa, lo que es muy intuitivo, y nos lleva a concluir que el efecto de añadir más lógica a un sistema aumenta el consumo de potencia. Lo anterior implica que a la hora de reducir el consumo de potencia de un sistema, es necesario reducir al máximo la lógica del mismo, que es una técnica de bajo consumo para circuitos digitales tenida en cuenta por los programadores.

CAPITULO 1: INTRODUCCIÓN

1.1 Antecedentes y Justificación

La eficiencia energética en Tecnología de circuitos integrados (ICTs) está teniendo mucha influencia, se está convirtiendo en sector estratégico dentro de la economía mundial. Este impacto en el desarrollo socio-cultural tiene gran influencia y se puede prever que continuará creciendo en el futuro.

El estado del arte de la Tecnología de circuitos integrados (ICTs) en un futuro cercano estará basada en dispositivos digitales, cuyo funcionamiento está actualmente dominado por perdidas de potencia que producen calor. Es este un grave problema, por el siguiente número de razones:

- ***Razones Socio-económicas***: primeramente por razones socio-económicas, lo que implica el funcionamiento de los dispositivos basados en Tecnología de circuitos integrados (ICTs) se realice mediante reducidos montos de energía, en un futuro cercano considerado un objetivo de muy alta relevancia en el ámbito económico. De acuerdo con un estudio del congreso SMART 2020 [9] hoy en día la porción de consumo de energía de los ICTs dentro del consumo de energía mundial está en el orden de 2-5%. Dado que el consumo de tecnología ICT continuara creciendo y todo el consumo de energía se espera se reduzca debido a la contribución de la propia tecnología ICT entre otras medidas, se espera que la porción de ICT dentro del consumo mundial de energía pueda crecer en el futuro. Las emisiones de Dióxido de carbono, producto de la utilización de tecnología ICT, estará como resultado en un futuro cercano incrementándose. Desde este hecho, esto se está transformando en más y más importante, de forma a considerar y mejorar la eficiencia energética de la tecnología ICT. A corto plazo esta será una obvia y practica solución, para explotar el potencial de tecnologías ya existentes o estén actualmente en desarrollo. A largo plazo se necesitaran nuevas y rompedoras ideas.

- ***Razones tecnológicas***: En los últimos cuarenta años la industria de semiconductores, fue manejada por la habilidad de escalar por debajo el tamaño de los dispositivos CMOS(interruptores transistores de efecto de campo), bloques construidos en dispositivos de computo actuales, y por incrementar la densidad de capacidad de computo por encima del punto donde la potencia se disipa en forma de calor y que durante la computación se convierte en un serio problema. De hecho, la densidad de potencia de los chips de hoy en día, es varias decenas de W/cm² lo cual implica una superficie más caliente que los (hot plates) de alta potencia. La disipación de potencia versus la velocidad de conmutación de dispositivos ha sido caracterizada desde 1970 por medio de reglas de escalamiento donde las capacidades de micro-fabricación a través del reemplazo de transistores bipolares por tecnología CMOS, permitió la continuación del incremento exponencial en dirección a la capacidad de procesamiento de información. Esto ha sido conocido como leyes de Moore, sin embargo desde 2004 un consorcio Americano de compañías de la industria de semiconductores[11], ha lanzado el gran desafío de direccionar el límite fundamental en la física de interruptores. Estos límites están principalmente representados por la menor energía y el menor tiempo, requeridos para la operación en el interruptor. Con la presente estimación de energía mínima requerida para la corriente de un dispositivo basado en tecnología CMOS. El resultado de densidad de potencia para estos interruptores, a máxima densidad de encapsulamiento será del orden de 1MW/cm². Esto es comparable a la densidad de potencia en una boquilla de cohete y es varios órdenes de magnitud mayor que lo que es actualmente tecnológicamente manejable. Por lo tanto la cantidad de energía disipada a través de calor es actualmente el principal obstáculo para continuar el aumento en el rendimiento en términos de computación.

- **Razones científicas:** En la actualidad el principal esfuerzo para superar las limitaciones tecnológicas está dirigido a enfriar los chips mediante la eliminación del calor producido durante el cálculo. Con atención específica al transporte de carga por un lado y por otro lado en la reducción de los niveles de operación de tensión hasta el punto de no comprometer la tasa de error debido a las fluctuaciones de tensión. Tal estrategia ha producido algunos resultados interesantes sin

embargo, está llegando a su límite claramente debido al requisito de energía de entrada insostenible. Hay intentos de enfocar el problema desde un punto de vista más fundamental, abordando los mecanismos básicos detrás de la producción de calor y la función de las fluctuaciones derivadas mediante la reducción de las tensiones de umbral.

1.1.1 Diseño de bajo consumo

Para hablar de las principales técnicas de reducción de consumo en los diferentes niveles de abstracción se debe empezar por hablar del consumo en circuitos CMOS.

En la tecnología CMOS, el consumo de las puertas lógicas se rige por la ecuación a continuación:

$$P_{TOTAL} = P_{estática} + P_{fugas} + P_{dinámica} + P_{cortocircuito} \quad \text{[Ecuación 1.1]}$$

- **Potencia estática**: La potencia estática es debida a la existencia de un camino de menor impedancia entre VDD y GND en condiciones estáticas. Se calcula con la expresión:

$$P_{estática} = I_{estática} \times V_{DD} \quad \text{[Ecuación 1.2]}$$

- **Potencia por corriente de fugas**: como su nombre indica es provocada por las corrientes de fuga en los transistores, tiene expresión parecida a la de la potencia estática:

$$P_{fugas} = I_{fugas} \times V_{DD} \quad \text{[Ecuación 1.3]}$$

Esta corriente de fugas también se compone de la suma de las corrientes por las uniones pn inversamente polarizadas y por las corrientes sub-umbral de los transistores.

- **Potencia de cortocircuito**: Es debida al tiempo que tardan en conmutar los transistores, cuando pasan de estar activos a estar en estado de corte y viceversa hay un tiempo en el cual los dos están conduciendo y se provoca una corriente de cortocircuito.

- **Potencia dinámica**: es debida a la actividad del circuito y representa la mayor parte de consumo de potencia en sistemas digitales actualmente, y es la que más se estudia a la hora de reducir el consumo, la expresión aproximada es la siguiente:

$$P_{dinámica} = \propto C V_{DD}^2 f_{CLK} \quad \text{[Ecuación 1.4]}$$

La expresión más precisa para la potencia dinámica es:

$$P_{dinámica} = \frac{1}{2} C_L \cdot V_{dd}^2 \cdot E_{(sw)} \cdot f_{clk} \quad \text{[Ecuación 1.5]}$$

La reducción del consumo se puede obtener reduciendo la capacidad CL, la actividad del circuito $E_{(sw)}$, o la tensión de alimentación Vdd, pero las mayores reducciones de energía se logran con optimizaciones a nivel de arquitectura, algoritmo o sistema que es lo que tratara el capítulo 2 de este libro.

1.1.2 Sistemas Digitales

Son dispositivos que trabajan con señales discretas, que solo pueden tomar ciertos valores, para controlar señales digitales, se rigen por el álgebra de Boole y el código binario.

Se dividen en combinacionales y secuenciales

Las puertas lógicas que se utilizan para construir circuitos digitales a su vez están hechas básicamente de transistores.

1.1.3 FPGAs

Nombre que viene del Ingles Field Programmable Gate Array, es un dispositivo lógico programable, es decir un chip cuyas puertas lógicas a nivel físico podemos programar.

Los dispositivos FPGAS se utilizan en multitud de campos, que van desde las industrias de fabricación mecanizada hasta la industria aeroespacial, pasando por la industria militar.

Este tipo de dispositivo está a medio camino entre los circuitos de propósito específico (ASICs) y los procesadores de propósito general en prestaciones, posibilidad de optimización, consumo de potencia, etc. Su principal ventaja frente a los diseños específicos es que son reprogramables, por lo que proporcionan una gran flexibilidad de diseño, que los costes de desarrollo y adquisición son muy económicos, que el tiempo de desarrollo es mucho menor y que existe la posibilidad de realizar reconfiguraciones dinámicas (durante el funcionamiento del dispositivo) del diseño.

La tarea del programador es definir la función lógica que realizará cada uno de los bloques lógicos de la FPGA e interconectarlos. Para ello debe utilizar alguno de los entornos de desarrollo especializados en el diseño de sistemas sobre FPGA. Si se trata de un diseño sencillo puede hacerse con un esquemático, si no, habrá que utilizar un lenguaje de programación especial HDL (Hardware Description Language) como VHDL o Verilog.

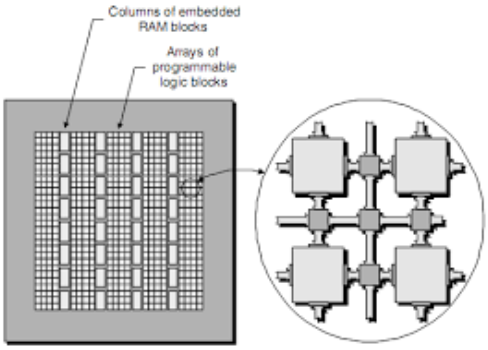

Figura 1.1 Esquema de un dispositivo FPGA

Se pueden diseñar circuitos muy básicos a sistemas muy complejos como microprocesadores.
Estos dispositivos FPGAs vienen montados en tarjetas de desarrollo o Board Kits, que tienen periféricos como LEDs, botones, interruptores y osciladores puesto que como las placas están orientadas a la fase de desarrollo de circuitos, estas entradas y salidas facilitan al programador el trabajo de variar parámetros de entrada y visualizar la salida mediante LEDs Displays, y hasta un puerto VGA.
El lenguaje que utilizamos en este proyecto es el VHDL las ventajas que tiene son las siguientes:

- Portabilidad
- Permite que el comportamiento de un diseño pueda ser descrito y verificado antes que las herramientas de síntesis lo traduzcan a hardware.
- Permite la descripción de lenguaje concurrente y no secuencial como C.

Del tema de FPGAs hablamos más en profundidad en el capítulo 2 de este libro a continuación.

Capítulo 2: Definición e Historia de las FPGAs

2.1 Introducción

FPGA viene de las siglas en inglés (Field Programmable Gate Array), o sea, es una matriz de puertas lógicas programables según el campo. Fueron creadas en 1984 por Ross Freeman y Bernard Vonderschmitt (que después fundaron XILINX).
Inicialmente se utilizaban puertas lógicas como componentes individuales y discretos que se soldaban uno a uno en nuestra tarjeta electrónica (ya fuese un circuito impreso o no). La evolución de esto fue las PAL (Programmable Array Logic) que eran chip que contenía un arreglo de puertas lógicas. Con la instrumentación adecuada la PAL se programaba quedando un circuito de puertas lógicas que no podía volver a ser programada. A pesar de esto supuso un avance respecto a lo anterior porque no había que soldar las puertas lógicas una a una ni había

que buscar cada tipo de puerta lógica en un cajón para colocarlas en nuestra tarjeta. Además, una vez colocada se podía reemplazar la PAL por otra PAL con otra programación y se podían realizar circuitos combinacionales que funcionaban como registros o conseguir en poco espacio un mayor número de entradas y salidas.
Por otro lado, en paralelo, estaba surgiendo la industria de los ASICs (Aplication-Specific Integrated Circuit), circuitos integrados de aplicación específica, que se diseñaban específicamente para un producto concreto con el fin de conseguir mayores integraciones de componentes en poco espacio. El diseño de los ASIC era complejo y caro y no eran fácilmente reutilizables (además se solían realizar en sala limpia). Tenían una ventaja a nivel conceptual y era la de que el ASIC tenía una arquitectura en bloques (eran como celdas integradas formando una unidad física).

Juntando los 2 paradigmas surgió la FPGA, que al igual que en los ASICs hay una estructura de bloques y al mismo tiempo se puede realizar una reprogramación de los mismos.
Posteriormente como consecuencia de utilizar lo mejor del mundo de los microprocesadores tipo ARM (por ejemplo Cortex) y las FPGAs, se crearon los SoC (System on Chip) donde coexisten ambos entes compartiendo recursos.

Los lenguajes de programación de la FPGA son básicamente el VHDL y Verilog aunque se pueden utilizar otros como SystemC, TCL, entre otros.

2.2 Introducción a la arquitectura xilinx all programable soc (system on chip)

2.2 .1 Motivos de uso del SOC

SoC es el acrónimo de System on a chip. Como su propio nombre indica, este tipo de dispositivos, integra en un solo chip los diferentes componentes de un sistema. Un SoC muy utilizado es la Zynq de XILINX.

2.2.3 SoCs Programables de la Zynq-7000

		Gama Inferior			Dispositivos de rango medio			
	Nombre del dispositivo	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z010	7 Z 0 6	XC7Z 020	XC7Z 030	XC7Z 035	XC7Z 045	XC7Z 100
Sistema de Procesamiento	Procesador	ARM Dual Cortex A9 MPCore cton CoreSigh						
	Extensiones del Procesador	NEON y Punto de coma flotante de precisión doble/sencilla para cada procesador						
	Frecuencia Máxima	866 MHz			Hasta 1 GHz			
	Cache L1	Instrucciones de 32 KB y datos de 32 KB por procesador						
	Cache L2	512 KB						
	Memoria Interna del Chip	256 KB						
	Soporte de Memoria Externa	DDR3, DDR3L, DDR2, LPDDR2						
	Soporte de Memoria estática Externa	2 Quad-SPI; NAND; NOR						
	Canales DMA	8 (4 dedicados a lógica programable)						
	Periféricos	2 UART; 2 CAN 2.0B; 2 I2C; 2 SPI y 4 GPIO de 32 bits						
	Periféricos w/DMA empotrada	2 USB 2.0 (OTG); 2 Ethernet Gigabit trimodo; 2 SD/SDIO						
	Seguridad	Autenticación RSA de Boot Loader de primera generación; desencriptación y autenticación de 256 bits AES y SHA para arranque seguro						
Sistema de procesamiento para puertos interfaces de lógica programable (sólo interrupciones e interfaces primarios)		2 Maestros AXI de 32 bits; 2 esclavos AXI de 32 bits; 4 memorias AXI de 64 bits/32 bits; ACP AXI de 64 bits; 16						
Lógica Programable	Logica programable equivalente de la serie 7 de Xilinx	FPGA Artix-7	FPGA Artix-7	FPGA Artix-7	FPGA Kintex-7	FPGA Kintex-7	FPGA Kintex-7	FPGA Kintex-7
	Celulas lógicas Program	Celulas lógicas de 28K (≈430K)	Celulas lógic	Celulas lógicas de 85K (≈1.3M)	Celulas lógicas de 125K (≈1.9M)	Celulas lógicas de 275K (≈4.1M)	Celulas lógicas de 350K (≈5.2M)	Celulas lógicas de 444K (≈6.6M)

Tabla 2.1 Todos los SoCs programables de la Zynq-7000

2.3 Entorno integrado para la síntesis de ISE ó Integrated Synthesis Environment (ISE) de xilinx

2.3.1 Introducción

Una vez que se ha decidido utilizar una FPGA (en este caso de XILINX), si no se dispone de herramientas de alto nivel como ActiveHDL o Vivado, siempre se puede trabajar directamente con las herramientas que tiene XILINX (de más bajo nivel). Estas herramientas constituyen lo que se llama ISE (de las siglas inglesas Integrated Synthesis Environment) de XILINX, o sea, entorno integrado para la síntesis. Realmente con el ISE no sólo se hace la síntesis sino que se cubre todo el ciclo de desarrollo (desde el diseño hasta la introducción del código en la FPGA). Normalmente las herramientas de nivel alto suelen hacer llamadas a los elementos que constituyen el ISE para que ejerza de intermediario con la FPGA.

2.3.2 Metodología

Para cubrir el ciclo de desarrollo se pueden distinguir claramente las siguientes etapas:

- Creación del diseño
- Síntesis
- Introducción de las restricciones "constraints"
- Implementación
- Análisis y mejora de la implementación
- Configuración y programación de la FPGA

Además de estas etapas básicas, es habitual realizar simulaciones para comprobar tanto el diseño como la implementación.

A continuación se verán en detalle cada una de estas etapas

2.3.3 Diseño

Para la realización del diseño se debe tener claro que bloques conceptuales requiere el sistema que se va a diseñar. También se debe conocer los interfaces que va a tener cada uno de los periféricos con los que comunica la FPGA y los algoritmos internos que va a utilizar para llevar a cabo las funcionalidades de nuestro sistema.

El lenguaje más utilizado para realizar el diseño es el VHDL y este lenguaje se puede escribir directamente (sin hacer uso de código generado) o bien se puede utilizar código en VHDL producido por herramientas como por ejemplo: altium, simulink, etc.

Si tenemos un componente o un circuito en altium, este paquete tiene la opción de traducirlo a VHDL. Igualmente si se tiene un algoritmo en Simulink, el "code generator" de simulink traduce también el algoritmo a VHDL.

También XILINX facilita un conjunto de librerías ("CORE") con elementos muy usados que se pueden descargar e incorporarse al resto de código VHDL de nuestro sistema (por ejemplo, una FIFO).

2.3.4 Síntesis

Una de las características más importantes de un lenguaje de descripción del hardware es la posibilidad de generar un circuito físico a partir de una descripción de RTL o de comportamiento. El proceso de generación de una representación en puertas lógicas de una descripción en VHDL se denomina síntesis.

Dentro del ISE se cuenta con una herramienta llamada XST (XILINX Synthesis Technology) que sintetiza todo el código asociado a nuestro diseño: VHDL, Verilog, CORES, Constraints.

Después de la síntesis se generan ficheros NGR, ficheros NGC y ficheros LOG.

2.3.5 Implementación

La implementación tiene como misión convertir el diseño lógico procedente de la síntesis en un formato de fichero físico que puede ser descargado dentro de la FPGA.

2.3.6 Análisis

Básicamente se trata de analizar cómo se han implementado los constraints, los recursos del dispositivo, los tiempos y la alimentación. Este análisis se puede hacer con el editor de la FPGA o alguna herramienta como por ejemplo PlanAhead.

2.3.7 Depuración

La depuración posterior al análisis de la implementación se puede abordar con herramientas del tipo ChipScope.

2.3.8 Simulación

Se pueden realizar simulaciones en varias fases del ciclo de desarrollo de nuestro sistema:

- Simulación del diseño (ISim o Modelsim)
- Simulación de la implementación

2.4 Descripción de la estructura interna de una FPGA

La arquitectura de un dispositivo FPGA consiste en cinco elementos programables fundamentales:

2.4.1 CLB (bloques lógicos configurables):

Se pueden programar de diversas maneras logrando así una amplia gama de funciones lógicas. Cada CLB está compuesto por cuatro slices (figura 1) y estos a su vez contienen las llamadas LUTs (en inglés, Look up tables), las cuales son elementos basados en memoria RAM que se pueden usar como biestables o latches. Las LUTs pueden tomar la forma de un bloque lógico e implementar multiplexores, o bien utilizarse como elementos de memoria (RAM distribuida) donde cada una tiene una capacidad de hasta 16 bits. También puede utilizarse como un registro de desplazamiento. Las LUTs son el elemento fundamental para la síntesis de funciones lógicas.

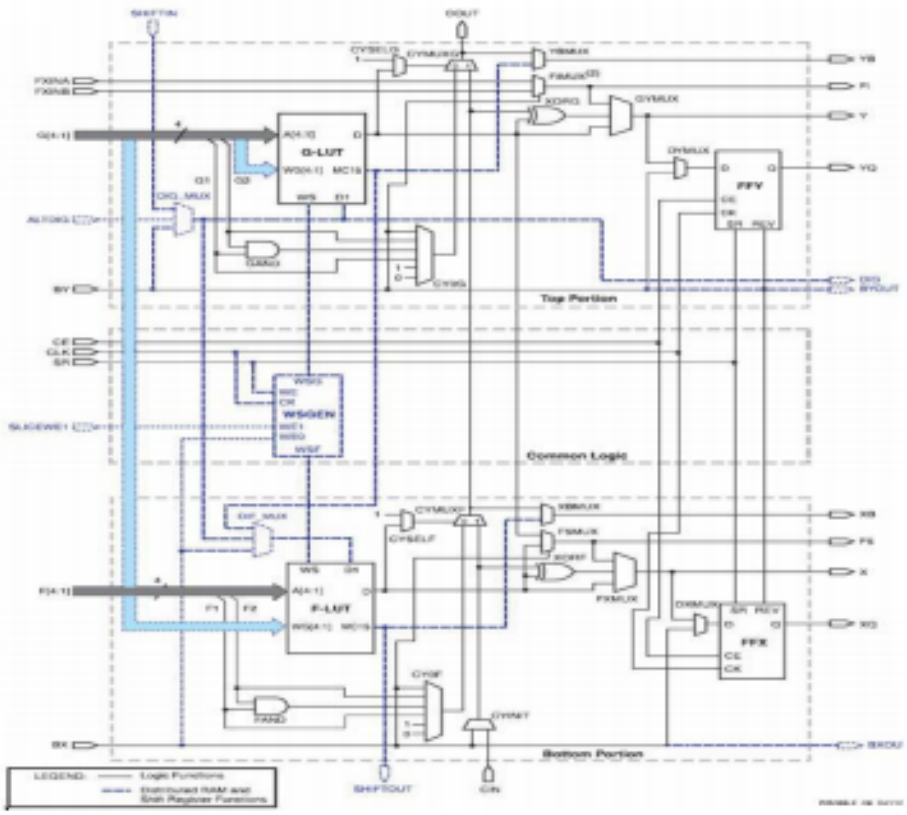

Figura 2.1 Estructura de un slice.

2.4.2 IOB (bloques de entrada-salida):

Se encargan del flujo de datos desde y hacia el FPGA a través de los pines del chip. Soportan flujos de datos bidireccionales, operaciones tri-estado, y un total de 24 estándares de señales. Poseen además control digital de impedancias.

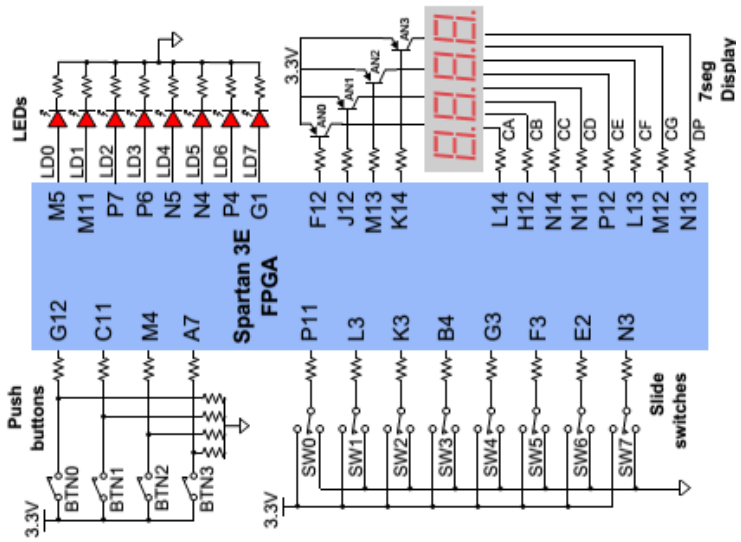

Figura 2.2 BASYS2 Entras/salidas.

2.4.3 BRAM o RAM de bloque:

Consiste en varios bloques (internos del FPGA) de 18 Kbits. Cada uno se comporta como un chip de memoria de doble puerto. Cada puerto tiene sus propias señales de control para las operaciones de lectura y escritura.

2.4.4 Multiplicadores:

Son bloques dedicados que efectúan esta operación entre dos números de 18 bits cada uno. A la salida se obtiene un número de 36 bits. Se puede asociar un bloque multiplicador con un bloque de RAM, de manera que se obtiene un multiplicador síncrono con las salidas registradas. La cercanía física de los bloques multiplicadores y los bloques de RAM posibilita esta característica. Haciendo multiplicadores en cascada es posible lograr la multiplicación de más de dos números e incluso de números de más de 18 bits. Figura 2. Multiplicador de 18 bits asíncrono. Multiplicador de 18 bits con salidas registradas.

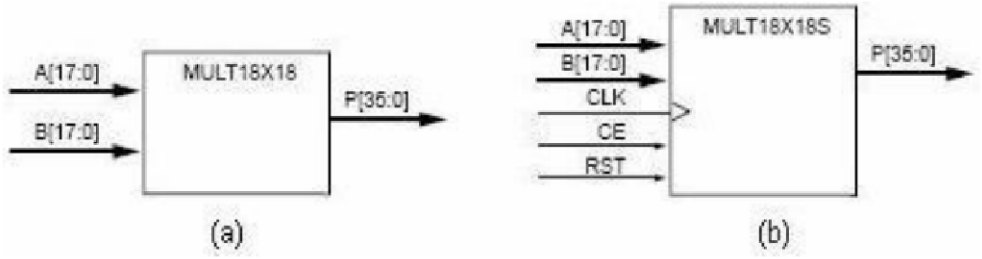

Figura. 2.3 Multiplicador de 18 bits asíncrono. Multiplicador de 18 bits con salidas registradas.

2.4.5 DCM (manejador de reloj digital):

Por lo general, los miembros de la familia Spartan 3 poseen cuatro DCMs. Estos elementos están destinados a proveer una señal de reloj de elevada exactitud. Eliminan los cambios de fase en la señal de reloj, así como las desviaciones de esta señal producto de perturbaciones externas, de altas temperaturas u otros efectos. Para esto implementan un DLL (en inglés, Delay-Locked Loop). El DLL rastrea las desviaciones de la señal de reloj y a través de una realimentación logra eliminar el error en la señal original. El DCM es capaz de proveer al sistema de un conjunto de señales desfasadas con respecto a la señal de reloj original [MicroC - FPGA].

2.5 FPGA Spartan 3

En la realización de este proyecto hemos configurado como dispositivo la FPGA Spartan 3 de XILINX, la cual viene integrada en la BASYS2, de DIGILENT. La placa BASYS2 es una completa herramienta de desarrollo a la hora crear diseños para FPGAs.

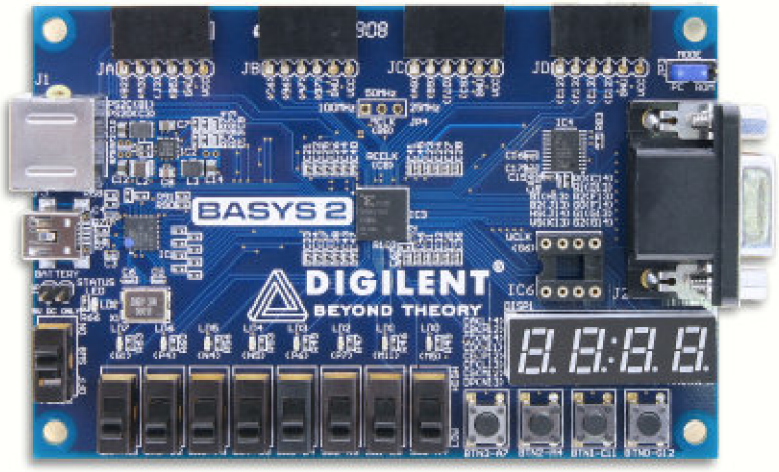

Figura 2.4 FPGA BASYS2

2.5.1 Características de la Placa:

- 100.000 puertas con las que cuenta la Spartan 3E FPGA.

- Puerto ATMEL AT90USB2 Alimentación/interface de programación.

- Plataforma Flash ROM P/ configuraciones de la FPGA.

- Reloj configurable (25/50/100 MHz), ranura p/ 2do reloj.

- ESD y protección de cortocircuito en todas las señales de Entradas/Salidas..

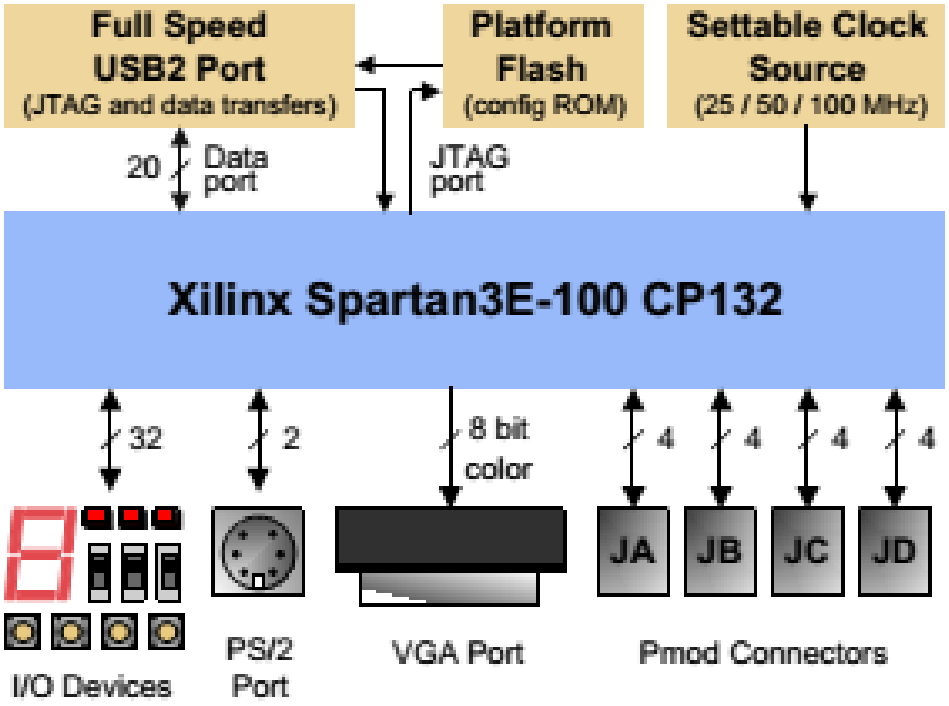

Figura 2.5. Diagrama de bloques y características de la placa BASYS2

CAPITULO 3: Consumo en CMOS y técnicas de bajo consumo

Como introducción al tema de bajo consumo, se describen en este capítulo los principios fundamentales y las principales componentes del consumo en los circuitos de tecnología CMOS. Además también presentaremos las principales técnicas utilizadas tradicionalmente en la reducción del consumo, con especial atención en las consecuencias de la aplicación de dichas técnicas en las tecnologías de lógica programable o configurable.

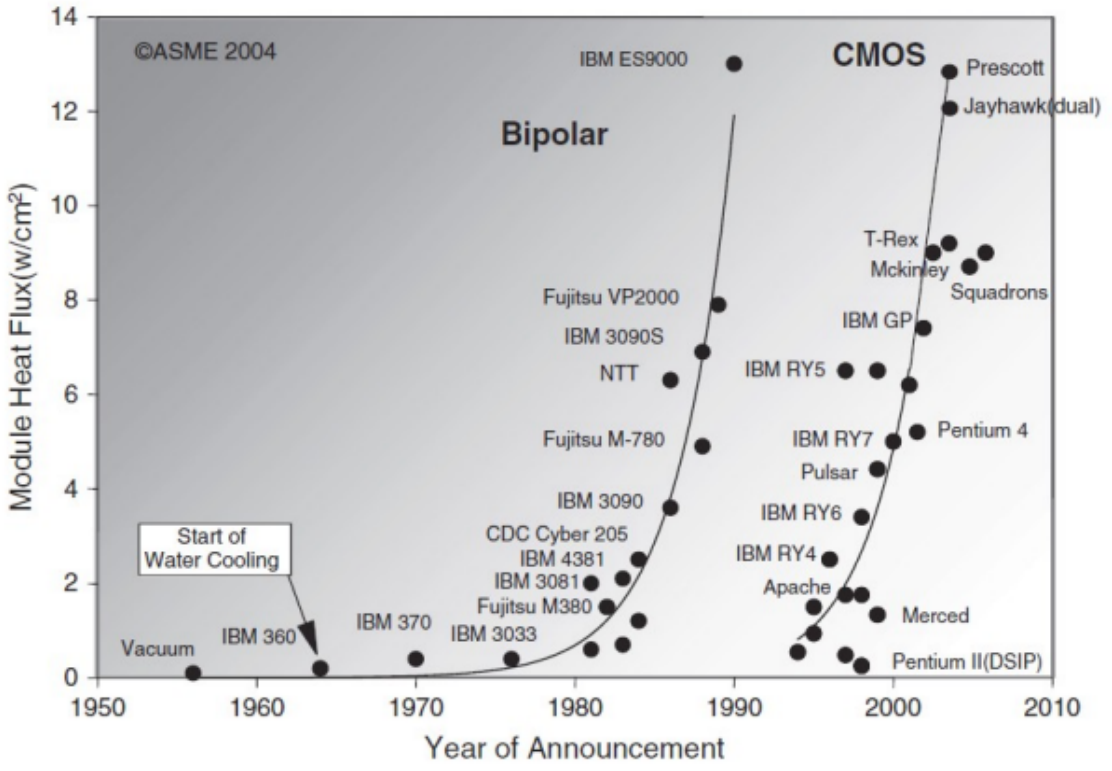

Figura 3.1 Consumo en Tecnología CMOS y Bipolar en los últimas décadas.

La gráfica muestra la introducción de la tecnología CMOS sobre la tecnología bipolar en las últimas décadas se puede deducir la importancia que ha cobrado la tecnología CMOS hoy en día.

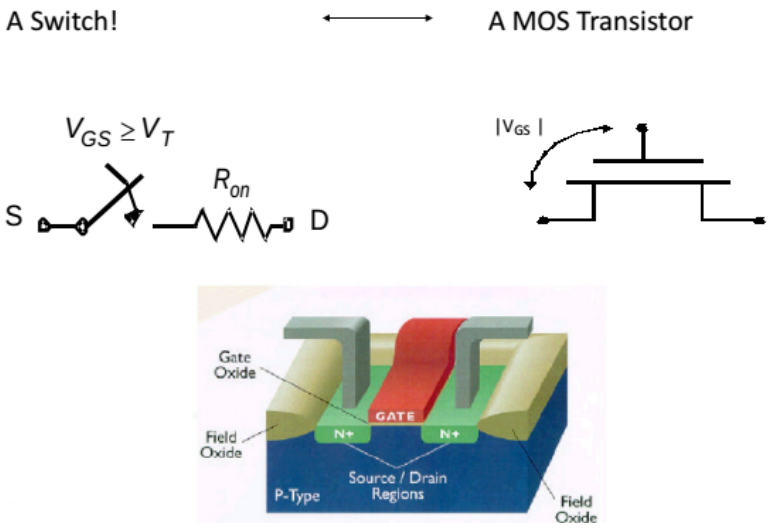

Figura 3.2 Transistor MOS estructura interna.

3.1 Consumo en circuitos CMOS

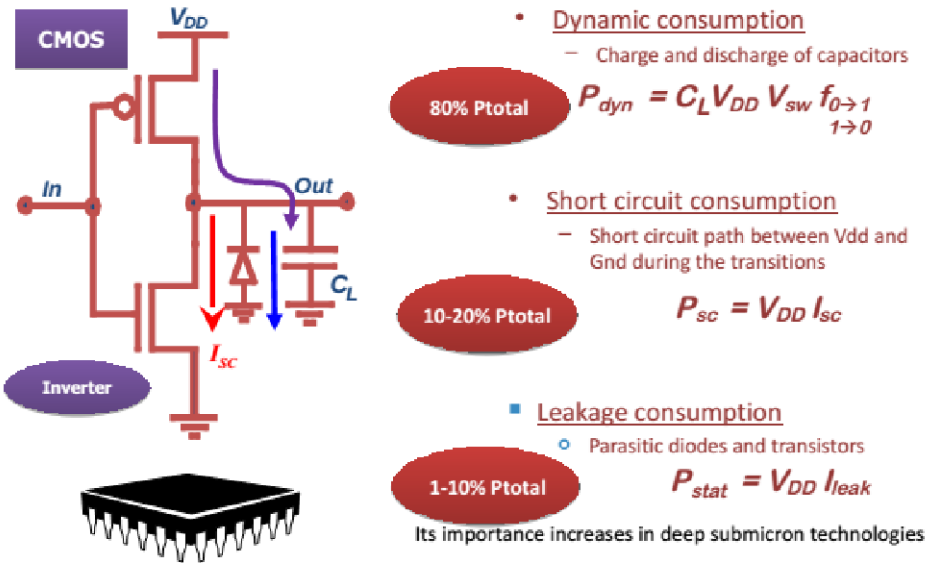

Figura 3.3 Consumo de Potencia en circuitos CMOS.

Para empezar analizamos las fuentes de consumo en circuitos CMOS clásicos. Para hacer un análisis consideramos un inversor CMOS como el descrito en la Figura 3.3 hay dos tipos de disipación de corriente en los circuitos integrados aquellas generadas durante las operaciones estáticas y las generadas en las operaciones dinámicas.

El consumo estático se refiere a las corrientes que fluyen mientras no hay cambios de estado en los circuitos. El consumo dinámico, que es el más importante en la tecnología CMOS, depende de las cargas y descargas de las capacitancias en las transiciones [1].

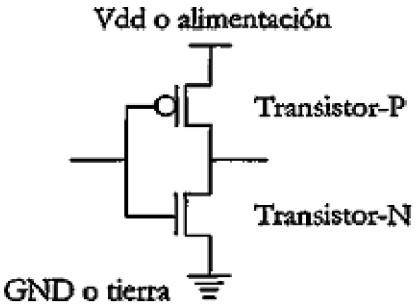

Figura 3.4 Inversor CMOS.

3.1.1 Consumo estático

En estado ideal los circuitos CMOS no disipan corriente estática en estado de equilibrio, es decir no existe un camino directo desde Vdd a GND [1]. No obstante existen dos fuentes de disipación estática a considerar [20]:

- **Corriente de fuga (leakage current)**: determinada fundamentalmente por la tecnología de fabricación y referida a dos fuentes:

1) A las corrientes inversas de los diodos formados entre las difusiones de la fuente(source) y el colector (Drain) y la región del transistor MOS.
2) La corriente de subumbral (subtreshold current) que aparecen en las inversiones de tensión en la puerta (gate) por debajo de la corriente de umbral.

- **Corriente Stanby(Standby current):** que es la que siempre fluye desde Vdd a GND.

La corriente de stanby 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 está en un valor entre Vdd y Gnd. En general esta potencia es igual al producto entre Vdd y la corriente continua fluida entre la fuente y la tierra [1].

Tradicionalmente el consumo estático se ha considerado prácticamente despreciable en aplicaciones reales [20][34][36], no obstante en los últimos procesos Tecnológicos por debajo de los 100 nm puede alcanzar el 40% del consumo total [21][22].

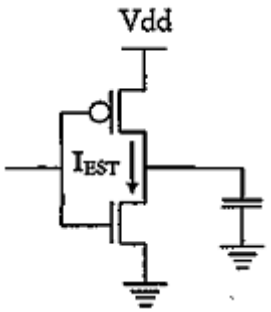

Figura 3.5 Consumo estático en un inversor CMOS.

3.1.2 Consumo dinámico

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

- Corriente de cortocircuito (Short-circuit current) la que es producida a través del camino que se forma entre la Vdd y GND durante las transiciones.

- Corriente de capacitancia (capacitance current), la que fluye para cargar y descargar las cargas capacitivas durante los cambios lógicos.

La corriente de cortocircuito surge durante la transición de estados de un dispositivo CMOS cuando al mismo tiempo trasmiten tanto los dispositivos pMOS como los nMOS como se ve en la figura de abajo. La corriente de cortocircuito para una puerta inversora es proporcional al tiempo de subida o bajada de la señal(rampa), la carga y el tamaño del transistor. Tradicionalmente la corriente de cortocircuito suele ser menor del 15% de la corriente dinámica [23], no obstante los últimos procesos tecnológicos por debajo de los 0,25 nm esta componente suele superar el 20 % [24]. Más aún según [25] un diseño descuidado puede aumentar significativamente dicho valor. Tal lo expuesto, si bien no es despreciable, tampoco es un factor dominante en el consumo.

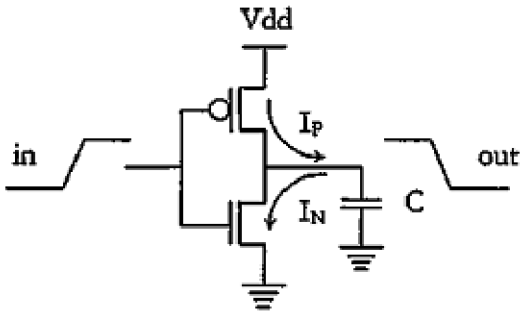

Figura 3.6 Consumo dinámico en un cambio de estado del inversor.

El factor dominante en la disipación de potencia en los circuitos CMOS es debida a la carga y descarga de las capacitancias de cada nodo [1].

Esta fuente de disipación es la llamada corriente de capacitancia y viene dada por la fórmula:

$$P = \frac{1}{2} \cdot C_L \cdot V_{dd}^2 \cdot E_{(sw)} \cdot f_{clk} \quad \text{[Ecuación 3.1]}$$

Dónde:

C_L = Capacitancia física a la salida de un nodo.

V_{DD} = *Tensión de alimentacion*

E_{sw} = Actividad de conmutación, número medio de transiciones

F_{clk} = Frecuencia de 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 [1].

Para el caso de las FPGAs valen las mismas consideraciones anteriores, es decir la corriente estática ha sido mucho menor que la dinámica [Gar02], pero el consumo estático comienza a ser una de las mayores preocupaciones [27][28] Se puede afirmar que la corriente estática depende de la tecnología del dispositivo y del tamaño(cantidad de bloques de cálculo, patas, etc).

El porcentaje de ocupación no es desde el punto de vista del consumo estático demasiado relevante. El consumo dinámico en los dispositivos re-programables es también proporcional a la frecuencia y al cuadrado de la tensión [1].

3.2 Espacio de diseño para bajo Consumo:

Como expresa en la ecuación del 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 esos factores [1].

Desafortunadamente estos factores no son independientes entre si y en ocasiones no pueden ser optimizados independientemente uno de los otros. A continuación se realiza una clasificación de ellos [1].

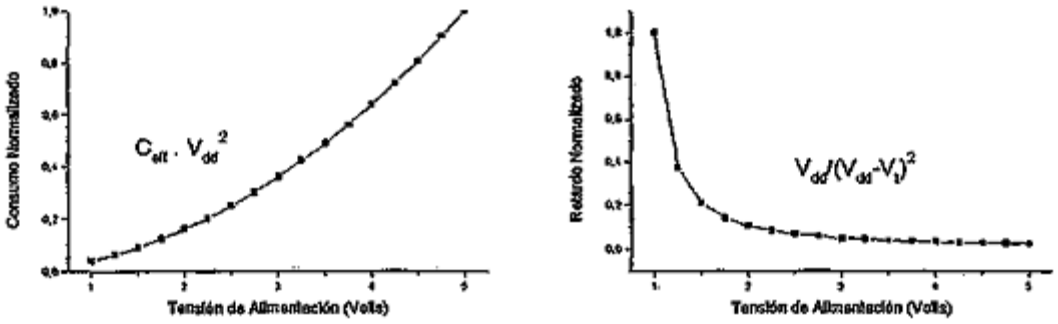

Figura 3.7 Relación energía-tensión y retardo-tensión respectivamente.

3.2.1 Tensión de alimentación

Sin dudas el más atractivo de los parámetros a manipular dada su influencia cuadrática en la ecuación del consumo, es la tensión de alimentación. Muchos fabricantes están dispuestos a sacrificar capacidad en el circuito actividad de los datos con tal de reducir la tensión. Lamentablemente se paga un costo en velocidad muy alto cuando V_{DD} se acerca V_t (tensión umbral)[1]. Esto limita la reducción de consumo a un mínimo de dos o tres veces la tensión de umbral como se aprecia en la Figura 3.7. Los retardos en un circuito son proporcionales a:

$$\boldsymbol{T \propto V_{DD}/(V_{DD} - V_t)^2} \quad \textbf{[Ecuación 3.2]}$$

La aproximación más atractiva a la hora de reducir tensión sin pérdida de velocidad es la reducción de la tensión umbral (Vt). El límite para la reducción de Vt está dado por el margen de ruido soportado y el aumento en la corriente de fuga de subumbral [1].

En cuanto a los circuitos de lógica programable, si bien existen familias operando a diferentes tensiones (5V y 3,3V tradicionalmente) y la familia Spartan 3(2,5V, 1,8 y 1,2V), no es aconsejable con vista a la interoperación con otros dispositivos operar un circuito fuera de los rangos provistos por el fabricante [1]. Existen no obstante trabajos en esta línea donde se prueban diseños operando a diferentes tensiones [26], los que permiten evaluar la relación consumo tensión respecto de la merma en el rendimiento.

3.2.2 Capacidad:

La reducción de la capacitancia ofrece una disminución lineal del consumo, de modo que la opción es otra atractiva fuente de minimización. La capacidad en un circuito CMOS depende mayoritariamente de dos factores, por un lado los dispositivos (puertas lógicas) y por otro las interconexiones. Conforme las tecnologías siguen reduciéndose, las capacitancias de las interconexiones empiezan a ser más importantes [1].

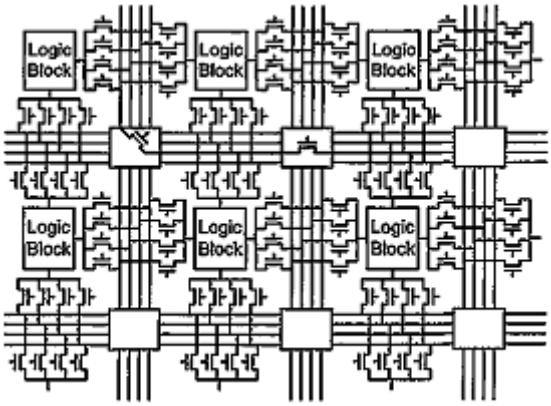

Figura 3.8 Rutado simplificado en dispositivos tipo FPGAs.

En las FPGAs es muy significativa la capacidad debido a las interconexiones. Es causa de más del 50% del consumo [29][30][31], en los dispositivos reprogramables, los canales de rutado son fijos y se debe pasar por una o múltiples matrices de interconexión para conectar dos elementos de cálculo, he ahí uno de los motivos para esta gran influencia del rutado [1].

Para la reducción del consumo debido a las capacidades en FPGAs existen básicamente dos posibilidades. La primera y más evidente es la reducción del largo de la pista. La utilización de herramientas optimizadas de emplazamiento y rutado (Place and Route) ayudan a la disminución del consumo. La segunda resulta en la actividad o bien aumentar los recursos de rutado necesario [1].

3.2.3 Actividad del circuito

Al margen de la tensión y la capacidad, la otra fuente de consumo dinámico es la actividad en el circuito. La actividad del circuito depende de dos factores. Por un lado la frecuencia de reloj fclk que especifica la periodicidad promedio de arribo de datos y E(sw) que determina cuantos cambios puede determinar cada arribo. La actividad es muy difícil de calcular, dado que depende de las entradas y de la función a computar, por ejemplo, la actividad dentro de un multiplicador de 16 bits puede variar en un factor de 5 dependiendo de la correlación entre los datos de entrada [32].

En la figura 3.10 se observa la interpretación de estos conceptos de fclk, el que se puede ver como la frecuencia de funcionamiento de un circuito síncrono y E(sw) que expresa cuantas transiciones se desarrollan en un ciclo de reloj. En un circuito sin glitches(fallos), E(sw) puede ser interpretada como la probabilidad que ocurra una transición durante un ciclo de reloj, en tanto en un circuito con glitches(fallos) especifica tanto la probabilidad de ocurrencia de una transición, así como la cantidad de movimientos espurios generados por esta [1].

Los glitches(fallos) 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 [1].

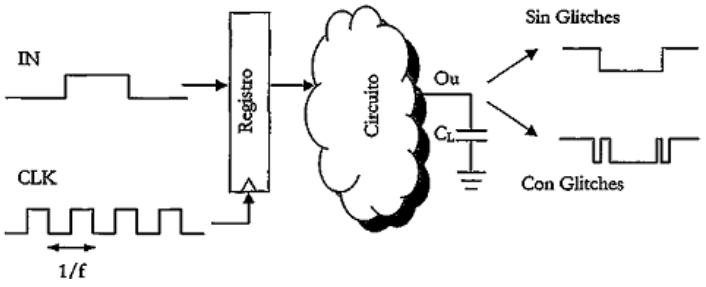

Figura 3.9 Interpretación de la actividad en circuitos síncronos: Glitches(fallos).

Por comodidad, se suele combinar la actividad de los datos $E_{(sw)}$, con la capacidad física $\left[\frac{1}{2}.C_L\right]$ para obtener capacidad efectiva:

$$C_{eff} = \frac{1}{2}.C_L.E_{(sw)}.f_{clk} \quad \text{[Ecuación 3.3]}$$

Que es la que se refiere a la capacidad promedio cargada en cada ciclo de reloj, transformando así la expresión del consumo en:

$$P = \frac{1}{2}.C_L.V_{dd}^2.E_{(sw)}.f_{clk} = C_{eff}.V_{dd}^2 \quad \text{[Ecuación 3.4]}$$

Esto refleja que el consumo no depende ni de la frecuencia de funcionamiento, ni de la capacidad, ni de la actividad de conmutación por separado, sino de la integración e interacción de estos factores en la capacidad efectiva [1].

La figura 3.9 ilustra la existencia de transiciones espurias producidas por la diferencia en el retardo de dos caminos. El efecto se refuerza con el aumento de la profundidad lógica, que produce avalanchas de glitches [1].

Como en el caso de la tensión y la capacidad, existen técnicas para la reducción de actividad como método de reducción del consumo. Por ejemplo, ciertas representaciones de datos como las de signo y magnitud poseen inherentemente menor actividad que otras como los datos representados por ejemplo en complemento a dos [35]. Dado que la implementación de las operaciones en signo magnitud son más complejas que en complemento a la base se debe pagar un costo extra en superficie de silicio así como consecuentemente en capacidad del circuito. Como siempre el problema de optimización de las tres variables tensión, capacidad y actividad no puede ser considerado independientemente y sin considerar el impacto sobre los demás factores [1].

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 la 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 [1].

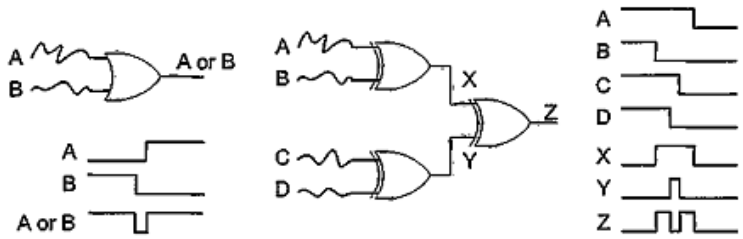

Figura 3.10 Transiciones espureas (glitches). a. Producidos por el desbalance en el arribo de las señales. B Efecto avalancha debido al aumento de la profundidad lógica.

Los glitches en los dispositivos FPGAs son muy significativos y se ven magnificados por el efecto de retardo, que originan las matrices de interconexión [1].

3.3 Temas destacados en el Diseño para bajo consumo

En la ecuación del consumo se puede observar que el consumo es determinado por tres parámetros importantes como son: la tensión, la capacidad del circuito y la actividad del mismo. No obstante la optimización de estos no puede ser interpretada individualmente sino que debe ser interpretada conjuntamente [1].

Quizás el tema más importante es el balance entre área-velocidad para bajo consumo.

La mayor parte de las técnicas de bajo consumo son a expensas del área o la velocidad, valga como ejemplo la reducción de la tensión de alimentación la que conlleva un ahorro cuadrático del consumo para afectar directamente a la velocidad de ejecución [1].

Si el diseñador no puede perder performance en la reducción del consumo posiblemente se vea obligado a utilizar técnicas de segmentación(pipelining) o paralelización para compensar la velocidad de ejecución, aumentando inevitablemente el área del circuito [1].

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 aún 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 arboles de reloj globales tienden a atentar contra el bajo consumo [1].

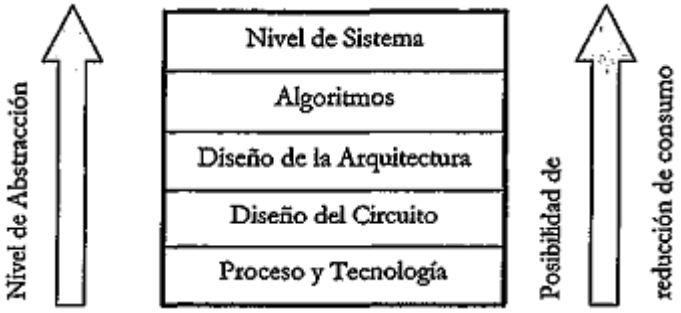

Figura 3.11 Jerarquía en el espacio de diseño.

Otro tema importante cuando se habla de bajo consumo es “evitar derroches”, entre estos se cuentan evitar glitches ecualizando caminos, “apagar” partes del circuito cuando estos no se utilizan (quitando el reloj o bloqueando los datos de entrada), otras fuentes de derroches de consumo son la no utilización de algoritmos y arquitecturas regulares, forzando de esta manera la lógica y la actividad [1].

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 [1].

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 cinco y se detallan en la Figura 3.11 de arriba.

3.4 Nivel de proceso y tecnología

En el nivel más bajo de abstracción se considera el proceso de fabricación así como el encapsulado del circuito. En cuanto a las FPGAs estos factores son determinados por el fabricante y existe poco margen de elección, más allá de elegir ciertos dispositivos dentro de la gama de oferta de circuitos reconfigurables, los que eventualmente operen a menor tensión, utilicen un proceso de fabricación más moderno, provean arquitecturas que se adapten mejor al bajo consumo [1].

3.4.1 Encapsulado

Generalmente una fracción importante del consumo total puede ser atribuida a las grandes capacitancias manejadas fuera del circuito(off-chip power) en vez de al circuito o núcleo(core) en sí. Esto se apoya en el hecho de que las capacidades fuera del chip se miden en pico-faradios. Circuitos con gran interacción de entrada/salida, pueden consumir ¼ ó ½ de la potencia total [1].

La tecnología de encapsulados ha evolucionado y sigue evolucionando rápidamente producto del aumento de densidad de los circuito y la multiplicación que han sufrido las entradas y salidas de los circuitos integrados. Con esto también aparecen nuevos estándares en entrada-salida que reducen considerablemente la capacidad de las patas(pads) de conexión [1].

Los encapsulados de las FPGAs siguen las tendencias de toda la industria de circuitos integrados(IC). Es importante destacar, que el aumento en la potencia de cálculo que experimentan las nuevas familias de dispositivos también hacen posible integrar en un solo circuito más lógica, lo que eventualmente redunda en una menor actividad de entrada/salida. Las FPGAs han evolucionado de ser una solución para generar lógica de conexión a ser utilizadas para implementar lo que se da en llamar Soc(System on a Chip) o sistema en un chip, haciendo relativamente menos voluminosa la comunicación con otros dispositivos electrónicos [1].

3.4.2 Proceso de Fabricación

Al margen de las consideraciones del encapsulado, la tecnología de fabricación posee una importante componente en la reducción del consumo. Varias modificaciones en el proceso de fabricación reducen el consumo en los circuitos integrados CMOS [1].

Muchas de estas ocurren con cada nueva generación de procesos de fabricación (reducción del tamaño de las puertas lógicas, agregado de capas de metal, etc). Estos parámetros no están disponibles para los diseñadores de circuitos integrados y mucho menos para quien diseña con FPGAs, no obstante es importante conocer las causas del consumo y las posibilidades de reducción en este nivel.

3.4.2.1 Optimización de la tensión umbral (Treshold Voltaje Vt)

Cuando se pretende reducir la tensión como método para la disminución del consumo, la tensión umbral comienza a tener gran importancia dada su directa influencia en la velocidad de operación (inversamente proporcional a $(\boldsymbol{V_{DD} - V_t})^{\boldsymbol{2}}$).

Lamentablemente, la conducción subumbral y consideraciones de márgenes de ruidos limitan cuan baja puede ser situada $\boldsymbol{V_t}$ llamada tensión umbral.

3.4.2.2 Reducción en la tecnología (Technological scaling)

La reducción de 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 óxido, ancho y alto de las interconexiones, etc. Si se considera que le capacidad de una puerta es $\boldsymbol{C_g = W.L.C_{ox}}$ donde, $\boldsymbol{W}$ es el ancho de la puerta, $\boldsymbol{L}$ el largo de la puerta, y $\boldsymbol{C_{ox}}$ es la capacidad por unidad de superficie, entonces si decimos que si reducen por un factor de $\boldsymbol{S}$ cada una de las dimensiones $\boldsymbol{W, L\ y\ C_{ox}}$ entonces la capacidad $\boldsymbol{C_g}$ se reducirá también por un factor de $\boldsymbol{S}$, de este modo si la tensión y la actividad de los datos se mantienen la potencia se disminuirá en un factor de S también $\boldsymbol{P \propto \frac{1}{S}}$ [1].

3.4.2.3 Trazado (Layout)

La técnica más simple a nivel Layout es colocar las pistas de mayor actividad en las capas superiores. Los niveles superiores de metal están separados por una capa más gruesa de dióxido de silicio. Dado que la capacidad física decrece linealmente con el incremento del espesor de óxido, existen claras ventajas de rutar las líneas de mayor actividad en las capas superiores. Según [36] 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.

3.4.2.4 Dimensiones de los transistores

Otro punto de discusión en bajo consumo es el tamaño de los transistores.

Transistores con puertas grandes pueden manejar más 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 buen funcionamiento lo que no siempre es aceptable [1].

3.5 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 de la FPGA. Nuevamente aquí se discute la relación de funcionamiento, área y consumo ahora desde este nivel de abstracción. Aquí se discuten mapeo(mapping), reducción de glitches o mal funcionamiento y la actividad, formas de sincronización, así como métodos de concurrencia y redundancia para la disminución de consumo [1].

3.5.1 Descomposición tecnológica y mapeo

se denomina descomposición tecnológica al proceso de transformar una descripción booleana a nivel de puertas en una mapeo para la arquitectura elegida. En el caso de implementaciones por ejemplo en ASICs será elegir la configuración de transistores que realizara la función, o en caso de lógica reconfigurable elegir la forma de mapear los recursos básicos (LUTs, lógica de acarreo, Block RAMs, multiplicadores embebidos, etc), de acuerdo a la funcionalidad de la descripción [1].

3.5.2 Reordenar las entradas

Una simple estrategia para disminuir la actividad total, será posponer las señales que poseen mayor movimiento. Por ejemplo, simplemente reordenando las entradas en una cascada de cálculos se puede reducir la cantidad de transiciones totales como en la Figura 3.12. La probabilidad de que exista transición T_r para una señal en la que se conoce la probabilidad de ser uno (p) en un modelo sin retardos viene dado por la expresión:

$$T_r = 2.p.(1-p) \quad \text{[Ecuación 3.5]}$$

Naturalmente es la suma de la probabilidad de que pase la señal de 0 a 1 más la que pase de 1 a 0.

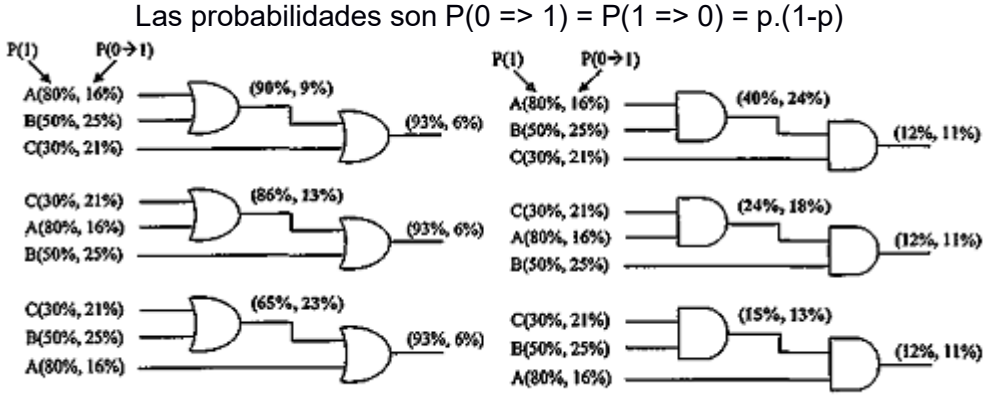

Figura 3.12 Dependencia del orden de entrada para la reducción de actividad

3.5.3 Reducción de glitches

Una de las técnicas más importantes para la reducción de consumo es la de evitar las transiciones espurias asociadas a los glitches. En la Figura 3.13 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 a la salida amplificando el efecto [1].

En contraste si se logra una estructura donde la llegada de las señales a cada puerta sucede al mismo tiempo se evitan las transiciones espurias. Este concepto de "balancear" las estructuras arbóreas, se puede extender como técnica de reducción de consumo [1]. Algunos estudios sugieren reducciones de hasta 15-20% del consumo [20].

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 calculo que solo utilizan como retardos para lograr la ecualización de los caminos [1].

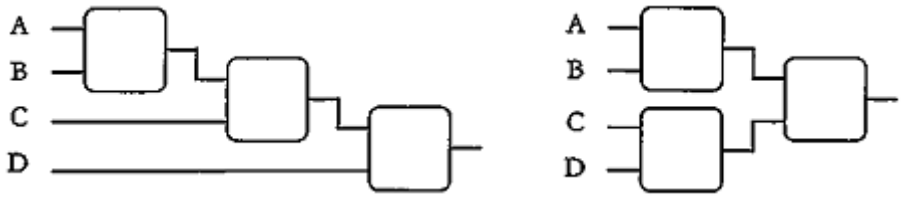

Figura 3.13 (a) Estructura en cascada (b) Estructura balanceada.

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 pipelining (segmentación) en la reducción de consumo [1].

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 [1].

3.5.4 Concurrencia y redundancia

La idea de la concurrencia es al aumento de las prestaciones del circuito (aunque a expensas del aumento del área) con el objeto de reducir la tensión de alimentación la que tiene una influencia cuadrática con el consumo. Cuando por cuestiones de diseño del circuito o de interrelación con otros sistemas, la tensión no puede ser modificada esta técnica pierde atractivo [1].

Por otra parte el objetivo de la redundancia, es básicamente reducir los glitches, o bien a través de la ecualización de caminos o bien 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 modelo a modo de evitar retardos indeseables que ocasionen glitches y aumentos de consumo [1].

El segundo caso, un fan-out muy grande implica puertas y pistas de mayor capacidad y eventualmente buffers, esto invariablemente genera retardos que pueden atraer movimientos espurios. La redundancia trae aparejada el aumento del área y la capacidad del circuito, de modo que debe sopesarse correctamente con el ahorro en potencia producida por los glitches [1].

3.6 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 suele llamar nivel de transferencia de registros (RT level) lo que para los científicos del área informática seria el 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 entre el área y la velocidad todo para reducir el consumo [1].

3.6.1 Procesamiento concurrente:

A nivel arquitectural el procesamiento concurrente es un excelente ejemplo de balance de área – velocidad para bajar el consumo. A través de técnicas conocidas para aumentar la performance como lo son el paralelismo o la segmentación (pipelining) se aumenta el rendimiento para luego bajar la tensión de alimentación y así bajar el 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 [1].

3.6.1.1 Paralelismo

Suponemos en el ejemplo de la Figura 3.14 que 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 puede paralelizar sin restricciones. Si se paraleliza el cálculo, duplicado N veces el bloque de cálculo A, se tendrán N procesadores idénticos los que podrán funcionar a una frecuencia N veces inferior y sin embargo mantener la velocidad total de cálculo [1].

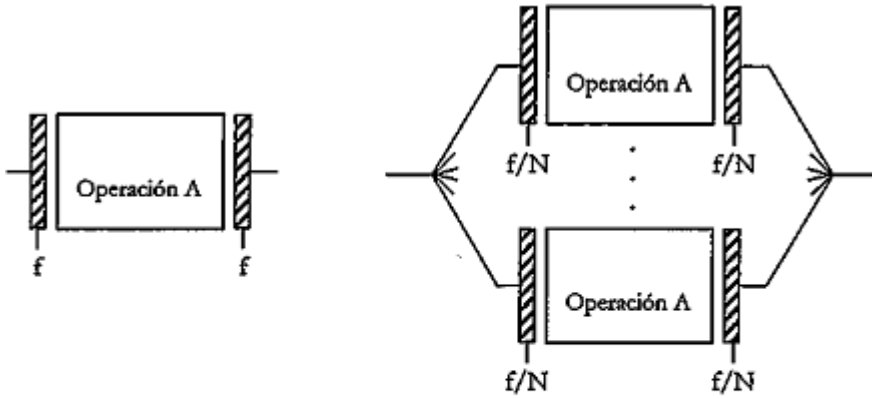

Figura. 3.14 (a) Reducción de tensión y paralelismo para bajo consumo.

La clave para la reducción del consumo es la posibilidad de reducir en un factor N la frecuencia de cálculo. Dado que la velocidad de cálculo se puede considerar aproximadamente lineal a la tensión de alimentación, esto permite reducir la tensión en un factor N. La capacidad total del circuito se incrementa N veces (dado que hay N procesadores iguales y despreciando la sobrecarga que representa la interconexión) [1].

Si partimos sobre la base de la ecuación donde se deduce que $\boldsymbol{P = C.V^2.F}$, se puede inferir que la potencia para una concurrencia de N niveles ($\boldsymbol{P_n}$) es:

$$P_n \alpha C.N\left(\frac{V}{N}\right)^2.\frac{f}{N} \alpha \frac{P}{N^2} \quad \text{[Ecuación 3.6]}$$

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 de N de la velocidad total. Frecuentemente los algoritmos no son totalmente paralelizados y existen tareas que deben ser necesariamente ejecutadas secuencialmente, a bien el nivel de paralelismo es limitado [1].

Otro aspecto importante es que 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 lo 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 esta tensión, como se aprecia en la **Ecuación 3.7**, la velocidad de un circuito se puede considerar lineal a la tensión cuando ($V_{dd} \gg V_t$), más precisamente la expresión dice que:

$$T \alpha V_{dd}/(V_{dd} - V_t) \quad \text{[Ecuación 3.7]}$$

Como se mencionó en la sección 3.1 la disminución de la tensión de alimentación en FPGAs no resulta muy atractiva, por tanto esta técnica pierde interés en el marco tecnológico de la lógica reconfigurable [1].

3.6.1.2 Segmentación (pipelining)

El pipelining 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 casa subetapa debe calcularse solo una N – é**sima** 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 [1].

Como ocurre con el paralelismo, el pipelining incurre en alguna sobrecarga aunque no tan notoria. Aquí se deben agregar tantas etapas de registro como etapas de pipeline se tengan, generando una mayor superficie, capacidad y necesidad de distribución de reloj, y esto puede disparar el consumo de registros de sincronización lo que puede superar el consumo debido al cálculo [1].

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 [37].

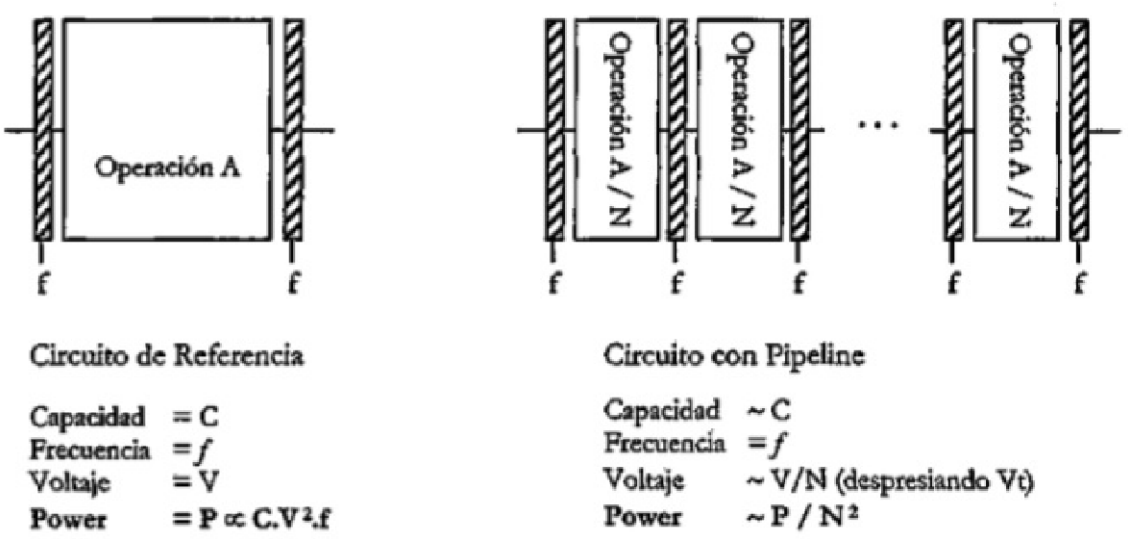

Figura. 3.14 (b) Reducción de tensión y pipeline para bajo consumo.

3.6.2 Manejo de potencia

Cualquier consumo de potencia que no produce una operación útil es un desperdicio de energía. Las estrategias para evitar el desperdicio de energía se denominan manejo de potencia o energía (power management), entre las técnicas utilizadas se cuenta con apagado selectivo (selective power down) modo de descanso (sleep mode), y sistemas adaptativos de la velocidad de reloj y tensión [1].

3.6.3 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 partición para aprovechar la localidad de los datos, el proceso distribuido es de mayor eficiencia que los procesos centralizados [1].
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 [1].

3.6.4 Representación de los datos

En esta técnica el diseñador dispone de diferentes alternativas, a escoger, por ejemplo, punto fijo versus punto flotante, signo magnitud versus complemento a dos, datos codificados versus datos sin codificar. Cada decisión involucra tener en cuenta varios aspectos como, exactitud, facilidad de diseño, prestaciones, área, consumo [1].
Por ejemplo podríamos utilizar punto fijo a punto flotante. El punto fijo necesita menos hardware y por ende menos consumo.

3.7 Nivel Algorítmico

Los algoritmos tienen influencia directa e indirecta sobre el consumo. La complejidad algorítmica y la cantidad de operaciones tienen una influencia directa en el consumo. Otras características como la posibilidad de aplicar concurrencia, pipelining, deshabilitación parcial del reloj o utilizar operaciones de menor consumo tienen una influencia más indirecta [1].

3.7.1 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 [1].

3.7.2 Algoritmos para arquitecturas de bajo consumo

Otro aspecto importante, aunque más sutil, es reconocer cuan bien se adecua un algoritmo a una arquitectura de bajo consumo. Para una eficiente implementación de algoritmos en arquitecturas de bajo consumo, existen características deseables, las cuales son la concurrencia y la modularidad [1].

3.8 Nivel sistema

Se considera dentro de este nivel la optimización de 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) [1].

3.8.1 Optimización de memoria

Estas técnicas intentan reducir el consumo tanto en el uso de memoria como en la comunicación y transferencia de datos. Muchas aproximaciones se centran en explorar sistemas de caches para reducir el consumo [6]. Otras técnicas buscan jerarquías más complejas de diferentes tipos de memoria (SRAMs, DRAMs, etc) controlando la transferencia de módulos y el emplazado [7].

3.8.2 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 óptima [17]. El mayor desafío en este tema es contar con potentes herramientas de estimación de consumo ya que sin una estimación eficaz es imposible optimizar posteriormente.

3.8.3 Optimización a nivel de instrucciones.

El enfoque de las optimizaciones a nivel de instrucciones es en general sobre sistemas de procesadores estandares. Estos métodos están basados en modelos del consumo de los procesadores en el que cada par de instrucciones tiene asociado un costo, la optimización consiste en elegir la combinación de instrucciones con menor(mínimo) consumo[18].

3.8.4 Manejo dinámico del consumo

Es un método de control que permite a los sistemas, o parte de ellos de operar en estado dormido (sleep mode) cuando están inactivos [1].

3.8.5 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 [19] [50] [51] [52] [53] [54] [55] [56].

Capítulo 4: METODOLOGÍA

Lo que se trata en este capítulo son las diferentes etapas en este proyecto para realizar la medida de potencia en la placa y utilizando la herramienta XPA de XILINX con la metodología que se describe a continuación.

- Estudio de la placa de medida y circuito de medida.
- Circuito de prueba y variaciones.
- Estudio del XPower Analyzer.
- Distintas pruebas realizadas con ambos métodos(XPA y Pruebas físicas) y comparación de resultados.

4.1 Placa de medida BASYS2 y circuito de medida

Empleamos en este proyecto la placa BASYS2 de Digilent.

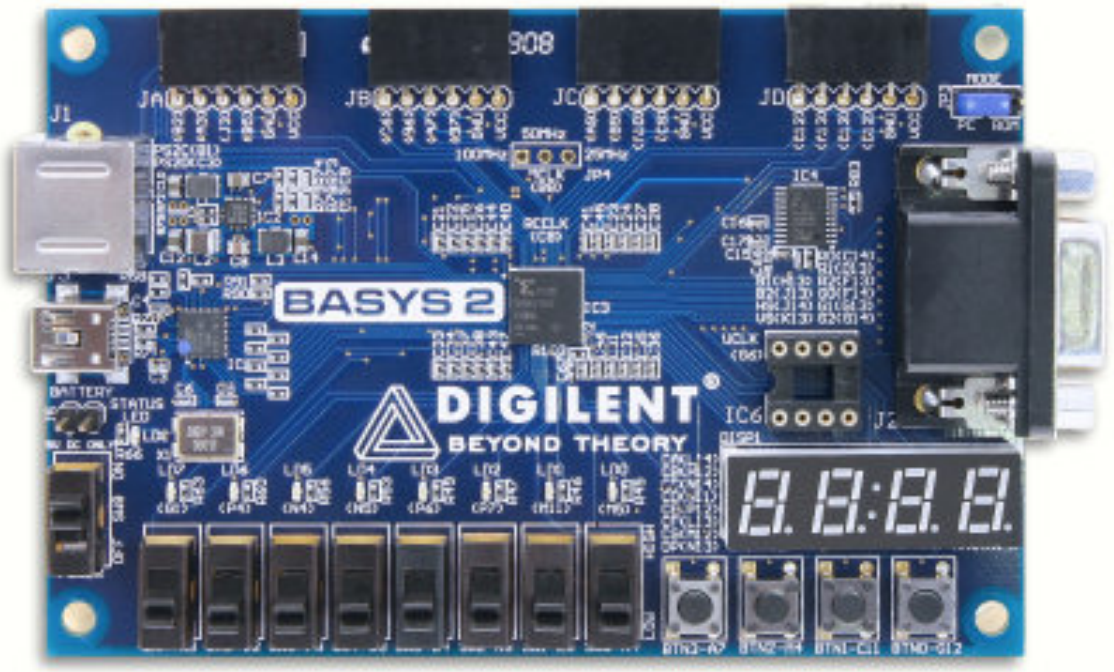

Figura 4.1 Placa BASYS2 de Digilent.

El modelo de FPGA con el que cuenta la placa es el XILINX Spartan-3E XC3S250E CP132que tiene 250000 puertas lógicas, también se pueden usar solo 100000 puertas lógicas dependiendo de cuál se elija a la hora de crear un diseño. Dispone de 91 Inputs/ Outputs accesibles para el usuario (IOB). Además posee 4 DCMs o Digital Clock managers, que son una especie de manejadores de reloj, y sirven estos para variar la frecuencia de trabajo del reloj en nuestro sistema.
La placa contienes dos destinos en los que se puede programar el circuito, la memoria RAM y la memoria ROM.

4.2 Circuito de medida

En nuestra búsqueda de un circuito de medida tuvimos en cuenta los dos parámetros para calcular la potencia que consume un elemento.
Tuvimos en cuenta que el circuito debe ser lo más simple posible para evitar pérdidas en los elementos y eliminar al máximo los errores que puedan surgir al hacer simplificaciones en cada elemento.
El circuito más simple a modo de evitar pérdidas de potencia es colocar en serie a la alimentación de la placa una resistencia pequeña o de Shunt que en nuestro caso el valor es de 10,1 Ohms, a la vez en serie con una fuente de alimentación continua que proporcione una fuente de tensión constante de unos 5 Voltios acorde a los requerimientos de la BASYS2.
El regulador de continua que utilizamos para proporcionar a la placa una tensión continua es el LM2596, y se muestra a continuación:

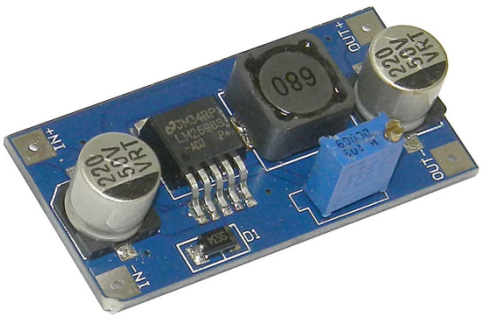

Figura 4.2 LM2596

Este circuito te permite tener un voltaje regulado a partir de una fuente de alimentación con un voltaje mayor, en nuestro caso una fuente de 12V que regulamos a aproximadamente 5V, para el uso con nuestra placa BASYS2.

Este módulo está basado en el Regulador DC-DC Step Down LM2596 que es un circuito integrado monolítico adecuado para el diseño fácil y conveniente de una fuente de conmutación tipo buck. Es capaz de conducir una corriente de hasta 3A. Maneja una carga con excelente regulación de línea y bajo voltaje de rizado. Este dispositivo está disponible con voltaje de salida ajustable. El módulo reduce al mínimo el uso de componentes externos para simplificar el diseño de fuentes de alimentación.

El módulo convertidor LM2596 es una fuente de alimentación en la cual su eficiencia es significativamente mayor en comparación con los populares reguladores lineales de tres terminales, especialmente con tensiones de entrada superiores.

4.2.1 Características

1. Basada en el regulador LM2596, salida entre 1,5 y 35Vdc
2. Voltaje de entrada: 4.5-40V
3. Voltaje de salida: 1.5-35V (Ajustable)
4. Corriente de salida: Máxima 3A
5. Dimensiones: 43*20*14mm
6. Frecuencia de conmutación: 150 KHz

Siguiendo con nuestro circuito de medida, podemos decir que al añadir esta pequeña resistencia contamos con un elemento simple mediante el cual se puede medir la tensión en la resistencia y poder así calcular la potencia en la misma.
Para medir la tensión en la resistencia se ha empleado un osciloscopio digital que permite la captura de datos y guardarlos en un archivo tipo plantilla de cálculo.

Figura 4.3 Osciloscopio USB Digital modelo RDS1021

4.2.2 Principales características del modelo RDS1021:

Model	RDS1021	RDS1021I
Trigger Type	Edge, Pulse, Slope	
Trigger Mode	Auto, Normal, Single	
Trigger Level	±5 divisions from screen center	
Acquisition Mode	Sample, Peak Detect and Average	
Cursor Measurement	△V and △T between cursors	
Automatic Measurement	Vpp, Vavg, Vrms, Freq, Period, Vmax, Vmin, Vtop, Vbase, Overshoot, Preshoot, Rise Time, Fall Time, +Width, -Width, +Duty, -Duty	
Waveform Math	FFT	
Communication Interface	USB2.0	
Dimension (W×H×D)	150 × 20 × 18 (mm)	
Weight (without package)	0.27 kg	
Bandwidth	25MHz	
Sample Rate	100MS/s	
Horizontal Scale (s/div)	5ns/div - 100s/div, step by 1 - 2 - 5	
Rise Time	≤14ns	
Record Length	5K	
Input Coupling	DC, AC, and GND	
Input Impedance	10MΩ±2%(X10), 1MΩ±2%(X1)	
Input Capacitance	20pF±5pF	
Max Input Voltage	50V (PK - PK) (DC + AC, PK - PK)	400V (PK - PK) (DC + AC, PK - PK)
DC Gain Accuracy	±3%	
DC Accuracy (average)	average≥16 : ±(3% reading + 0.05 div) for △V	
Analog Bandwidth	25MHz	
Probe Attenuation Factor	1X, 10X	
LF Respond (AC,-3dB)	≥10Hz	
Interpolation	sin(x)/x	
Displacement	±10div	
Interval (△T) Accuracy (full bandwidth)	Single : ±(1 interval time + 100ppm × reading + 0.6ns), Average>16 : ±(1 interval time + 100ppm × reading + 0.4ns)	
Vertical Resolution (A/D)	8 bits	
Vertical Sensitivity	5mV/div - 5V/div	

Tabla 4.1 Especificaciones de funcionamiento y principales características del modelo RDS1021.

Este modelo de osciloscopio USB es capaz de devolver un archivo en tres diferentes formatos, nosotros configuramos el descargar los datos con el formato plantilla de cálculo.

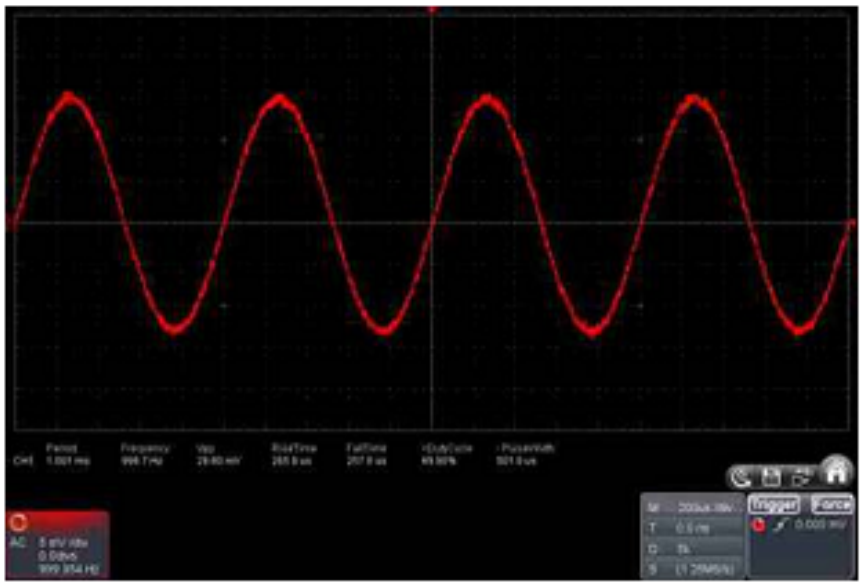

Figura 4.4 Aspecto de la interfaz de visualización, se muestra una señal en pantalla de 5 mVoltios.

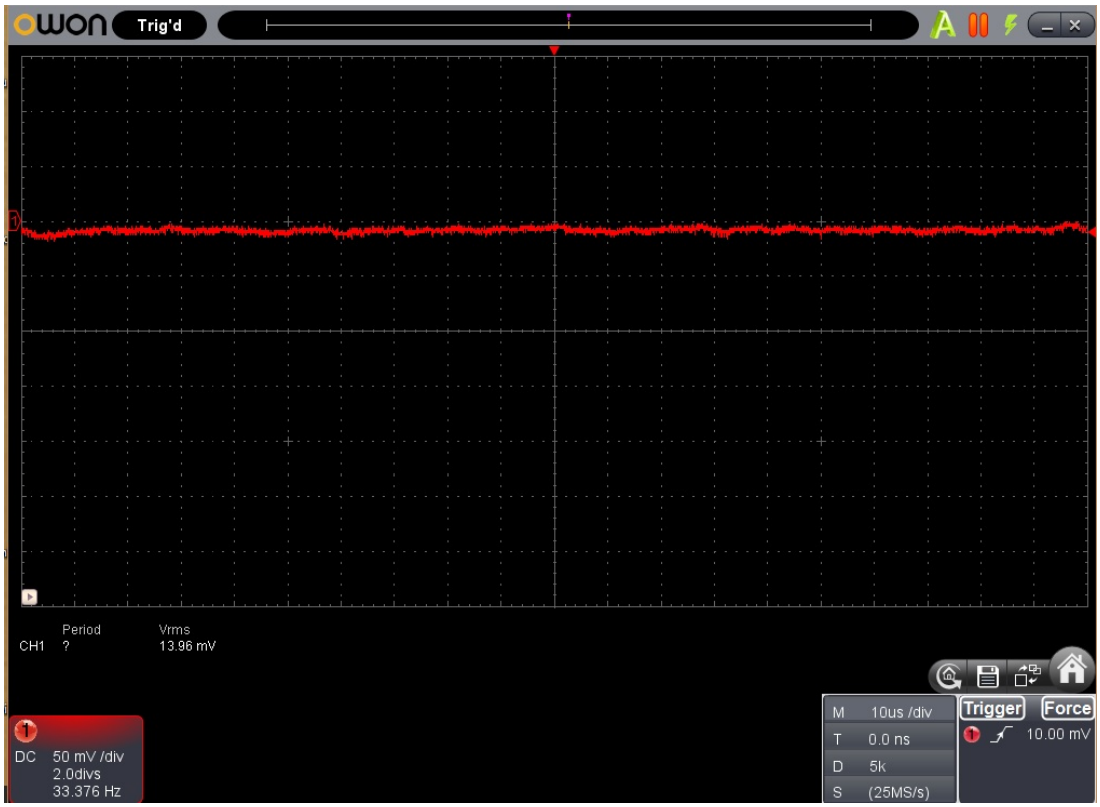

Figura 4.5 Variación de la tensión sobre la resistencia en serie con la fuente de alimentación y la BASYS2

Cuando tengo una longitud de registro(record length) de osciloscopio muy corta se comienzan a ver problemas en la amplitud muestreada, o cuando la propia señal incluye datos a frecuencias muy altas el cual no es nuestro caso a pesar de que el circuito trabaje a frecuencias altas, lo que medimos con el osciloscopio en cuestión son las variaciones en la fuente de alimentación para nuestra placa y teniendo en cuenta que el regulador de nuestra fuente de alimentación tiene una frecuencia máxima de operación de 150KHz lo que es muy inferior a los 25MHz de ancho de banda del osciloscopio utilizado para tomar las muestras, nuestro sistema de adquisición no deja lugar a problemas de esta índole aunque no se tengan este tipo de problemas como habíamos dicho cuando tengo una amplitud de registro muy corta y nuestra señal incluye datos a frecuencias muy altas y aparecen problemas de Aliasing, esto se solucionan

aumentando la longitud del registro del osciloscopio, en nuestro caso es bastante de unas 5000 muestras y el ancho de banda del mismo, en otras palabras, debemos tener al menos una frecuencia de muestreo del doble de aquella frecuencia más alta de la señal origen como dicta el teorema de Nyquist-Shannon.

$$F_s > 2F_{max} \quad \textbf{[Ecuación 4.1]}$$

Como hablamos ya afirmado arriba en el caso de este proyecto las fluctuaciones de la fuente no serán tan rápidas como la frecuencia a la que trabaja la placa con nuestro código.

4.3 Circuito de referencia

Tras proponer un método eficaz mediante el cual medir la potencia que consume la placa BASYS2, lo siguiente es disponer de un circuito que sirva de referencia para las pruebas en la placa (medidas físicas) y con la herramienta XPower Analyzer.
He desarrollado un código de mi autoría, basado en varios códigos a los que haré referencia al final del libro, en principio desarrolle un multiplicador genérico y un generador de números aleatorios, ambos están integrados y son el sistema base de todas las pruebas, este libro cuenta con dos apéndices donde explico a fondo cada uno de los códigos tanto del multiplicador como del generador de números aleatorios.

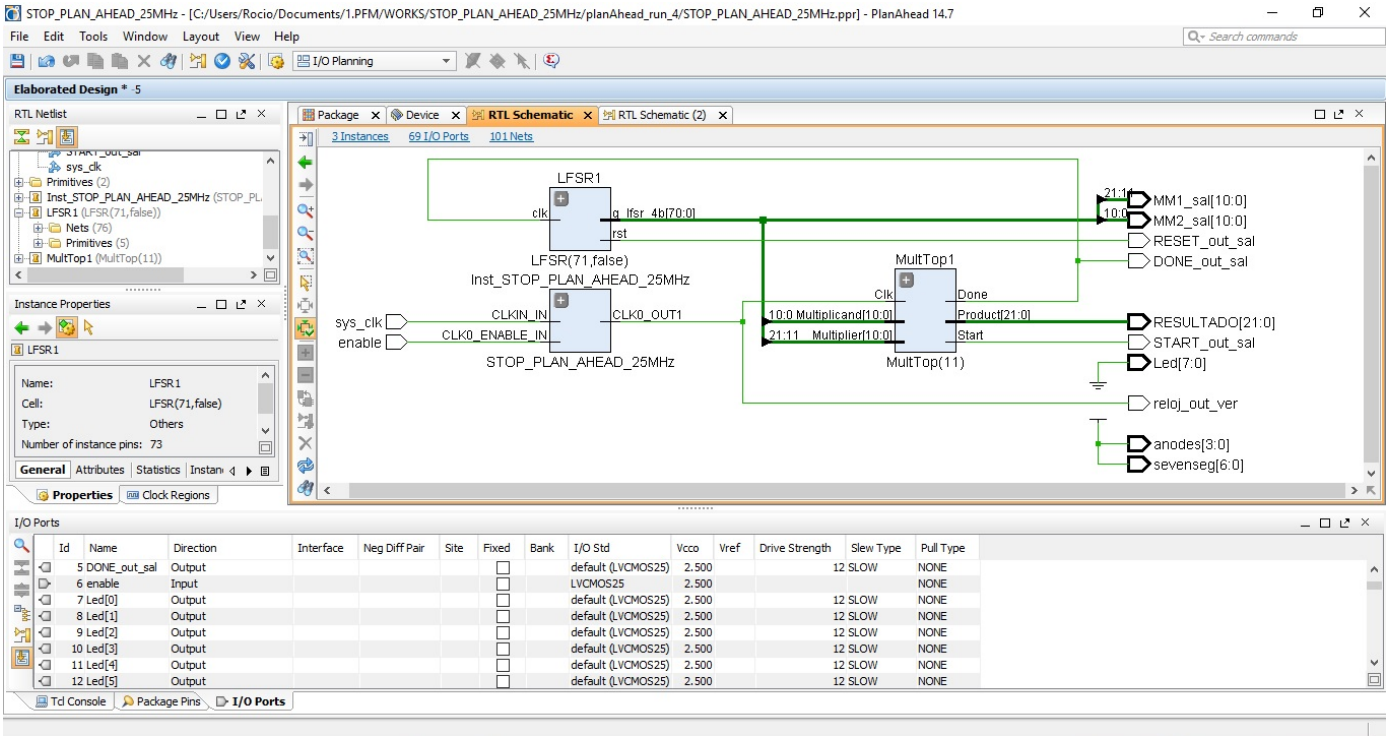

Figura 4.6 Esquema del circuito de referencia en PlanAhead.

Tanto el multiplicador como el generador de números aleatorios se diseñaron de modo a utilizar al máximo los recursos de la placa, que como se explicó anteriormente nuestra placa tiene un número limitado de entradas y salidas y el circuito de referencia necesitó de al menos 71 bits para la salida del generador de números aleatorios y además de 22 bits de salida máxima para el multiplicador, esto lo calculamos de modo a conseguir una utilización de aproximadamente el 100% de los recursos de la placa.

En principio diseñamos un Generador de números aleatorios de hasta N= 168 bits, pero poco a poco fuimos reduciendo empíricamente este número debido a que no era factible de realizarse este LFSR de 168 bits en la placa BASYS2, a continuación el sumario resultado del intento de implementación.

Los errores recogidos al principio de la implementación son los siguientes:

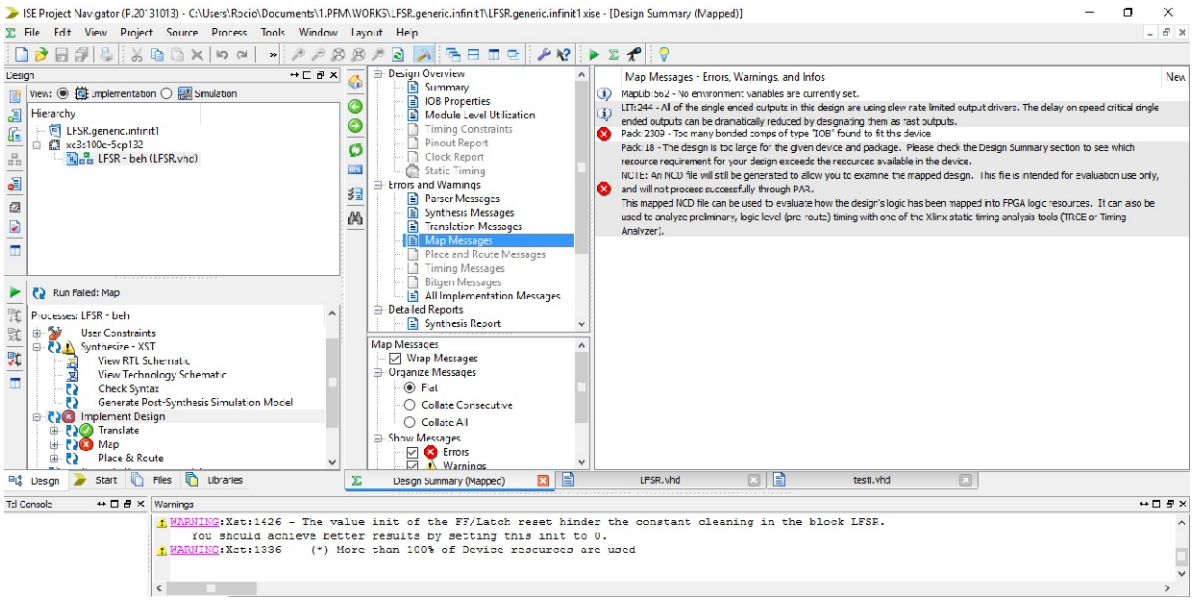

Figura 4.7 Errores de implementación P/ N=166 bits.

EL error pack 18 es muy común e indica que el diseño es demasiado grande para el dispositivo elegido.

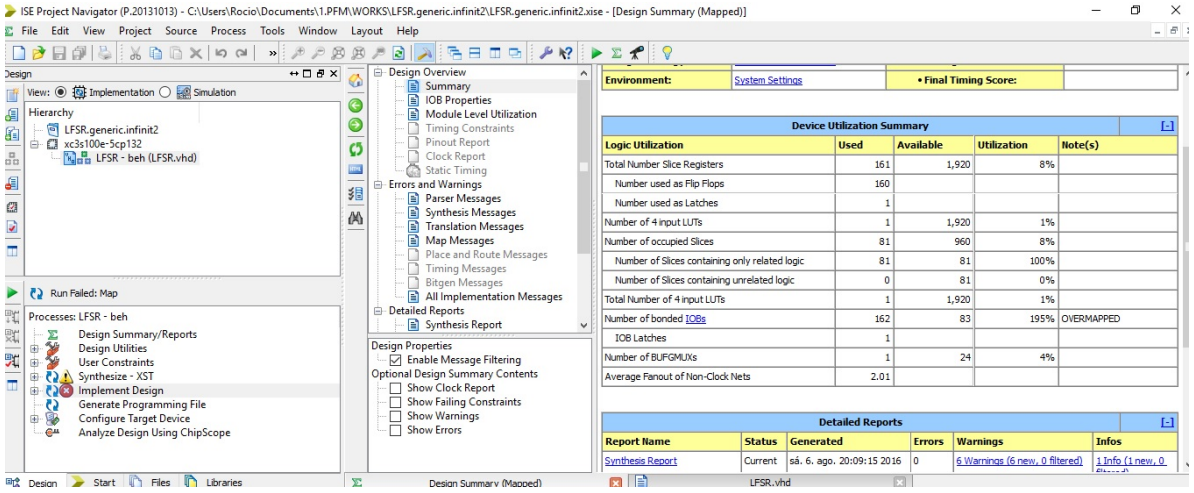

Figura 4.8 Sumario de utilización del dispositivo reportando sobre-mapeado de 195% en la placa P/ N= 160 bits.

Se puede notar que sobrepasa el número de Entradas/salidas que se pueden utilizar en la placa en un 195%.A partir del sumario anterior dedujimos por una regla de tres simple que el 100% de utilización de la placa correspondía a N= 82 bits, e intentamos de nuevo implementar el circuito, a lo que obtuvimos como resultado el siguiente sumario:

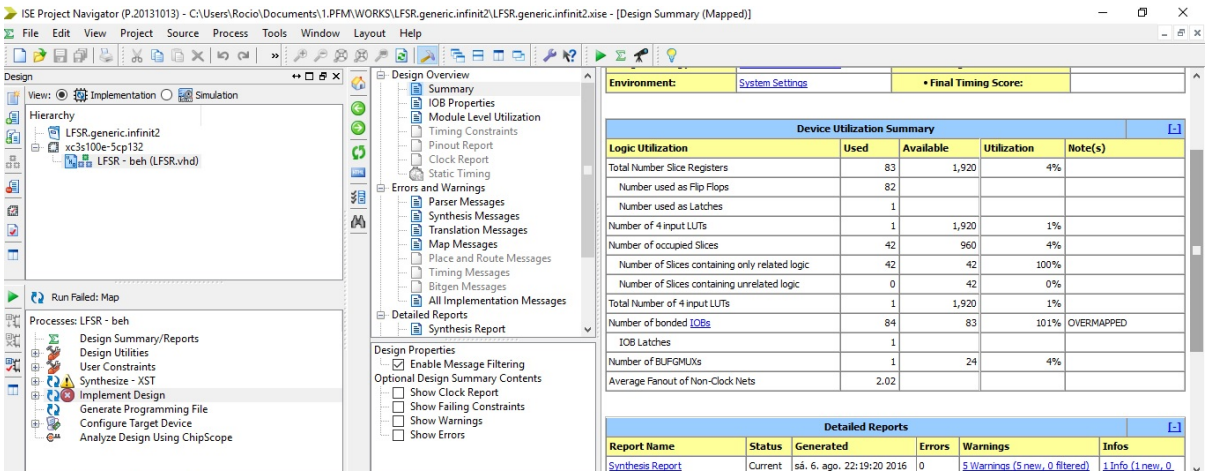

Figura4.9 Sumario de utilización del dispositivo para N= 82 bits.

Con N= 82 bits obtenemos una utilización de las entradas salidas de un 101% nuevamente la implementación no es posible esta sobre-mapeada en un 1% por encimad e las posibilidades de la BASYS2, con lo que continuamos reduciendo así el número de bits de nuestro LFSR a 71, con el que obtuvimos el siguiente sumario de utilización del dispositivo.

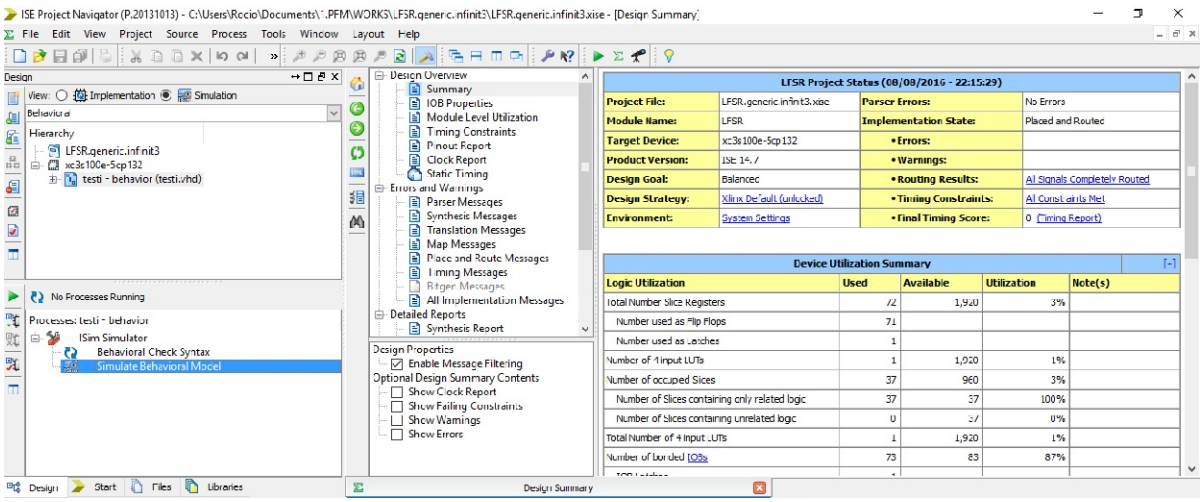

Figura 4.10 LFSR de 71 bits implementado en la BASYS2.

Se puede notar que no se reporta ningún sobre-mapeado en la placa, así que el LFSR implementado finalmente es de 71 BITs.

De la misma forma procedimos a la hora de acoplar el multiplicador al LFSR.

Empezamos por el número máximo de entradas y salidas para el multiplicador que se pueden implementar con un LFSR de 71 bits, que serían Entradas de 35 Bits, repartiendo el numero de 71 Bits en dos, para asignar dos números a los operandos, que serían de 35 bits cada uno, esto tampoco se pudo implementar también tuvimos que ir bajando el número de bits de entrada para los operandos de la multiplicación, y los resultados se muestran a continuación.

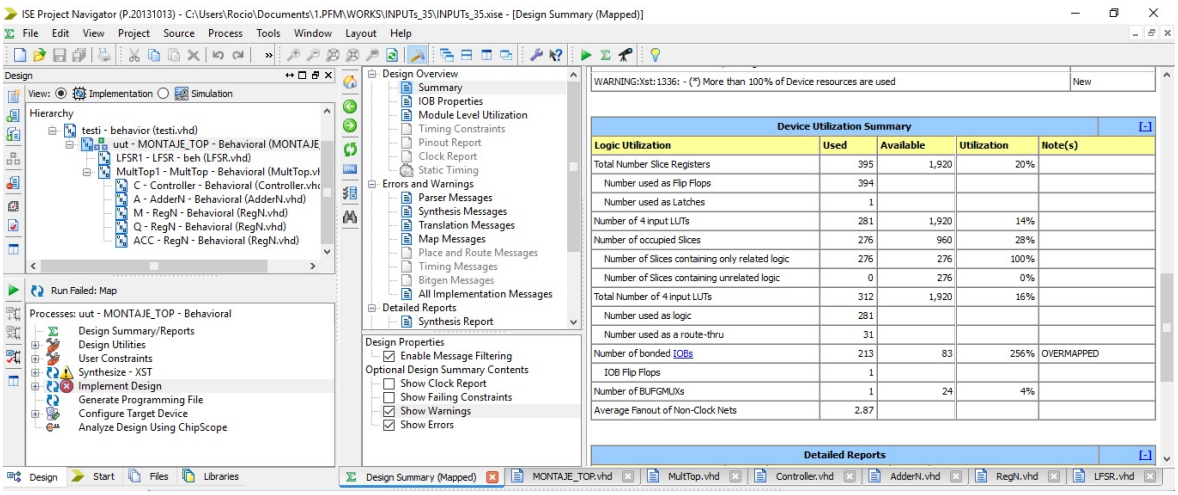

Figura 4.11 En el sumario del dispositivo tras la fase de implementación se nota un sobre-mapeado de 256% en el dispositivo para entradas de operandos de 35 bits.

Nuevamente inferimos por regla de tres que el 100% de utilización de la placa reduciría el número de Bits a 13, y el resultado de la fase de implementación reporto los siguientes errores.

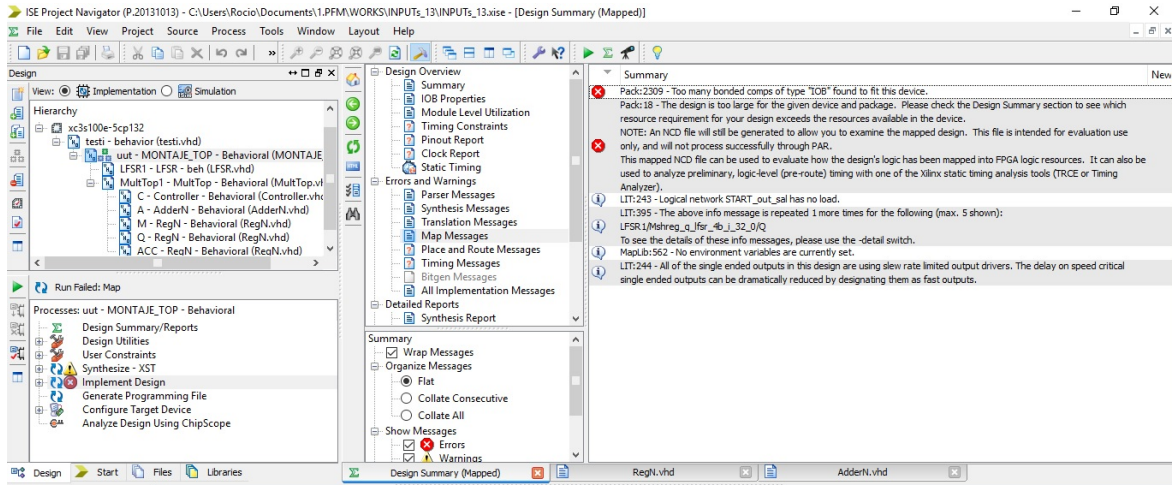

Figura 4.12 Errores en la fase de implementación para N= 13 bits.

Nuevamente obtuvimos el error pack 18 que nos indica que el diseño es muy grande para el dispositivo elegido, por lo que fuimos disminuyendo el número de bits para los operandos o entradas de nuestro multiplicador, y dimos con el numero N= 11 bits.

Al final nuestro circuito de referencia seria el Generador de números aleatorios o LFSR de 71 bits, integrado a un multiplicador genérico de operandos de máximo 11 bits con salida de máximo 22 bits.

Con esto las simulaciones que podríamos realizar son de 11 a 3 diferentes frecuencias de 25MHz, 12,5MHz y 5 MHz, probamos estas frecuencias con la ayuda de un jumper en JP4 de la placa BASYS2 y utilizando cores DCM.

4.3.1 Código final:

Al código integrando el multiplicador y el generador de números aleatorios también le agregamos la funcionalidad de probarlo tanto en funcionamiento normal (reloj a frecuencias 5MHz, 12,5MHz y 25MHZ) como a reloj parado para medir de esta forma la potencia estática consumida por la placa.

4.3.1.1 Modificación del código para agregar un habilitador(enable)

Primeramente configuramos el buffer de configuración global de reloj o BUFGCE del core DCM de la siguiente forma. Primeramente luego de crear el proyecto en el ISE se agrega el core DCM como una nueva fuente(new source) al proyecto como se muestra en la figura a continuación:

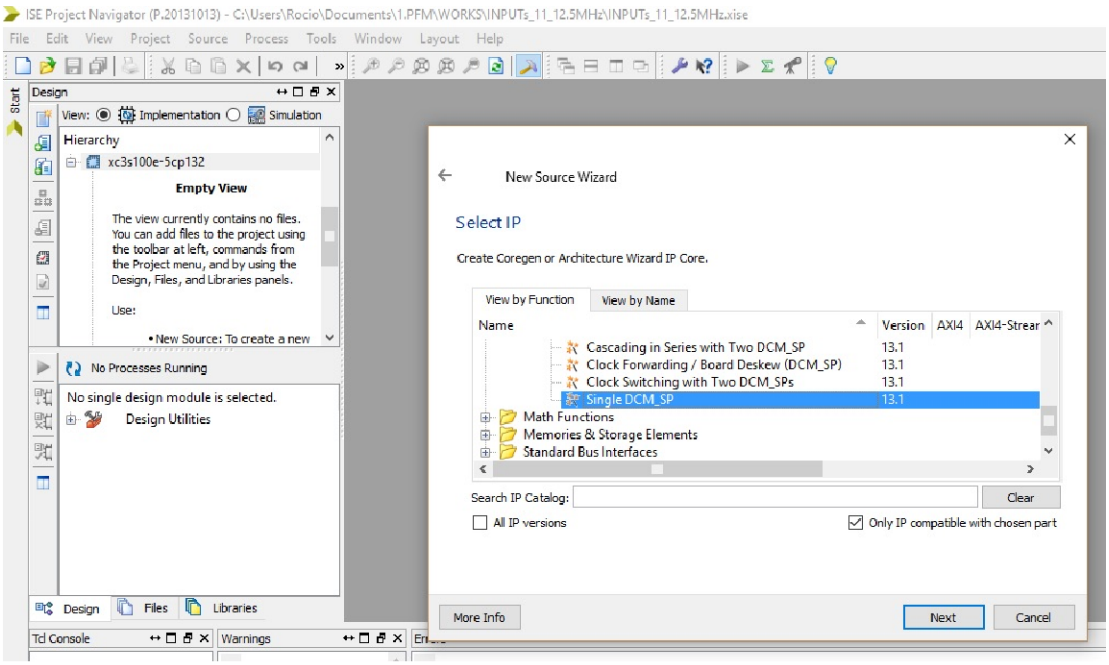

Figura 4.13 Adición del core como nueva fuente.

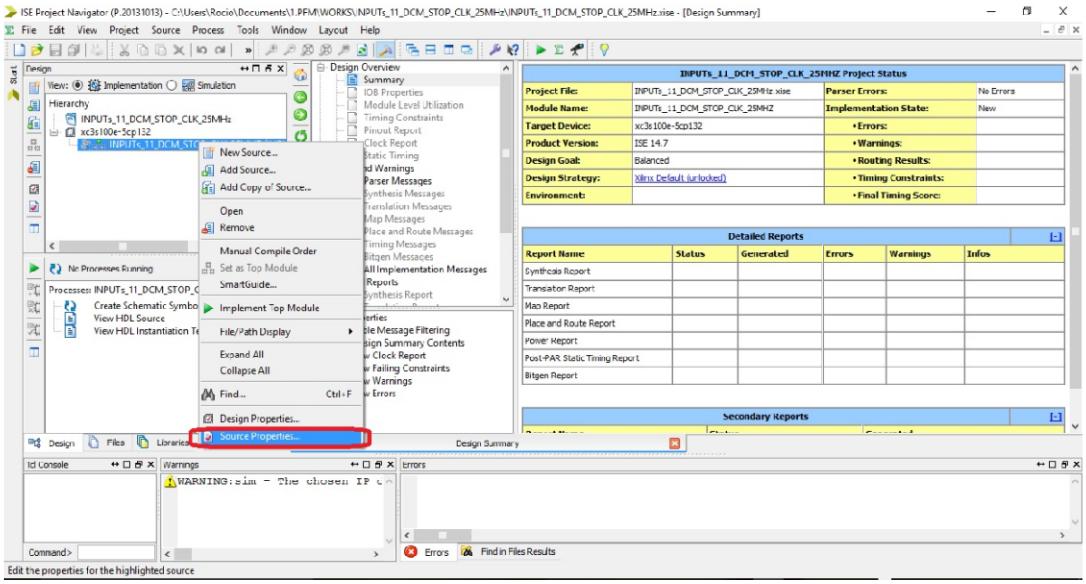

Figura 4.14 Apartado source properties para asignar archivo DCM como global.

Es sumamente importante designar el archivo de core como un archivo global en el proyecto, como se muestra en la figura 4.14 de arriba

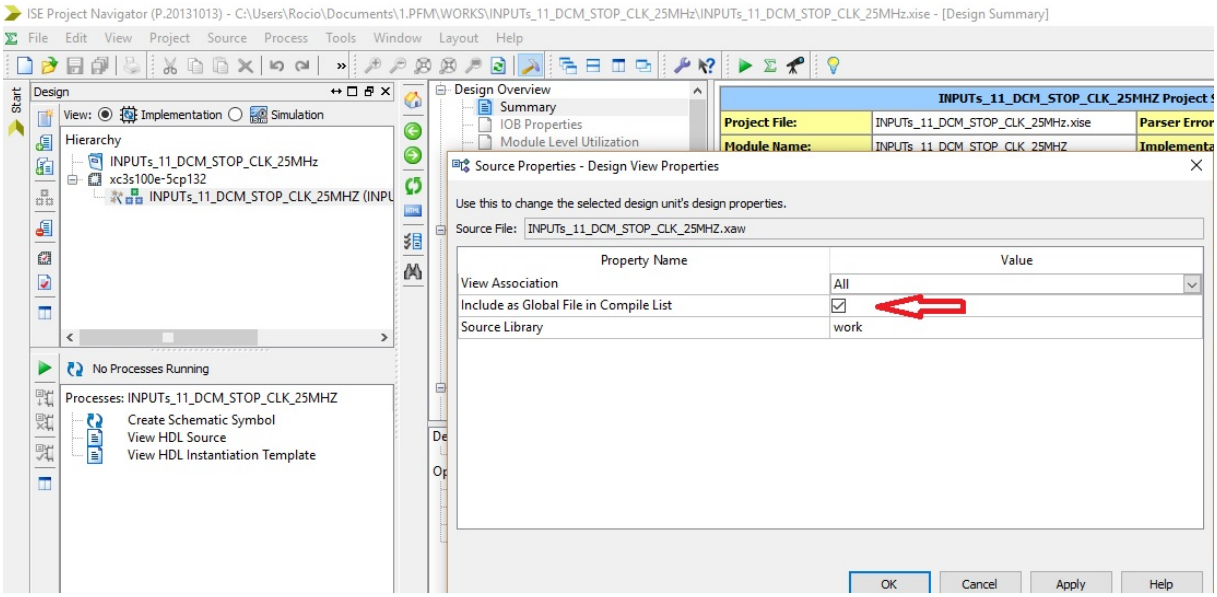

Figura 4.15 Core DCM como archivo global en la lista de compilación del proyecto.

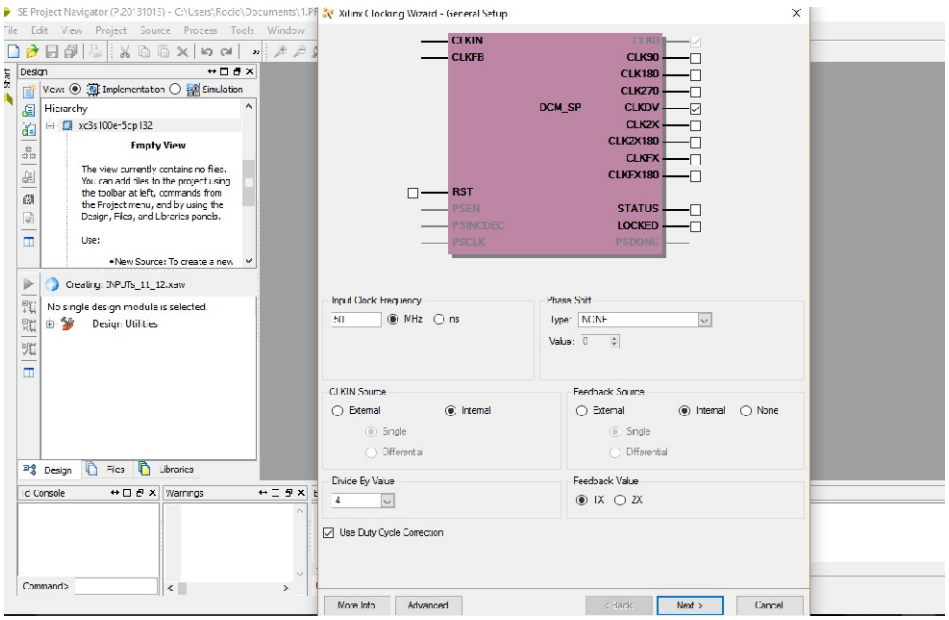

Figura 4.16 Adición de core DCM y configuración.

Los buffers de reloj pueden ser configurados como libre de glitches seleccionando la opción de Duty Cycle correction como se ve en la figura de arriba.

Luego configuramos que el reloj tenga un interruptor habilitador o enable.

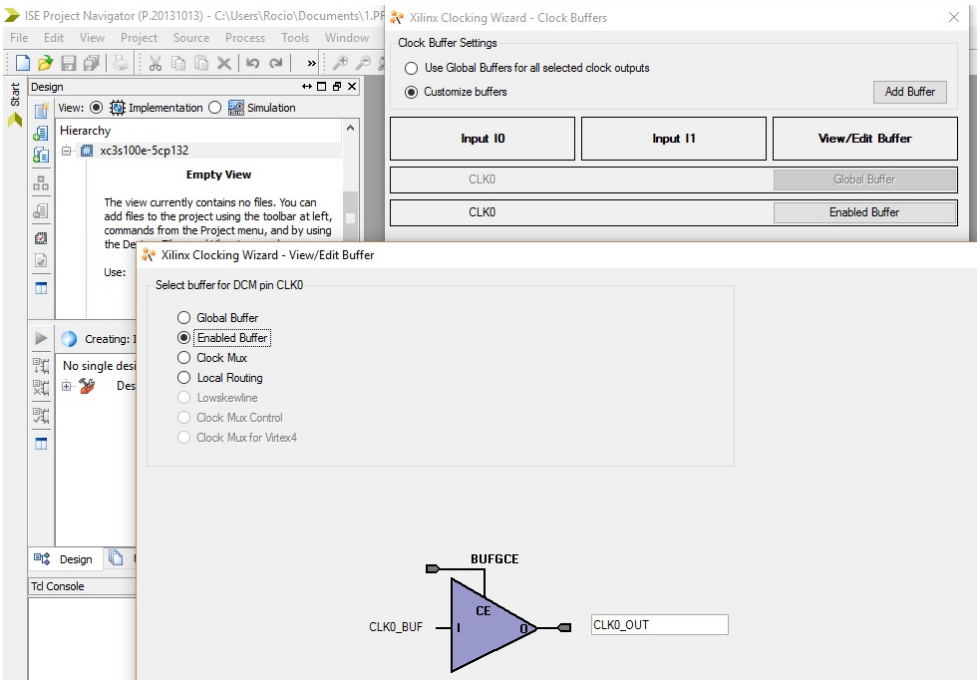

Figura 4.17 Asignación del habilitador de reloj o enable.

En cuanto a la codificación, es importante incluir la librería UNISIM en el archivo de TOP de la siguiente forma:

```
library UNISIM;

use UNISIM.VComponents.ALL
```

También se deben agregar definiciones de los atributos en nuestro código específicamente en el archivo del TOP en donde se definen las señales de la arquitectura antes del BEGIN, como se muestra a continuación:

```
attribute bufgce: string;

attribute bufgce of sys_clk : signal is "yes";
```

También en la misma área de definiciones se debe definir el Componente para el bloque DCM :

```
COMPONENT DCM

 PORT(  CLKIN_IN : IN std_logic;

        CLK0_ENABLE_IN : IN std_logic;

        CLK0_OUT : OUT std_logic;

        CLK0_OUT1 : OUT std_logic   );

 END COMPONENT;
```

En el área de instanciaciones de nuestro código asignamos las señales para el componente DCM, como sigue:

```
Inst_DCM: DCM

 PORT MAP(     CLKIN_IN => sys_clk ,
```

```
CLK0_ENABLE_IN =>  enable,

CLK0_OUT => signal_clk_original,

CLK0_OUT1 => signal_clk_result );
```

La señal *"sys_clk"* es la señal de reloj(clock) global del sistema, la señal que habilita el reloj del sistema es la llamada *"enable"*, y la señal de reloj resultante es *"signal_clock_result"*, la señal *"signal_clock_original"* es el reloj original de nuestro sistema. A continuación se muestra el resultado de la simulación de nuestro código:

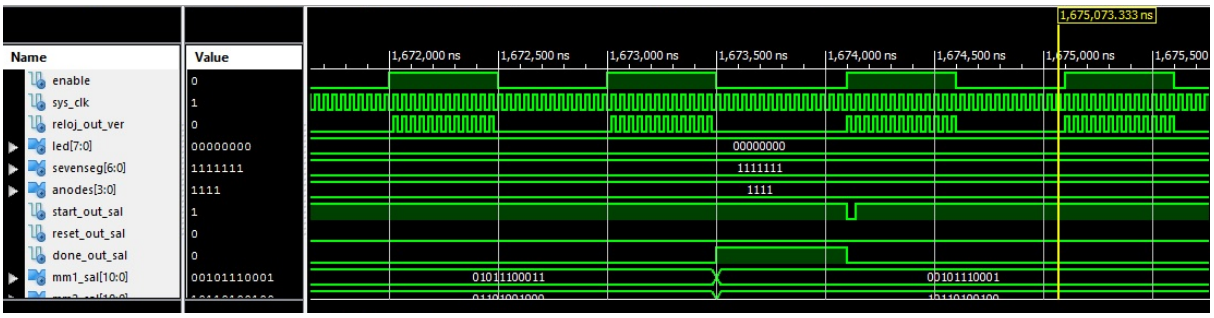

Figura 4.18 Simulación del código con habilitador ó enable

Se puede notar en la simulación que cuando el enable tiene el valor lógico '0' la señal de reloj es cero y está totalmente parado, cuando la señal de enable tiene el valor lógico de '1' el reloj funciona normalmente.

4.3.1.2 Funcionamiento normal del código(25MHz, 12,5MHz y 5MHz) uso de bloque DCM:

En esta sección hacemos una breve descripción de cómo se utilizan los bloques DCM para generar señales a diferentes frecuencias. Vamos a generar una señal de 12,5 MHz agregando un core DCM, se procede igual que como en la sección anterior como muestran la Figura 4.19

Luego cambiamos la frecuencia a 12,5MHz en el área señalada en la siguiente figura:

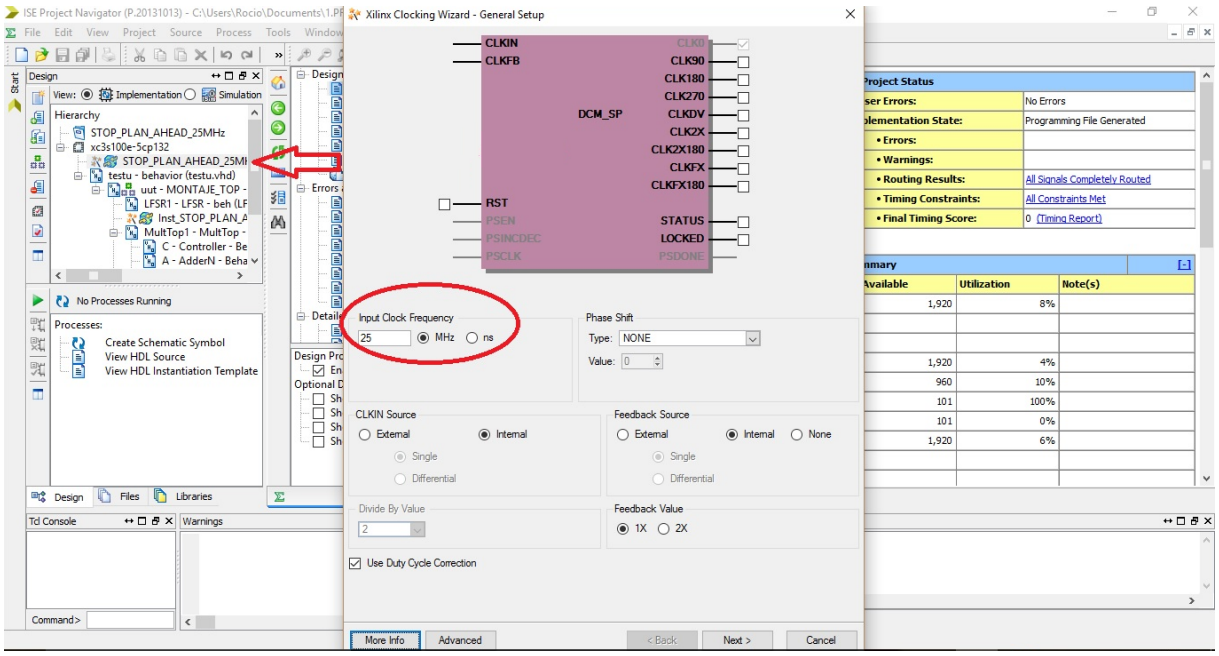

Figura 4.19 Configuración de bloque DCM p/ 12,5MHz.

Es importante el asignar el bloque adicionado DCM como un archivo global del proyecto de lo contrario obtendremos mensajes de error en nuestro reporte de compilación. La definición del bloque DCM como componente y la instanciación del mismo ya se ha explicado en profundidad en la sección anterior, con lo que a continuación mostramos la simulación de un bloque DCM para 12,5MHz, la frecuencia base de la placa es de 50MHz y la frecuencia generada o reloj de salida es 12,5MHz.

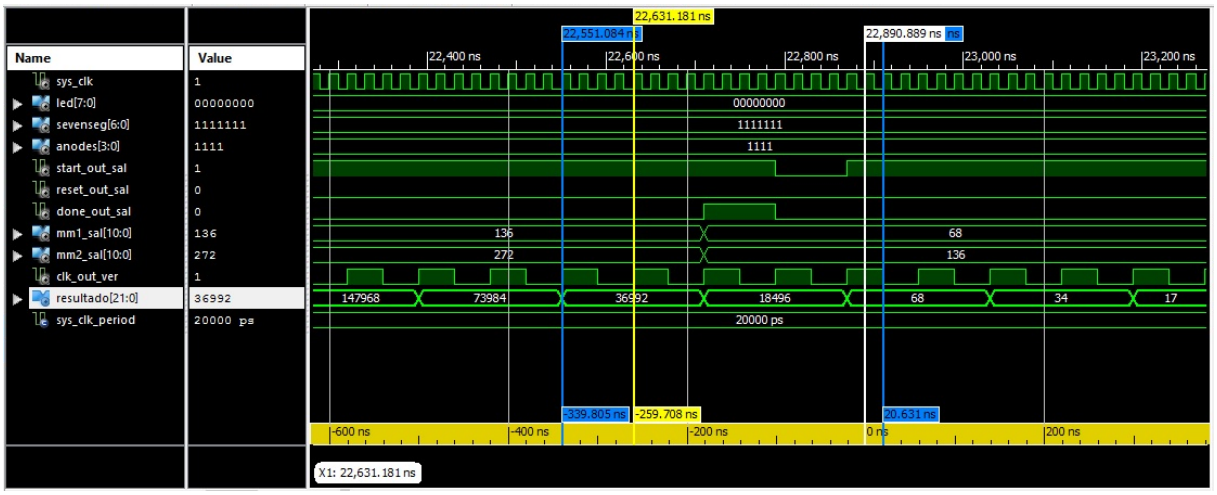

Figura 4.20 DCM de 12,5MHz de reloj de salida.

Se pueden distinguir gracias a los “markers” de la herramienta de simulación ISIM los periodos de ambas señales de reloj, la original que es de 50MHz pues con periodo de 20ns y la señal generada o reloj de salida al DCM de 12,5MHz con periodo 80ns.

Se han realizado 33 simulaciones con DCM para las frecuencias de 25MHz, 12,5MHz y 5 MHz y 11 simulaciones sin DCM utilizando el Jumper JP4 para por hardware tener un reloj de trabajo de 25MHz, lo que da un total de 44 simulaciones en la herramienta ISIM, también generamos 44 reportes de consumo de potencia con Xpower Analyzer que es la herramienta de la que pasamos a hablar en el siguiente capítulo.

Capítulo 5: Análisis del consumo de potencia con la herramienta XPower Analyzer (XPA)

5.1 Introducción

ISE Desing Suite es la herramienta que se utiliza en este proyecto, esta herramienta permite diseñar circuitos y pertenece a la empresa XILINX creadora de las FPGAs.
Es básicamente una interfaz que permite la implementación de circuitos tanto por lenguajes de descripción de hardware como por utilización de puertas lógicas en un esquema gráfico. La versión utilizada en este proyecto es la ISE 14.7.
ISE de XILINX tiene varias herramientas adicionales que proporcionan a este programa una capacidad de trabajo muy elevada. Las dos herramientas utilizadas en este proyecto con ISIM que es la herramienta de simulación y XPower Analyzer que es la herramienta para estimar el consumo de potencia.
Para la creación de un nuevo diseño el primer paso es crear un nuevo proyecto en la interfaz gráfica y seleccionar el modelo de la placa en el que se implementaría el diseño. El paso siguiente es crear un archivo *.vhd y escribir el código en lenguaje de descripción de hardware VHDL.
Una vez que ya tenemos el código completo, se añade un archivo *.ucf, con las conexiones de las entradas y salidas a la del circuito a los pines correspondientes de la FPGA y las restricciones de reloj. En este archivo *.ucf se indican que pines van a cada puerto.

5.2 Fases de un proyecto en ISE

A continuación presentamos las diferentes fases por las que pasa el proyecto, en la figura siguiente serían los puntos verdes.

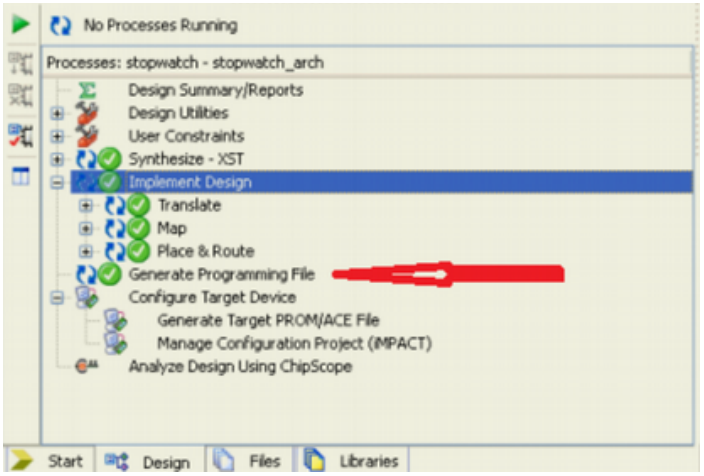

Figura 5.1 Fases de procesos.

La primera fase es para sintetizar el diseño y detecta errores en el código del circuito. Genera un reporte que muestra el tamaño que ocupará el circuito en la placa de manera generalizada.

La siguiente fase de ejecución es la implementación del diseño, que consta de tres sub-fases, traducción del circuito, mapeado del circuito y asignación de puertos y pistas del circuito. También en esta fase se detectan errores y se obtiene un resultado muy preciso de cómo se implementaran las interconexiones en la placa y el valor exacto de la ocupación de la lógica.

La última fase es la de generar el archivo *.bit que contiene todos los datos de cómo implementar el diseño en la placa.
Siempre y cuando tengamos un circuito sin errores, sea cual sea su tarea procedemos a la simulación, para eso utilizamos la herramienta ISIM incluida en el ISE.
Para realizar la simulación necesitaremos un test-bench, o banco de pruebas, que no es otra cosa que un fichero en el que especificamos la frecuencia de trabajo o restricciones de tiempo, y los

datos que debe tener en cuenta ISIM a la hora de correr la simulación estos son cambios de señales de entrada y tiempos de espera.
Si contamos con resultados de la simulación coherentes, significa que el circuito cumple con las tareas preestablecidas, si no los valores de la simulación no son correctos habrá que revisar el código en busca de de un posible error y repetir los fases anteriormente descritas.
Se recomiendan guardar los resultados de la simulación cuando todo funciona correctamente si después se pretende utilizar la herramienta de estimación de consumo de potencias, ya que es necesario tener actividad en el circuito para poder medir potencia dinámica.
El archivo en donde se guarda la actividad de los nodos durante la simulación es el *.saif, para generar este archivo se debe introducir en la consola de comandos las siguientes instrucciones como sigue:

- ***restart***
- ***saif open -file nombre_archivo.saif -allnets***
- ***run 1000ns***
- ***saif close***

La primera instrucción restablece la simulación, la segunda instrucción crea el archivo correspondiente, la opción allnets que incluye también los nodos internos del circuito, la tercera ejecuta la simulación en el tiempo indicado y va guardando los resultados en el archivo *.saif, y la última cierra el archivo. Es recomendable correr la simulación por un periodo extendido de tiempo para obtener resultados más precisos.

Contando ya con el archivo *.bit generado y los archivos generados en las diferentes fases de ejecución más el *.saif de la fase de simulación podemos generar un reporte de consumo de potencia en la herramienta XPower Analyzer.

XILINX tiene tres herramientas para la estimación de potencias para un diseño, Plan Ahead, XPower Estimator y XPower Analyzer(XPA). Las dos primeras herramientas son más simplificadas que XPA y se usan en fases del desarrollo cuando no se cuenta con código final del circuito, Es por eso que decidimos utilizar XPower Analyzer(XPA) al ser esta una herramienta más precisa y por tanto la que se utilizará en este proyecto.

XPower Analyzer entrega un reporte de consumo así como una estimación de temperaturas en la placa, a continuación la ventana del XPower Analyzer.

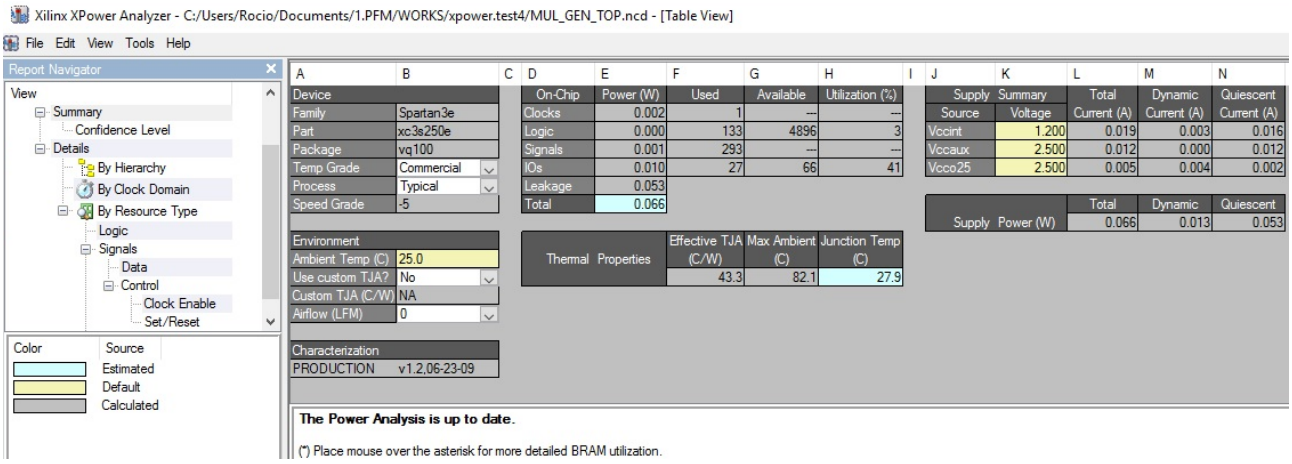

Figura 5.2 Captura de pantalla del reporte generado por XPower Analyzer.

Se puede observar en estas tablas los datos de la placa y las condiciones de ambiente y de trabajo, estas también se pueden fijar de antemano, en la tabla a continuación los datos de potencia

consumida detallada en función del tipo de potencia y las propiedades de temperatura. En cada pestaña podemos desplegar todos los datos de forma exhaustiva.

Es de suma importancia mencionar que la herramienta XPA calcula potencias medias y no instantáneas y permite guardar el reporte de consumo de potencia extendido en formato *.pwr que se puede visualizar mediante cualquier editor de texto. Para ellos debemos seleccionar en la opción "Generate Advanced Report" en el menú "Tools". El archivo que se genera se muestra a continuación, fijarse que en el reporte de la Figura 5.2 que la potencia esta en Wattios lo que imposibilita ver en algunas señales que consumen muy poco si se consume algo de potencia, por este motivo es mejor generar el reporte avanzado donde tenemos el consumo de potencia en milliWatios lo que nos permite ver el pequeño consumo de algunas señales referidas a la lógica del circuito, en la siguiente figura vemos que las señales de la lógica del circuito tienen un consumo de 0,46 mWatts.

```
|                          On-Chip Power Summary
|
-------------------------------------------------------------------------------
-
|       On-Chip        | Power (mW) |  Used  | Available | Utilization (%)
|
-------------------------------------------------------------------------------
-
| Clocks               |       2.37 |      1 |    ---    |       ---
|
| Logic                |       0.46 |    133 |      4896 |               3
|
| Signals              |       0.60 |    293 |    ---    |       ---
|
| IOs                  |       9.90 |     27 |        66 |              41
|
| Static Power         |      52.55 |        |           |
|
| Total                |      65.88 |        |           |
|
-------------------------------------------------------------------------------
-

2.2.  Thermal Summary
------------------------------
|      Thermal Summary       |
------------------------------
| Effective TJA (C/W) | 43.3 |
| Max Ambient (C)     | 82.1 |
| Junction Temp (C)   | 27.9 |
------------------------------
```

Figura5.3 Reporte extendido de XPower Analyzer.

La herramienta XPA proporcionará datos exclusivos de la FPGA obviando todos los periféricos además de desglosar la potencia en los distintos componentes de la FPGA, los datos proporcionados de potencia se pueden dividir en los siguientes Apartados:

- **Clock** o relojes.
- **Logic**, que sería la lógica.
- **Signals**, datos de señales en el diseño.
- **IOs**, entradas y salidas.
- **Static Power**, que sería la potencia estática.

La potencia debida al Bloque de RAMs o BRAMs es parte de la potencia estática. Estos apartados pueden variar de acuerdo a la versión del ISE y la actualización de XPA que se este utilizando. En versiones anteriores la potencia de ***BRAMs*** mas ***Quiescent*** aparecen separadas y se corresponderían a la potencia estática, que en las últimas versiones aparece como ***Static Power*** en un solo Apartado.

La suma de los primeros apartados Clock, Logic, signal e IOs se corresponde a la potencia dinámica.

Se necesita una simulación por prueba para poder generar un reporte, y guardar el resultado. Cada condición de trabajo del circuito consistió en variar las entradas y salidas del sistema hasta el limite de capacidad de la placa BASYS2 a diferentes frecuencias.

5.3 Archivos necesarios para generar el reporte de XPA

La herramienta XPower Analyzer (XPA) sirve para hacer un análisis de los datos de un diseño real. XPA utiliza después de la implementación del diseño en ISE, el archivo de salida NCD generado desde la fase de Place & Route (PAR).

XPA ahora cuenta con un algoritmo de estimación llamado ***vectorless***, una manera de asignar las tasas de actividad de los nodos, incluso si estas tasas de actividad no están definidas en el archivo de diseño o especificadas de ninguna otra manera. Sin embargo, se recomienda utilizar los archivos de actividad en la simulación (SAIF o VCD) para un análisis más preciso del consumo de potencia.

Es recomendable utilizar siempre la versión más reciente de ISE para obtener la última versión del XPA, que contiene los últimos datos de caracterización para el análisis referente al consumo de potencia.

XPA se utiliza para el análisis del diseño en términos de consumo de potencia.

En el punto de la fase del diseño en el que XPA es utilizado, su proyecto ISE debería haber completado con éxito las fases de Place & Route (PAR), y haber generado un archivo de salida NCD.

Es importante entender qué archivos de diseño son utilizados por la herramienta XPA para la estimación del consumo de potencia. XPA se puede iniciar como interfaz gráfica de usuario desde el navegador de proyecto del ISE o por medio de introducir en la línea de comando XPA, para generar un informe extendido de potencia en formato texto, introduzca xpwr.

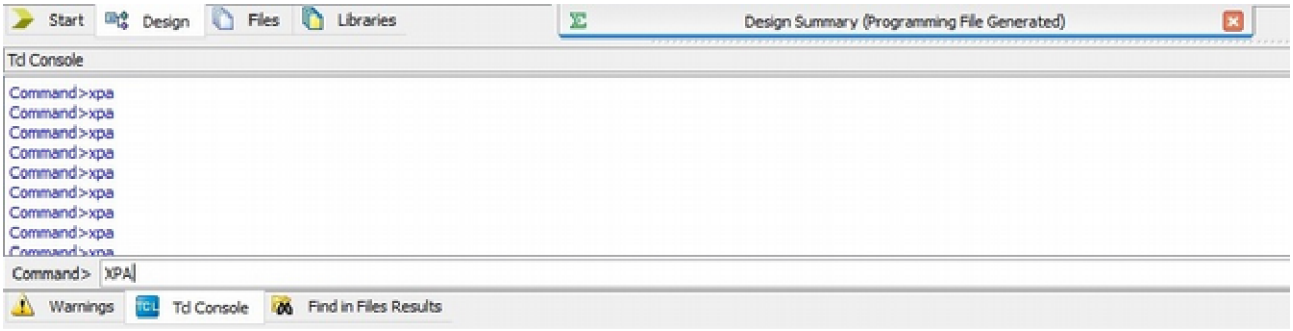

Figura 5.4 Comando XPA para iniciar la interfaz gráfica.

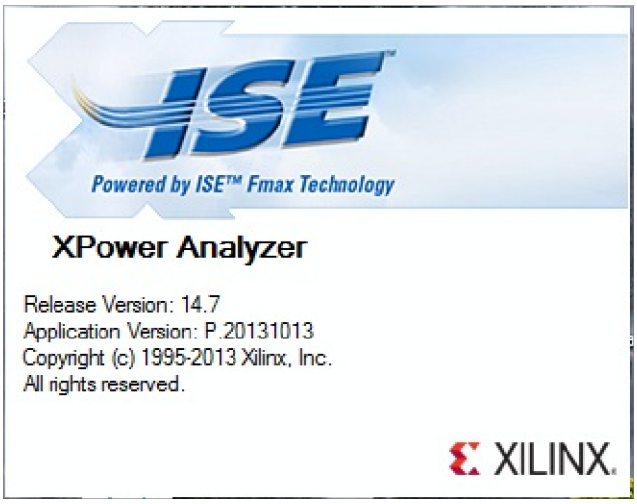

Figura 5.5 Aspecto del inicio de la interfaz gráfica de usuario del XPower Analyzer para ISE14.7

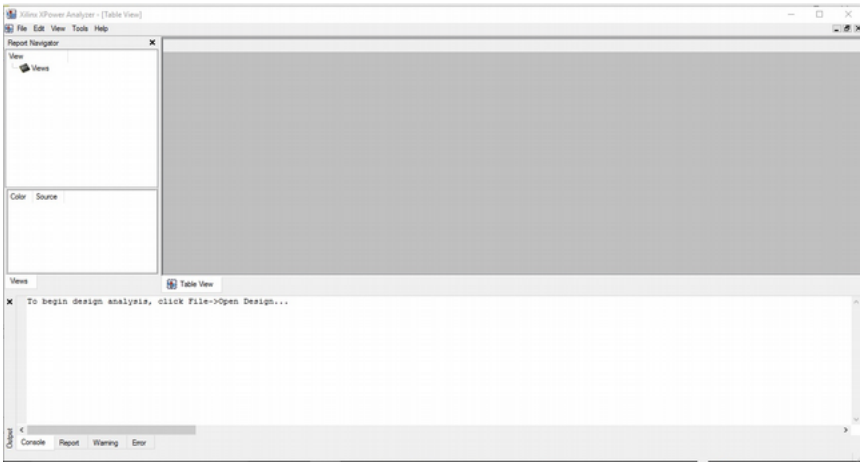

Figura 5.6 Interfaz gráfica del XPower Analyzer.

En el ISE el XPower se puede encontrar en la pestaña de procesos debajo de **Place & Route.**

Para mejorar la precisión de XPA también se puede leer en los archivos de diseño opcionales. Por ejemplo, desde el Navegador de proyectos, puede hacer lo siguiente:

1. En la pestaña de procesos debajo de **Place & Route**, en **Analyze Power Distribution (XPower Analyzer)** podemos seleccionar propiedades para el proceso.

2. En el cuadro de dialogo **propiedades** del **XPower Analyzer**, especifique los archivos. Por otra parte, en la herramienta independiente tendrá la opción de incluir estos archivos cuando se carga el diseño. Especificar los archivos al incluirlos en el cuadro de diálogo **Open Design**.

Los archivos son:

• **NCD** – Archivo de salida resultante de la fase de **Place & Route** este es requerido por XPA para el análisis de potencia.

• **XML** - Archivo de configuración – **Archivo opcional**. El archivo de configuración contiene parámetros de la aplicación y tasas de actividad de los nodos, guardados desde una sesión de XPower Analyzer, cualquier cambio manual (ediciones), que se han hecho en la ventana de XPA pueden ser guardadas en este archivo de configuraciones. El archivo de configuraciones (en formato XML) puede ser utilizado para cargar cualquier información de diseño relevante de una sesión de XPA previa a una sesión posterior.

• **PCF** (Physical Constraints File) - **Archivo opcional**. Contiene información sobre restricciones que ayudan a determinar la frecuencia de reloj. Toda la información acerca del archivo UCF (User Constraints File) es reportada en el archivo PCF, así si el diseño tiene correctas las restricciones al cargar el PCF se obtienen resultados más precisos de potencia.

• **VCD** ó **SAIF** - **Archivo opcional**. Estos archivos de actividad en la simulación incluyen información específica de conmutación (tasas sobre interruptores, tasas de señal e información de frecuencia) la que da la más precisa estimación en términos de potencia. Hay que tener en cuenta que no siempre XPA puede ser capaz de emparejar todas las señales de diseño con las señales en el archivo de simulación, por lo que se debe de introducir alguna información de conmutación a mano, esto es importante para asegurar que observamos cual información sobre el diseño fué extraída y cual no fue extraída. Se aconseja tomar nota de los mensajes de advertencia en la consola ya que pueden indicar que la información sobre el diseño no fue extraída correctamente para utilidad del XPA.

Archivos de entrada: En la interfaz gráfica independiente es necesario especificar los archivos de entrada de esta manera:

1. En la ventana de XPA seleccionar File > Open Design.

2. En el cuadro de dialogo de **Open Design** seleccionar los archivos y fijar las opciones de utilización para el dispositivo.

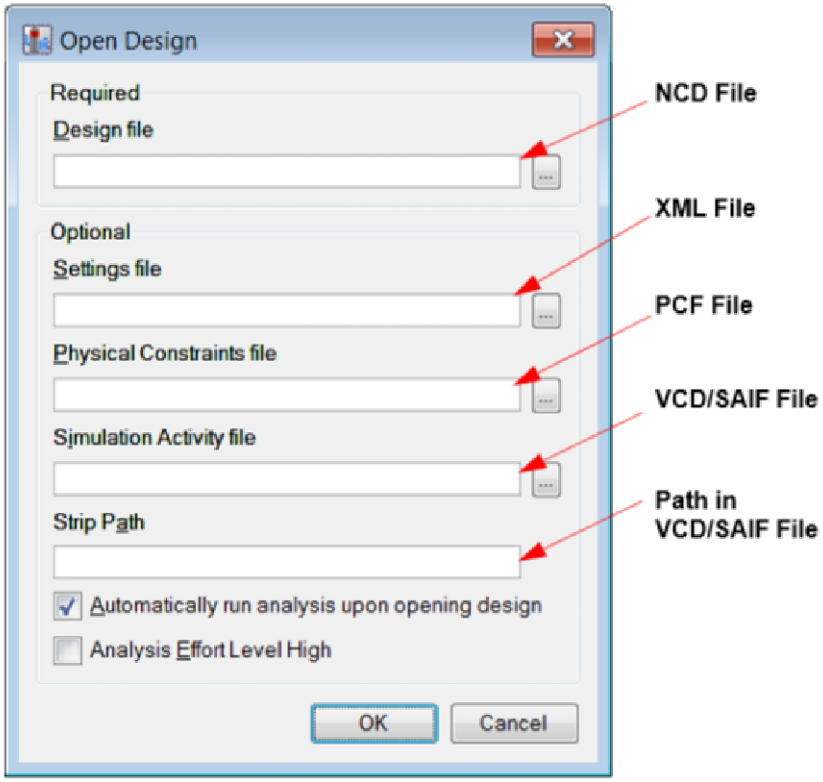

Figura 5.7 Cuadro de dialogo de abrir diseño en (XPA).

5.4 Restricciones temporales (Timing Constraint)

Para que un reporte de Xpower Analyzer sea preciso es importante el crear mediante la herramienta de User Constraints en Create timing constraint el archivo *.ucf para aplicar restricciones temporales.

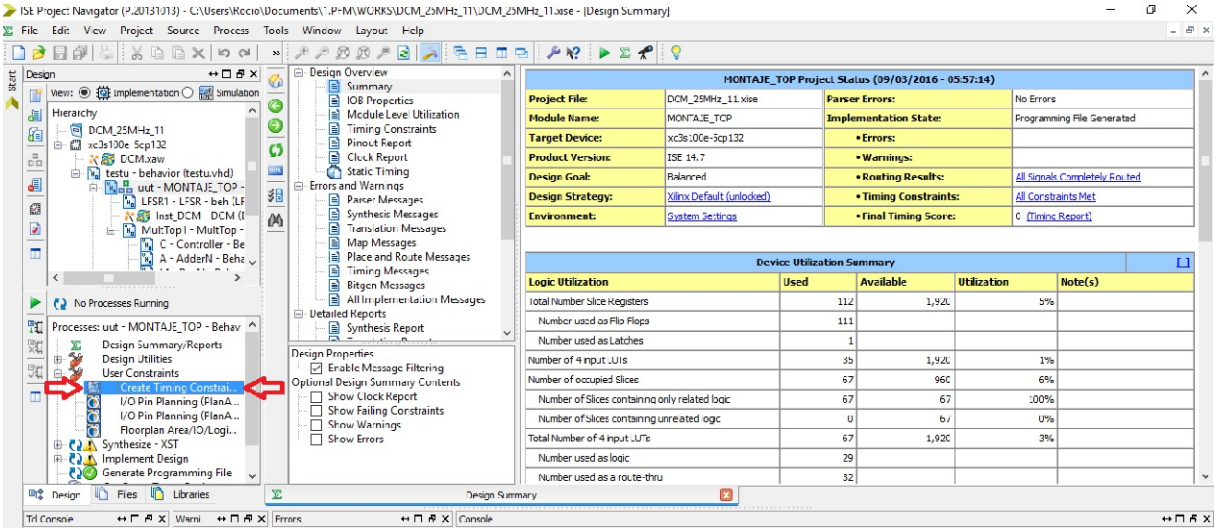

Figura 5.8 Herramienta ***User Constraints.***

Las *timing constraints* o restricciones temporales son las instrucciones que el diseñador da a la herramienta XILINX sobre la velocidad a la que el diseñador quiere correr el diseño. La herramienta XILINX utiliza estas instrucciones para construir una aplicación que cumpla con las limitaciones de

tiempo. Estas restricciones temporales se especifican en el archivo de restricciones de usuario *.ucf, este es el mismo archivo que se utiliza para especificar las restricciones en cuanto a la ubicación de pines. En esta sección hablaremos de dos restricciones importantes para un buen reporte en XPower Analyzer, las cuales son de período de restricción de reloj (periodo) y la restricción de pista/ruta falsa (TIG). La primera es una restricción en el período de reloj. Esta restricción indica a la herramienta la frecuencia a la cual desea ejecutar el diseño.

La herramienta trata de asegurar que todas las rutas combinadas entre registros tengan retardos más cortos que el periodo de reloj para que el diseño funcione de forma fiable para la frecuencia de trabajo dada.

5.4.1 Restricción de reloj

La sintaxis de esta restricción para trabajar a una frecuencia especifica es:

NET "<net_name>" PERIOD = <clock_period> ns HIGH <duty_cycle>%;

Donde net_name es el nombre de la señal de clock (e.g., clk, sys_clk, or Clk_Port como en el diseño), clock_period es el periodo de tiempo de la señal de clock y duty_Cycle es el ciclo de reloj.

Así si queremos especificar un periodo de reloj de 40ns(25MHz) con 50% de ciclo de reloj para la señal “sys_clk” se puede agregar la restricción al archivo.ucf escribiendo en el archivo la siguiente expresión

NET "sys_clk" PERIOD = 40.0ns HIGH 50%;

De forma alternativa se puede especificar también:

Net "ClkPort" TNM_NET = sys_clk_pin;

TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 25MHz;

Esto de arriba es por si la introducimos manualmente, pero si utilizamos la herramienta “User constraints” que es lo más recomendable, nos genera la siguiente sintaxis completa:

#Created by Constraints Editor (xc3s100e-cp132-5) – 2016/08/24

NET "Clk" TNM_NET = Clk;

TIMESPEC TS_Clk = PERIOD "Clk" 40 ns HIGH 50%;

OFFSET = IN 40 ns VALID 40 ns BEFORE "Clk" RISING;

OFFSET = OUT 40 ns AFTER "Clk";

A la hora de generar un reporte más preciso de consumo de potencia con XPower Analyzer es mejor utilizar la segunda expresión generada por la herramienta “User constraints” de XILINX, específicamente en la sección “create timing constraints”, se pueden también pasar restricciones manualmente especialmente para asignar entradas/salidas con interruptores/pines respectivamente, se pueden complementar ambas formas tanto el paso de restricciones de la forma manual como la herramienta de creación de restricciones del ISE.

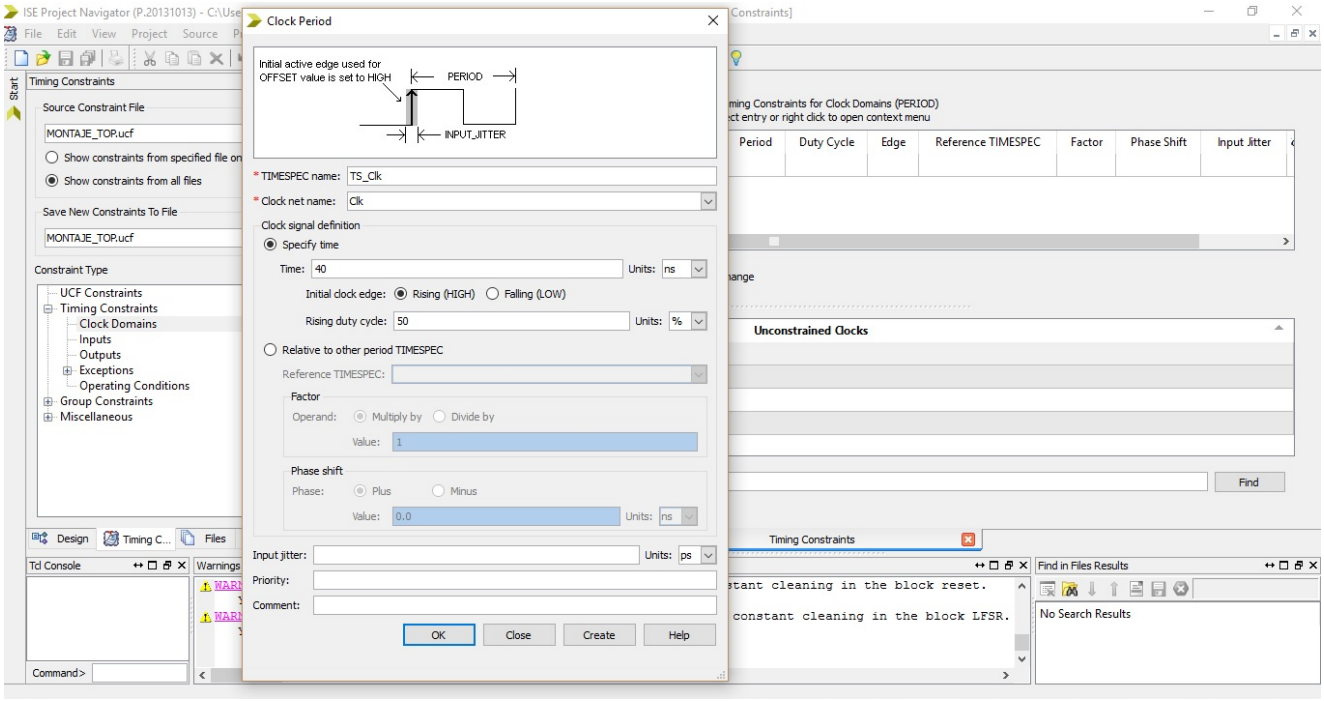

Figura 5.9 Herramienta de creación de restricciones del ISE “create user constraints”, asignación de periodo de reloj.

5.4.2 Restricción TIG

La segunda restricción que utilizamos, indica a la herramienta hacer caso omiso de la señal al determinar la temporización camino. Por ejemplo podemos instruir a la herramienta sobre ignorar una entrada (interruptores o botones) con la sintaxis:

PIN "SW1" TIG;

TIG es sinónimo de Ignorar sincronización de Grupo. Mediante la asignación de una señal a un TIG la herramienta pasa por alto los caminos de temporización que implican esta señal. Se podría declarar todas las demás entradas (interruptores y botones) como perteneciente a un TIG usando la sintaxis anterior.

Si no utilizamos TIG como restricción la otra opción es configurar retardos para esa señal más cortos que el periodo reloj.

A continuación se muestra el archivo *.ucf utilizado para este proyecto:

```
#Created by Constraints Editor (xc3s100e-cp132-5) - 2016/09/01

NET "sys_clk" TNM_NET = sys_clk;

TIMESPEC TS_sys_clk = PERIOD "sys_clk" 40 ns HIGH 50%;

NET "MultTop1/C/state_FSM_FFd51" TNM_NET = MultTop1/C/state_FSM_FFd51;

TIMESPEC TS_MultTop1_C_state_FSM_FFd51 = PERIOD "MultTop1/C/state_FSM_FFd51" 40 ns HIGH 50%;

OFFSET = IN 40 ns VALID 40 ns BEFORE "sys_clk" RISING;

OFFSET = OUT 40 ns AFTER "sys_clk";
```

```
NET "enable" TIG;

NET "enable_IBUF" TIG;

////// Esta parte del archivo UCF es tipeada manualmente

NET "enable" LOC = "N3"; // se asigna el enable al interruptor en pin N3

// Fue necesario el desactivar los LEDs y Display en la placa todas las asignaciones a //continuación sirven para ello.

# Pin anodes

NET "anodes<3>" LOC = "K14";

NET "anodes<2>" LOC = "M13";

NET "anodes<1>" LOC = "J12";

NET "anodes<0>" LOC = "F12";

# Pin sevenseg

NET "sevenseg<0>" LOC = "M12";

NET "sevenseg<1>" LOC = "L13";

NET "sevenseg<2>" LOC = "P12";

NET "sevenseg<3>" LOC = "N11";

NET "sevenseg<4>" LOC = "N14";

NET "sevenseg<5>" LOC = "H12";

NET "sevenseg<6>" LOC = "L14";

# Pin assignment for LEDs

NET "Led<7>" LOC = "G1";

NET "Led<6>" LOC = "P4";

NET "Led<5>" LOC = "N4";

NET "Led<4>" LOC = "N5";

NET "Led<3>" LOC = "P6";

NET "Led<2>" LOC = "P7";

NET "Led<1>" LOC = "M11";

NET "Led<0>" LOC = "M5";
```

En el capítulo de Metodología se explica detalladamente como utilizar un bloque DCM y como configurar el registro(buffer) BUFGCE para el reloj del sistema habilitándolo o no con una entrada enable asignada a un interruptor(el pin N3 asignado en el archivo *.ucf anterior), todo esto es importante a la hora de realizar los ensayos a reloj parado para medir la potencia estática del circuito.

Capítulo 6: RESULTADOS Y CONCLUSIONES

6.1 Introducción:

Se realizan cuarenta y cuatro pruebas físicas para compararlas con los cuarenta y cuatro resultados de estimación de potencia del XPower Analyzer, se realizan pruebas para frecuencias de reloj las cuales son 25MHz(sin DCM), 25MHz(con DCM), y las frecuencias de 5MHz, 12,5MHz y 25MHz(con DCM). Para cada frecuencia se varían las entradas y salidas del código del siguiente modo, para una entrada de 2 bits la salida tendrá el doble de bits o sea 4 bits, para una entrada de 3 bits la salida tendrá 6 bits, para una entrada de 4 bits la salida tendrá 8 bits, y así sucesivamente, con lo que las entradas varían en el rango desde 1 bit hasta 11 bits, y las salidas correspondientes desde 2 bits hasta 22 bits, así el número de simulaciones por frecuencia es de once.

6.2 Resultados de pruebas físicas y con XPA individualmente para cada frecuencia:

Para analizar los resultados se hacen gráficas de dispersión y se calcula la correlación y la covarianza de las variables las cuales son Potencia total física medida, potencia estática física medida, Potencia total estimada con XPA y Potencia estática estimada por XPA. Los resultados de las pruebas físicas y con XPA se muestran en las tablas siguientes:

Pruebas25 MHz (Sin DCM)	P. física P.Total	P. física Estática	Xpower P.Total	Xpower Estática
Entrada(1 bit)/Salida(2 bits)	64,3	59,1	36,48	33,69
Entrada(2bits)/Salida(4bits)	68	62,244	37,58	33,70
Entrada(3bits)/Salida(6bits)	69,03	61,9619	38,69	33,71
Entrada(4bits)/Salida(8bits)	72,2019	62,1834	39,27	33,71
Entrada(5bits)/Salida(10bits)	73,1003	61,319	40,13	33,72
Entrada(6bits)/Salida(12bits)	74,03519	61,5	41,10	33,73
Entrada(7bits)/Salida(14bits)	77,2549	61,9289	42,30	33,74
Entrada(8bits)/Salida(16bits)	78,2299	62,2359	42,67	33,75
Entrada(9bits)/Salida(18bits)	81,6784	63,4	43,61	33,75
Entrada(10bits)/Salida(20bits)	81,5791	62,2939	44,70	33,77
Entrada(11bits)/Salida(22bits)	83,3	62,3745	45,42	33,77

Tabla 6.1 Resultados de consumo de potencia en mWatts para 25 MHz sin DCM usando el Jumper JP4.

Se van a comparar primeramente las potencias medidas físicamente con las estimadas con Xpower Analizer de las frecuencias de 25 MHz sin DCM y con DCM, luego en la última parte se van a comparar las potencias a diferentes frecuencias con DCM.

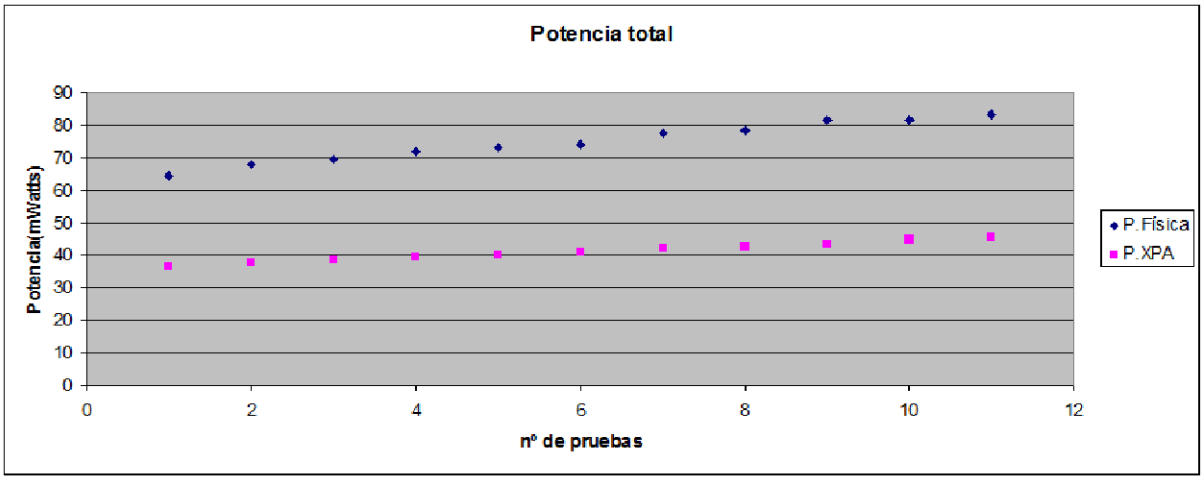

Figura 6.1 Comparación entre medidas físicas y estimaciones de la potencia total con la herramienta XPA(mWatts).

Correlación de ambas variables:

	Columna 1	*Columna 2*
Columna 1	1	
Columna 2	0,99083372	1

Existe una relación fuerte y directa entre las 2 variables.

Covarianza:

	Columna 1	*Columna 2*
Columna 1	34,56657	
Columna 2	16,256112	7,78709587

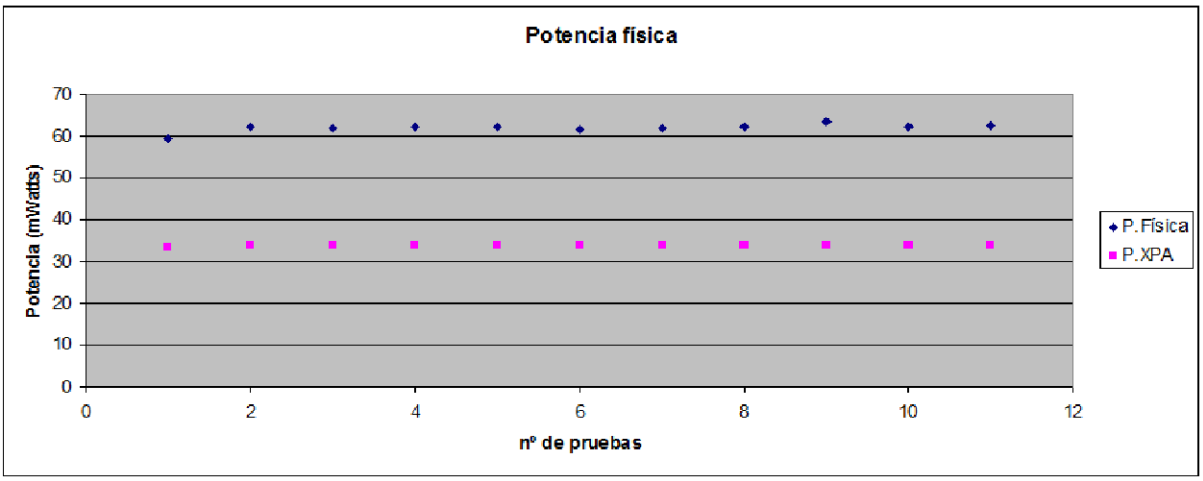

Figura 6.2 Comparación entre potencia estática medida y estimaciones de la potencia estática con la herramienta XPA.

Correlación de ambas variables:

	Columna 1	*Columna 2*
Columna 1	1	
Columna 2	0,56130683	1

También existe una relación fuerte y directa entre estas variables.

Covarianza:

	Columna 1	*Columna 2*
Columna 1	1,00769132	
Columna 2	0,01470398	0,00068099

Pruebas 25 MHz (con DCM)	P. física P.Total	P. física Estatica	Xpower P.Total	Xpower Estatica
Entrada(1 bit)/Salida(2 bits)	100,8134	61,338	56,71	33,88
Entrada(2bits)/Salida(4bits)	105,9914	61,39	57,79	33,89
Entrada(3bits)/Salida(6bits)	108,5089	61,424	58,87	33,9
Entrada(4bits)/Salida(8bits)	108,9604	61,487	59,33	33,91
Entrada(5bits)/Salida(10bits)	111,9989	61,489	60,27	33,91
Entrada(6bits)/Salida(12bits)	113,9119	61,52	61,25	33,92
Entrada(7bits)/Salida(14bits)	115,322	61,627	62,26	33,93
Entrada(8bits)/Salida(16bits)	115,9918	61,66	62,65	33,94
Entrada(9bits)/Salida(18bits)	118,879	61,685	63,72	33,95
Entrada(10bits)/Salida(20bits)	119,9175	62,736	64,55	33,96
Entrada(11bits)/Salida(22bits)	123,1752	62,727	65,50	33,96

Tabla 6.2 Resultados de consumo de potencia a 25MHz con DCM en mWatts.

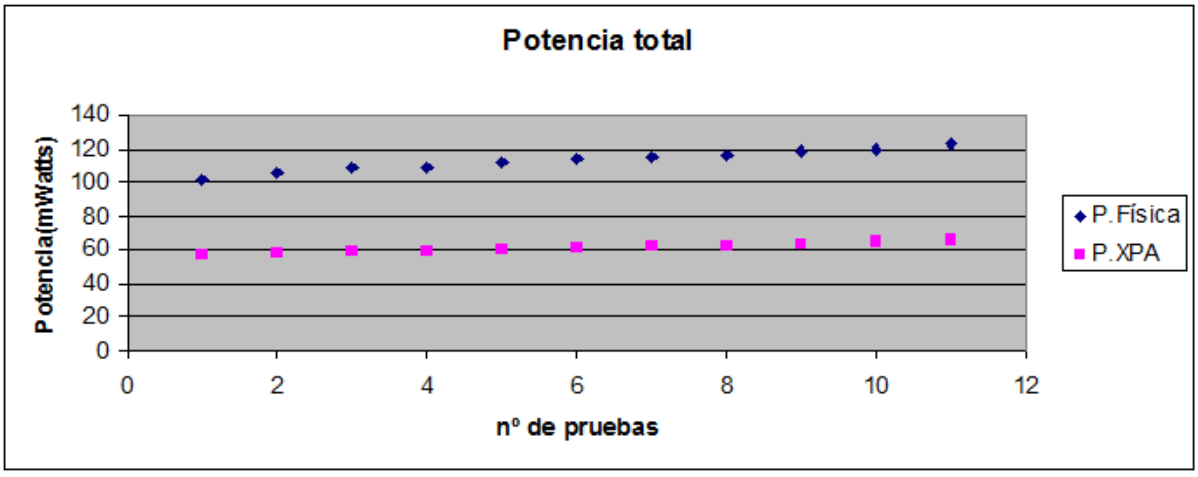

Figura 6.3 Gráfica de comparación de Potencia Total en mWatts

Se gráfica la potencia total física medida y la potencia total estimada con XPA.

Correlación:

	Columna 1	*Columna 2*
Columna 1	1	
Columna 2	0,9911705	1

Covarianza:

	Columna 1	*Columna 2*
Columna 1	39,6868447	
Columna 2	16,9296561	7,35111074

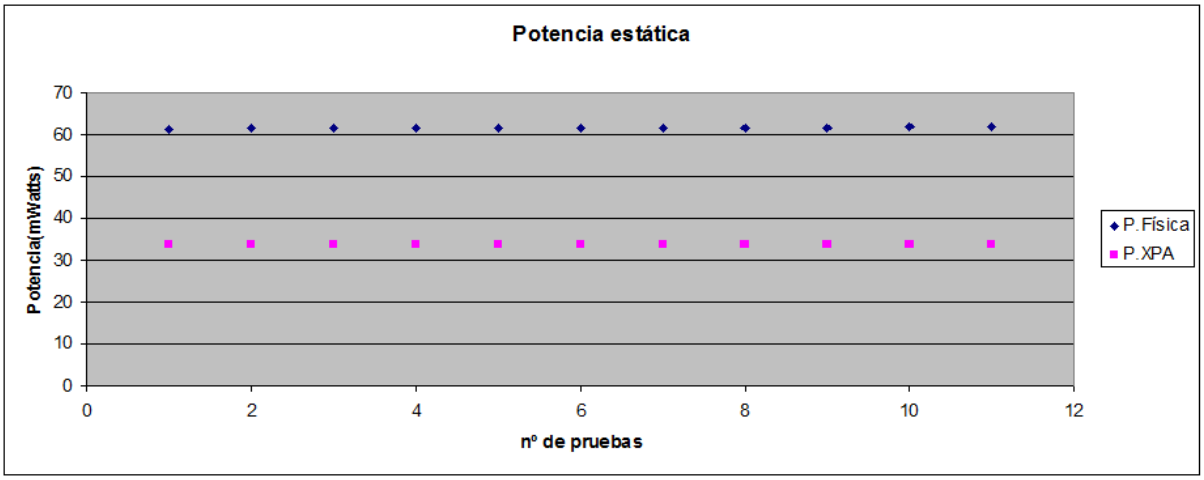

Figura 6.4 Gráfico de la potencia estática en mWatts.

Se compara la potencia estática medida físicamente y la estimada con XPA.

Correlación:

	Columna 1	*Columna 2*
Columna 1	1	
Columna 2	0,99332176	1

Covarianza:

	Columna 1	*Columna 2*
Columna 1	0,01791727	
Columna 2	0,00349909	0,00069256

Pruebas12,5 MHz (con DCM)	P. física P.Total	P. física Estatica	Xpower P.Total	Xpower Estatica
Entrada(1 bit)/Salida(2 bits)	93,6007	61,186	52,36	33,84
Entrada(2bits)/Salida(4bits)	96,0301	61,240	52,90	33,84
Entrada(3bits)/Salida(6bits)	96,8215	61,3011	53,40	33,85
Entrada(4bits)/Salida(8bits)	99,0077	61,30	53,52	33,85
Entrada(5bits)/Salida(10bits)	99,3331	61,338	53,91	33,85
Entrada(6bits)/Salida(12bits)	97,600	61,39	54,33	33,86
Entrada(7bits)/Salida(14bits)	100,9112	61,45	54,74	33,86
Entrada(8bits)/Salida(16bits)	97,9	61,485	54,66	33,86
Entrada(9bits)/Salida(18bits)	101,012	61,51	55,12	33,86
Entrada(10bits)/Salida(20bits)	101,600	61,50	55,42	33,86
Entrada(11bits)/Salida(22bits)	103,225	61,498	55,80	33,87

Tabla 6.3 Resultados de consumo de potencia para 12,5 MHz con DCM en mWatts.

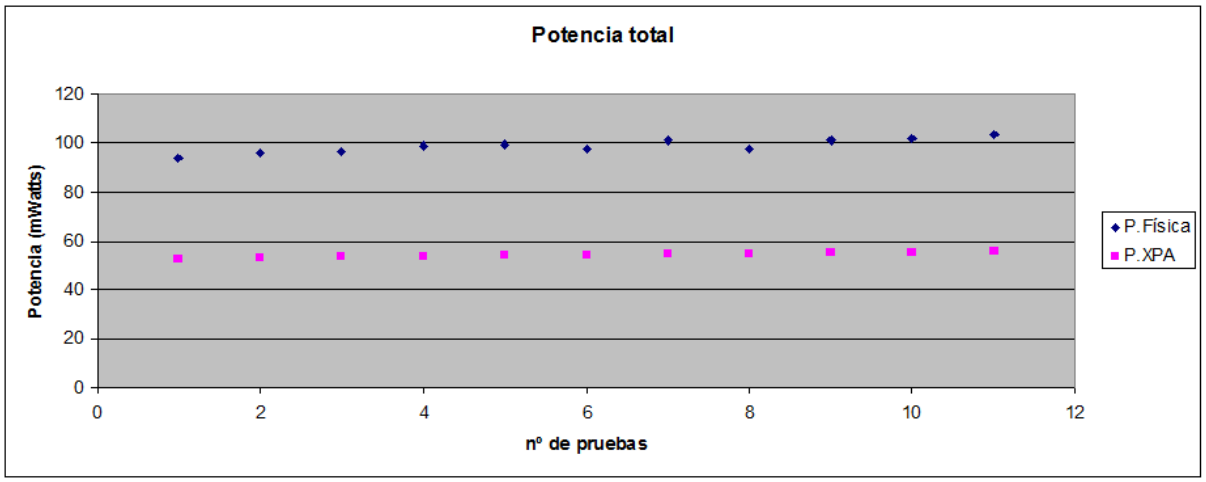

Figura 6.5 Potencia total p/ 12,5MHz en mWatts.

Se compara la potencia total medida físicamente y la potencia total estimada con XPA.

Correlación:

	Columna 1	*Columna 2*
Columna 1	1	
Columna 2	0,963177	1

Covarianza:

	Columna 1	*Columna 2*
Columna 1	4,36768247	
Columna 2	2,07394972	1,06153223

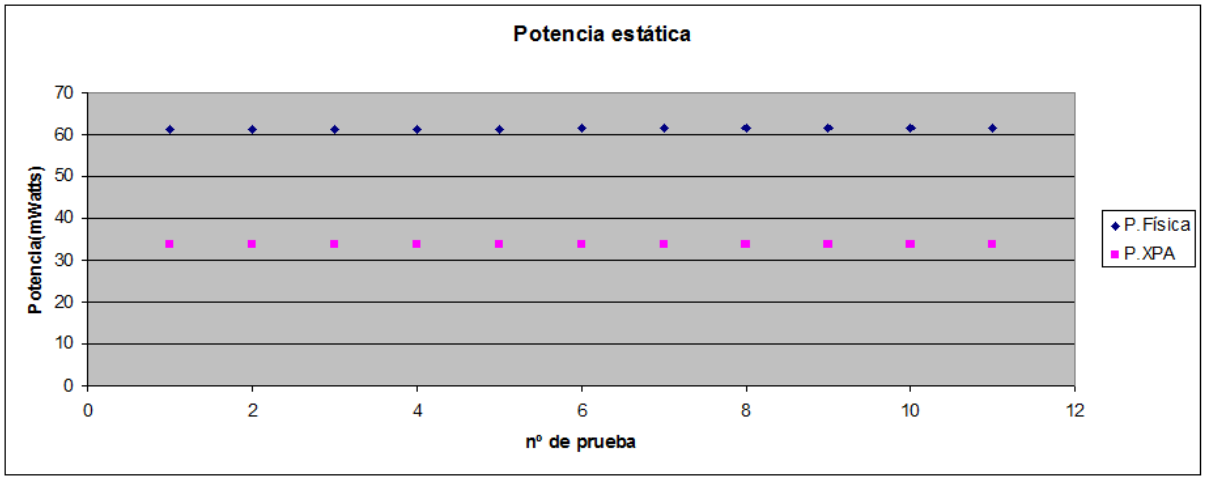

Figura 6.6 Potencia estática en mWatts.

Se compara la potencia estática medida físicamente con la potencia estática estimada con XPA.

Correlación:

	Columna 1	*Columna 2*
Columna 1	1	
Columna 2	0,79616217	1

Covarianza:

	Columna 1	*Columna 2*
Columna 1	0,02656721	
Columna 2	0,00115589	7,9339E-05

Pruebas 5 MHz (con DCM)	P. física P.Total	P. física Estática	Xpower P.Total	Xpower Estática
Entrada(1 bit)/Salida(2 bits)	88,0925	61,143	51,65	33,83
Entrada(2bits)/Salida(4bits)	92,616	61,1859	51,99	33,83
Entrada(3bits)/Salida(6bits)	92,985	61,2490	52,10	33,84
Entrada(4bits)/Salida(8bits)	93,1080	61,251	51,14	33,84
Entrada(5bits)/Salida(10bits)	94,6009	61,2940	51,19	33,84
Entrada(6bits)/Salida(12bits)	95,839	61,360	52,24	33,84
Entrada(7bits)/Salida(14bits)	97,119	61,411	52,50	33,84
Entrada(8bits)/Salida(16bits)	97,195	61,454	52,70	33,84
Entrada(9bits)/Salida(18bits)	97,2165	61,467	53,09	33,85
Entrada(10bits)/Salida(20bits)	98,046	61,45	53,16	33,85
Entrada(11bits)/Salida(22bits)	101,600	61,446	53,34	33,85

Tabla 6.4 Resultados de consumo de potencia para 5 MHz con DCM en mWatts.

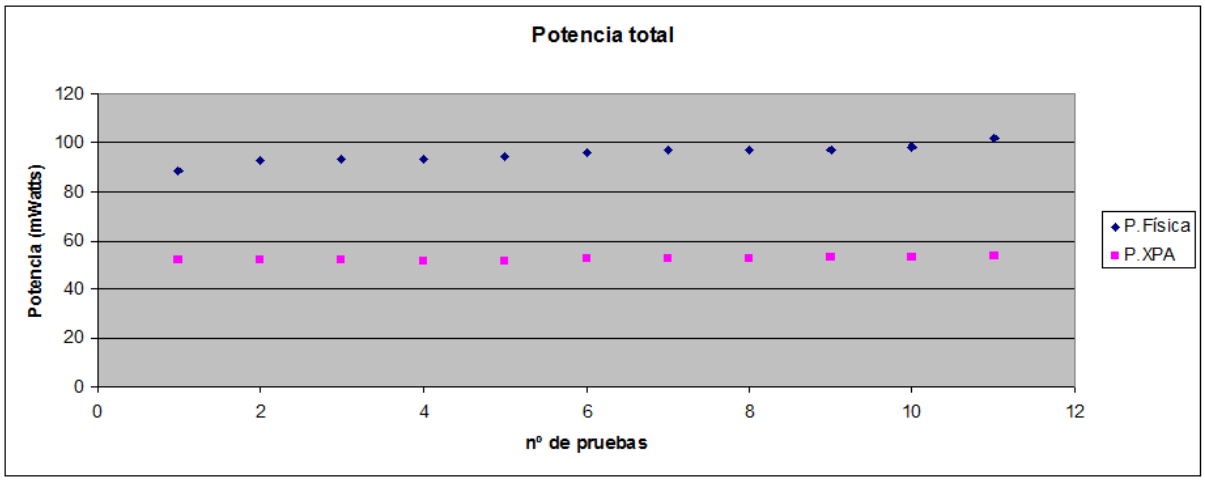

Figura 6.7 Gráfica comparativa de la Potencia total en mWatts.

Se compara la potencia total medida físicamente con la estimada con XPA.

Correlación:

	Columna 1	*Columna 2*
Columna 1	1	
Columna 2	0,82922918	1

Covarianza:

	Columna 1	*Columna 2*
Columna 1	5,78332811	
Columna 2	1,44573869	0,52559669

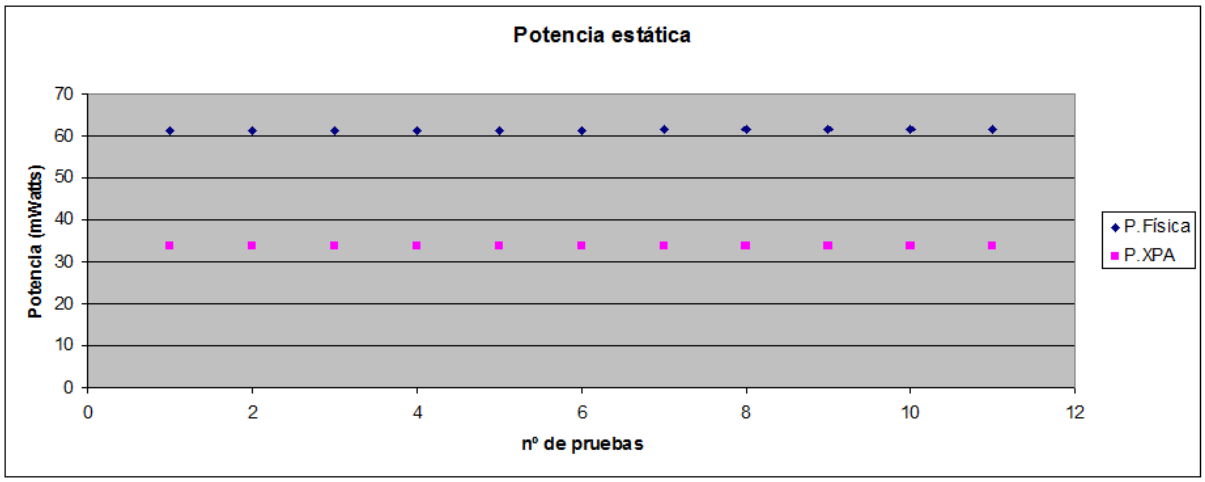

Figura 6.8 Gráfica de la Potencia estática en mWatts.

Se compara la potencia estática medida físicamente con la estimada con XPA.

Correlación:

	Columna 1	*Columna 2*
Columna 1	1	
Columna 2	0,51995925	1

Covarianza:

	Columna 1	*Columna 2*
Columna 1	0,18870522	
Columna 2	0,00150892	4,4628E-05

6.3 Comparación a diferentes frecuencias con DCM:

	25MHz	12.5MHz	5MHz
nº de pruebas	**P. física P.Total**	**P. física P.Total**	**P. física P.Total**
1	100,8	93,6	88
2	105,9914	96	92,616
3	108,5089	96,8	92,985
4	108,9604	99,0077	93,108
5	111,9989	99,3331	94,6009
6	113,9119	97,6	95,839
7	115,322	100,9112	97,119
8	115,9918	97,6	97,195
9	118,879	101,012	97,2165
10	119,9175	101,6	98,046
11	123,1752	103,225	101,6

Tabla 6.5 Potencia Total a frecuencias generadas con DCM 25Mz, 12,5MHz y 5MHz respectivamente.

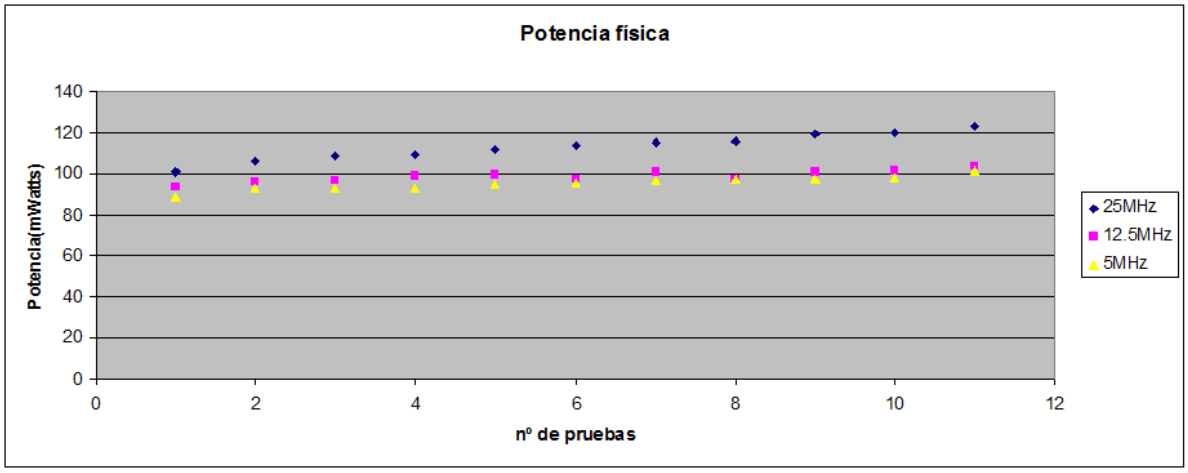

Figura 6.9 Comparación de Potencia total física a distintas frecuencias generadas con DCM en mWatts, 25Mz(serie 1), 12,5MHz(serie 2) y 5MHz(serie 3) respectivamente.

	25MHz	12,5MHz	5MHz
nº de prueba	P. física Estatica	P. física Estatica	P. física Estatica
1	61,338	61,186	61,143
2	61,39	61,24	61,1859
3	61,424	61,3011	61,249
4	61,487	61,3	61,251
5	61,489	61,338	61,294
6	61,52	61,39	61,36
7	61,627	61,45	61,411
8	61,66	61,485	61,454
9	61,685	61,51	61,467
10	61,736	61,5	61,45
11	61,727	61,498	61,446

Tabla 6.6 Potencia física estática a frecuencias generadas con DCM 25Mz, 12,5MHz y 5MHz respectivamente.

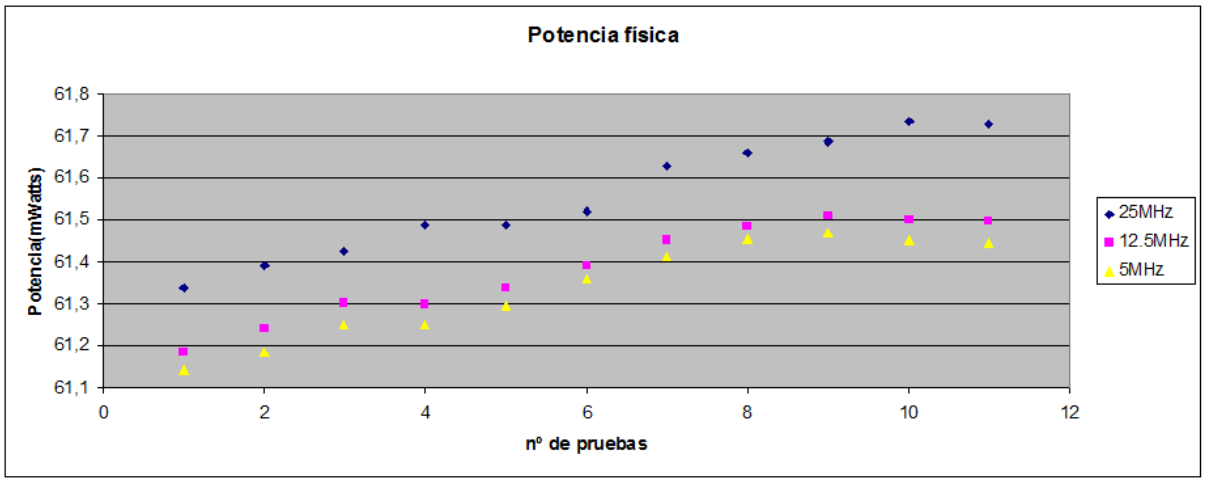

Figura 6.10 Comparación de Potencia estática física a distintas frecuencias generadas con DCM en mWatts, 25Mz(serie 1), 12,5MHz(serie 2) y 5MHz(serie 3) respectivamente.

	25MHz	12.5MHz	5MHz
nº de pruebas	**Xpower P.Total**	**Xpower P.Total**	**Xpower P.Total**
1	56,71	52,36	51,65
2	57,79	52,9	51,99
3	58,87	53,4	52,1
4	59,33	53,52	51,14
5	60,27	53,91	51,19
6	61,25	54,33	52,24
7	62,26	54,74	52,5
8	62,65	54,66	52,7
9	63,72	55,12	53,09
10	64,55	55,42	53,16
11	65,5	55,8	53,34

Tabla 6.7 Potencia con XPA Total a frecuencias generadas con DCM 25Mz, 12,5MHz y 5MHz respectivamente

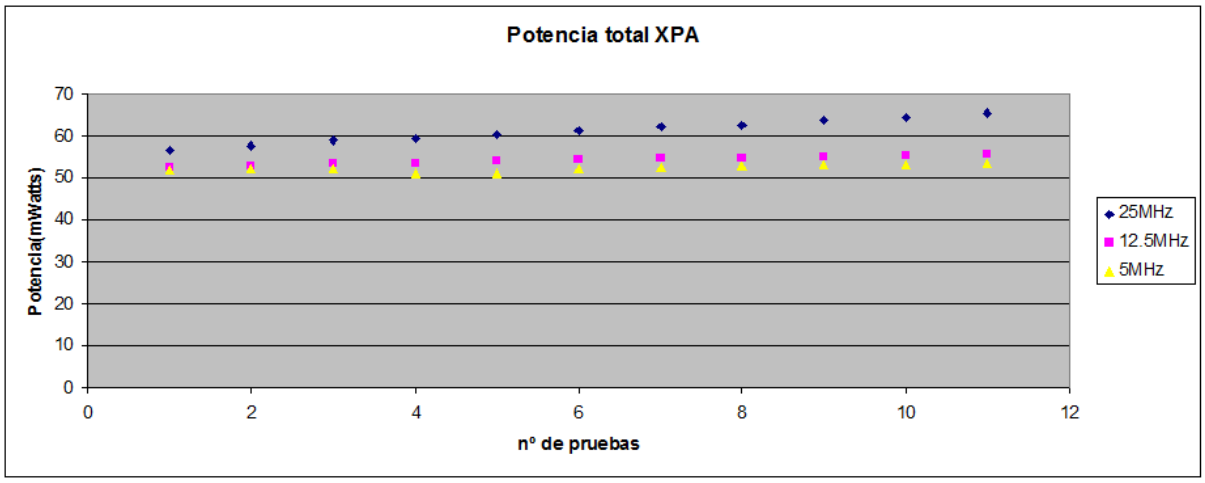

Figura 6.11 Comparación de Potencia total con XPA a distintas frecuencias generadas con DCM en mWatts, 25Mz(serie 1), 12,5MHz(serie 2) y 5MHz(serie 3) respectivamente.

	25MHz	12,5MHz	5MHz
nº de prueba	Xpower Estatica	Xpower Estatica	Xpower Estatica
1	33,88	33,84	33,83
2	33,89	33,84	33,83
3	33,9	33,85	33,84
4	33,91	33,85	33,84
5	33,91	33,85	33,84
6	33,92	33,86	33,84
7	33,93	33,86	33,84
8	33,94	33,86	33,84
9	33,95	33,86	33,85
10	33,96	33,86	33,85
11	33,96	33,87	33,85

Tabla 6.8 Potencia estática con XPA a frecuencias generadas con DCM 25Mz, 12,5MHz y 5MHz respectivamente.

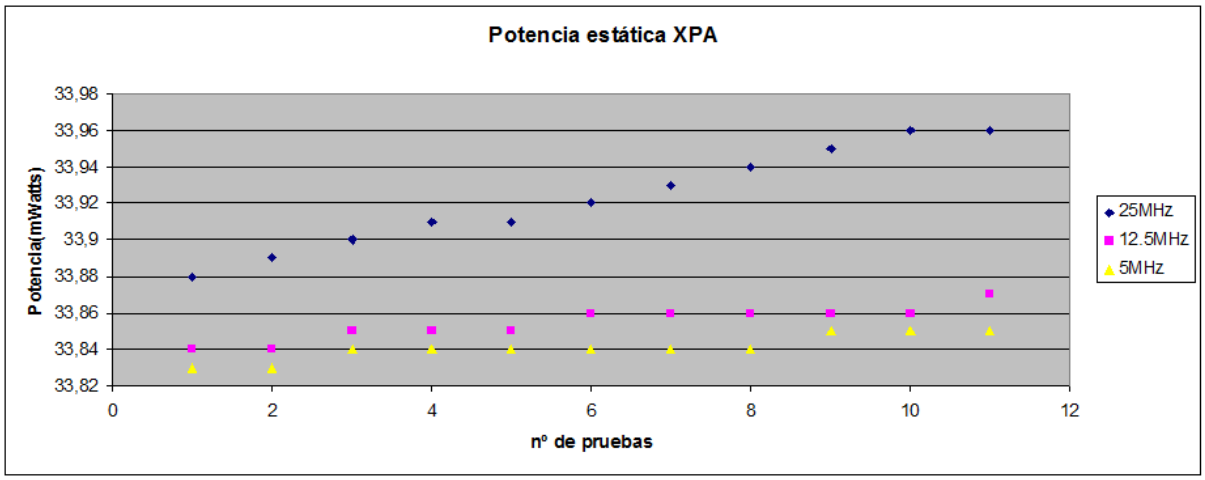

Figura 6.12 Comparación de Potencia estática con XPA a distintas frecuencias f generadas con DCM en mWatts 25Mz(serie 1), 12,5MHz(serie 2) y 5MHz(serie 3) respectivamente.

6.4 Análisis de resultados

A continuación se analizaran los resultados obtenidos en las secciones anteriores. Se compararan los resultados físicos medidos en la placa y los estimados por XPA, también se analizaran entre si datos de cada tabla y lo obtenido respecto a la Correlación y Covarianza entre las diferentes variables.

En cuanto a todas las variables comparadas, se aprecia una fuerte correlación entre ellas y covarianza positivas, en el caso de la covarianza como sabemos esta no implica causalidad, pero si en el caso de la correlación se puede afirmar que existe una fuerte relación y en el mismo sentido de las variables comparadas.

Según los resultados obtenidos, se puede asegurar que ambos métodos tanto las medidas físicas como la estimación con la herramienta XPA proporcionan las mismas conclusiones, las cuales son que a cuanta más frecuencia mayor será el consumo de potencia y que cuanta más lógica tenga el sistema (en este caso el aumento de bits de las entradas y salidas) mayor será el consumo de potencia. Sobre la potencia estática se puede concluir que esta es prácticamente constante y por eso se puede hacer un análisis de la potencia total del sistema a partir de la potencia dinámica del mismo.

CONCLUSIONES

A la vez que las mediciones físicas experimentales, se ha utilizado la herramienta XPA para la estimación del consumo de potencia, cabe mencionar que la estimación de potencia con la herramienta XPA es un proceso lento que requiere ficheros intermedios del orden de décimas hasta cientos de Kbytes, el tamaño de los ficheros puede aumentar de acuerdo a la complejidad del sistema, la generación de un reporte implica una simulación post-place & route de varios minutos dependiendo de la complejidad del sistema puede tardar más, la generación de un reporte también implica un análisis posterior con dicha herramienta. Todo esto da por hecho que utilizar la herramienta XPA es todavía un método tedioso de poca practicidad, no obstante esta herramienta de estimación cumple su función para la caracterización de sistemas no tan complejos como bloques aritméticos donde tanto el tiempo de simulación como la generación de los archivos intermedios para la estimación sea considerablemente menor ahorrando así horas de simulación.

Los resultados observados tanto en la experimentación física como con la herramienta XPA son aceptables, estando relacionadas las variables de consumo medido con la estimación de forma coherente aunque no en un porcentaje constante. Por medio de un estudio estadístico se comprueba correlación y covarianza positiva entre las variables en cuestión.

Los objetivos principales en este trabajo eran:

- Realizar un estudio sobre sistemas digitales, para entender cómo podemos reducir la potencia teniendo en cuenta los parámetros dentro de las ecuaciones que definen el comportamiento del consumo en sistemas digitales.

- Implementar en una FPGA un circuito de referencia para las pruebas físicas experimentales y para probar la eficacia de la herramienta XPA de XILINX.

- Se ha modificado el circuito de referencia aumentando el número de bits de las entradas y salidas, también se ha variado la frecuencia por medio de DCM y por hardware (jumper JP4), para proporcionar diferentes situaciones de medida.

- Se han hecho estas pruebas en la placa obteniendo resultados aceptables y fiables del consumo de potencia en una FPGA.-Se han hecho estas pruebas en la placa obteniendo resultados aceptables y fiables del consumo de potencia en una FPGA.

- Se ha estimado el consumo con la herramienta XPA de cada uno de estos circuitos coincidiendo el aumento de la lógica(aumento de bits de entrada/salida) con el aumento de la potencia consumida mas no en un porcentaje constante. Este resultado también se pudo constatar con el aumento del consumo dinámico al aumentar la frecuencia como se ha probado ya en proyectos anteriores a este, por ende se comprobó cómo afectan los distintos parámetros al consumo de potencia.

- Se han comparado las pruebas físicas con las estimaciones del Xpower Analyzer, para dar con la precisión de esta herramienta de software, se ha hecho un estudio estadístico de estos resultados comprobando que las variables en cuestión presentan correlación y covarianza positiva entre ellas, ahora bien la precisión de XPA según empleados de la compañia XILINX es nominalmente de +-20% para un rango típico de unos pocos Wattios(por debajo de los 15Wattios) lo que difiere enormemente de nuestros resultados de pruebas físicas que resultan en aproximadamente en un 80% más que las estimaciones de la herramienta XPA, esto es debido a que es necesario mantener constante la temperatura de los dispositivos(junction temperature ºC, temperatura de la unión) porque de lo contrario la potencia debido a la variación de temperatura abruma las medidas reales. Entonces debido a que la potencia baja es muy difícil de medir con precisión en esta herramienta XPA y se necesitaría de un dispositivo controlador de temperatura (handler/controlled temperature head) para obtener correctas mediciones físicas, este tema podría derivar en líneas de trabajo futuras para este área.

Por todo esto y más se concluye que este proyecto a la vez que a logrado los objetivos establecidos al principio también añade líneas adicionales en el estudio de este área de consumo de potencia en circuitos digitales.

LINEAS FUTURAS

El objetivo general de este proyecto de fin de máster es hacer un aporte mas al estudio sobre la eficacia y precisión de la herramienta de estimación de potencia XPA de XILINX junto con otro objetivo que es el de analizar los parámetros que influyen en el consumo de sistemas digitales, se puede seguir contribuyendo a este área de estudio y la ampliación de estos experimentos implicaría varios proyectos, las líneas de trabajo que se podrían desprender incluirían entre otras, las técnicas de reducción de consumo en los diferentes niveles de abstracción sean estos, nivel de sistema, de algoritmos, de arquitectura, diseño del circuito, proceso y tecnología y referente al tema principal sobre precisión de la herramienta XPA se podría plantear el diseñar un mecanismo de control de temperatura orientado a FPGAs involucrando sensores cuya caracterización y prueba nos ayuden a controlar la temperatura o bien obtener datos sobre como la temperatura de unión de los dispositivos influyen en las medidas físicas de consumo de la placa BASYS.

REFERENCIAS Y BIBLIOGRAFÍA

[1] Sutter Gustavo, Aporte a la Reducción de Consumo en FPGAs.

[2] Pong P. Chu, RTL Hardware Design Using VHDL

[3] Molinero de la Fuente Rubén, "Caracterización y Reducción de Consumo de Energía en Sistemas Digitales".

[4] Spartan-3E FPGA Family Datasheet.

[5] Digilent BASYS2 Board Reference Manual.

[6] C.Su, C.Tsui,"Low Power Architecture Design and Compilation Techniques for High-Performance Processors", Proc.IEEE C0MPC0N'94.

[7] T.C.Lee and V.Tiwari, "Memory Allocatíon Technique for Low-Energy Embedded DSP Software". 1995 IEEE Symposiumon Low Power Electronics.

[8] Juan Luis Fernández Moya, Implementación y evaluación de algoritmos de compresión en FPGAs.

[9] SMART 2020: "enabling the low carbon economy in the information age" is a report published by "The Climate Group", an independent, not-for-profit organization. The report is available here: http://www.theclimategroup.org/_assets/files/Smart2020Report.pdf

[10] Disruptive Solutions for Energy Efficient ICT, Expert consultation, FET Proactive, 2010. Available at: http://cordis.europa.eu/fp7/ict/fet-proactive/docs/shapefetipwp2011-12-10_en.pdf

[11] ITRS (International Technology Roadmap for Semiconductors), Semiconductor Industry Association, 2001, http://public.itrs.net

[12] R. K. Cavin, V.V. Zhirnov, D. J. C. Herr, A. Avila, and J. Hutchby, Research directions and challenges in nanoelectronics, J. Nanoparticle Res., vol. 8, pp. 841–858, 2006.

[13] Energy Harvesting and Storage for Electronic Devices 2011-2012 – IDTechEx
http://www.idtechex.com/research/reports/energy-harvesting-and-storage-for-electronicdevices-2011-2021-000270.asp

[14] Energy Harvesting for Structural Monitoring – A Roadmap to New Research Challenges, UK EPSRC EH Network Workshop Report, May 2011 http://eh-network.org/resource1.php

[15] Cristian Sisterna, Nota Técnica 12 : Generador de Secuencia Binaria Pseudo Aleatoria www.c7t-hdl.com

[16] Henrik Forstén, Nota Técnica:Generating normally distributed pseudorandom numbers on a FPGA http://hforsten.com/generating-normally-distributed-pseudorandom-numbers-on-a-fpga.html

[17] Jörg Henkel,"A Low Power Hardware/Software Partitioning Approach for Core-BasedEmbeddedSystems". DesignAutomationConference(DAC'99).

[18] VivekTiwari, DeoSingh,"Reducing Power in High-performance Microprocessors", Design Automation Conference(DAC'98).

[19] Luca Benini, Giovanni De Micheli, "Address Bus Encoding Techniques for System-Level Power Optimization", Design Automation and Testin Europe (DATE'98).

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

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

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

[23] 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.

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

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

[26] A. Garcia, "Power consumption and optimization in field programmable gate arrays," Ph.D. thesis, Département Communications et Electronique, Eco/e Nationale Supérieure des Télécommunications, 2000.

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

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

[29] E. A. Kusse, and J. Rabaey, "Low-energy embedded FPGA structures," International Sjmposium on IJOW Power Electronics <&• Design, pp.155-160, August 1998.

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

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

[32] R, Marculescu, D. Marculescu and M. Pedram, "Effícient Power Estímation for Highly Correlated Input Streams", Proceeding of32th Design and Automation Conference (DAC). 1995.

[33] Amelia Shen, Abhijit Ghosh, Srinivas Devadas, and Kurt Keutzer. "On average power dissipation and random pattern testability of CMOS combinational logic networks". In IEEE Int. Conf. on Computer-Aided Design, pages402-407,1992.

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

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

[36] Rabaey, Jan M. "Low power design methodologies", Kluwer Academic publtshers, 1996.

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

[38] Intel corp., "Mobile Intel Pentium III with SpeedStep". Intel web page Second Quarter2000, www.intel.com. 2000.

[39] NEC Electronics Inc, "Power Management Modes In the VR4100 Family of 64-BitMIPS RISC Microprocessors", NEC Application Note, July 1998.

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

[41] C. Su, C. Tsui y A. Despain, "Low Power Architecture Design and Compilation Techniques for High-Performance Processors", Proc. IEEE 1994 Spring C0MPC0N, pp.489-498. IEEE Press, 1994.

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

[43] 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.

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

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

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

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

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

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

[50] Mircea R. Stan; Wayne P. Burleson, "Coding a Terminated Bus for Low Power", Proceedings o/Great Lakes Symposium on VLSI, Buffa/o, NY, USA, pp. 703. March, 1995.

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

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

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

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

[55] 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, París - France, pp. 63-67, March 1996.

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

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

[58] 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.

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

[60] Scott Thompson, Paul Packan and Mark Bohr. "MOS Scaling: Transistor Challenges for the 21st Century", Intel Technology JournalQ3'98,1998.

[61] Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel and Franldin Baez, "Reducing Power in High-performance Microprocessors" In Proceedings of35th Desiga Automation Conference (DAC '98) San Francisco, CA USA. June 1998.

[62] Compaq, Intel, Microsoft, Phoenix, Toshiba, "ACPI Advanced Configuration & Power Interface", Specificatíon Revisión 2.0b, October 11,2002.

[63] 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.

APÉNDICE A: Generador de números Aleatorios

A.1 Generador de Secuencia Binaria Pseudo Aleatoria

La generación de una secuencia pseudo aleatoria de números binarios es útil en este proyecto para test y desarrollo. Un generador de secuencia binaria pseudo aleatoria, es un circuito que genera una serie de números binarios de $\boldsymbol{n-bits}$, un número por ciclo de reloj, sin seguir un patrón determinado, pero que se repite luego de $\mathbf{2^{n-1}}$ ciclos de reloj. Por lo general se implementa como un registro de desplazamiento de realimentación lineal (en inglés Linear Feedback Shift Register – LFSR).

En este apéndice se describe en VHDL un LFSR que genera la secuencia binaria pseudoaleatoria. Se representa en VHDL distintos ejemplos de código y su implementación en una FPGA.

Finalmente se detalla un ejemplo de código VHDL parametrizado ó genérico a fin de tener un generador de números aleatorios a utilizar en este proyecto.

A.1.1 Diseño lógico

Registro de desplazamiento de realimentación Lineal:

El registro de desplazamiento con realimentación lineal (Linear feedback shift register), es un tipo especial de registro de desplazamiento que utiliza un circuito de realimentación particular para generar el dato de entrada serie al registro de desplazamiento. El resultado de esta implementación es la generación de una secuencia pseudo aleatoria de n-bits de $\mathbf{2^{n-1}}$ posibles valores, donde $\boldsymbol{n}$ es el número de registros del registro de desplazamiento. El contenido de cada registro del registro de desplazamiento se desplaza hacia la derecha una posición en cada ciclo de reloj.

El diseño de LFSR se basa en la teoría de campos finitos, desarrollado por Evariste Galois(1811 – 1832). Las operaciones de un LFSR se basan en operaciones en un campo finito con $\mathbf{2^n}$ componentes. Un LFSR de $\boldsymbol{n-bits}$ es una cadena de $\boldsymbol{n}$ registros con una única entrada, en uno de los registros de los extremos. Este dato de entrada, bit de entrada al registro de desplazamiento, es el resultado tras realizar la operación lógica XOR(XNOR) entre ciertos y determinados bits del registro de desplazamiento. Obteniendo los valores lógicos a la salida de cada registro de desplazamiento y uniéndolos de modo de formar un vector a bus de datos, se obtiene como resultado una secuencia de $\mathbf{2^{n-1}}$ números binarios de $\boldsymbol{n-bits}$ que cambia con cada flanco de reloj, cuyo ciclo se repite luego $\mathbf{2^{n-1}}$ ciclos de reloj. Por ejemplo, en un LFSR de n-bits es posible utilizar una simple compuerta lógica XOR, con determinados bits del registro de desplazamiento como entradas, en el camino de realimentación y usando la salida de la XOR como entrada serie del registro de desplazamiento se genera una secuencia pseudo aleatoria de $\mathbf{2^{n-1}}$ números binarios, cuyo ciclo recomienza cada vez que se alcanza el $\mathbf{2^{n-1}}$ ciclos de reloj. Por lo tanto utilizando la teoría del campo finito, se puede demostrar que para cualquier valor de $\boldsymbol{n}$ (número de bits del registro de desplazamiento) existe al menos una ecuación de realimentación basada en la operación lógica XOR o XNOR que genera una secuencia pseudo-aleatoria de números binarios cuando se unen las salidas de los $\boldsymbol{n-bits}$ del registro de desplazamiento, que se repite cada $\mathbf{2^{n-1}}$ ciclos de reloj(el único estado que no es visitado por todos los ceros).

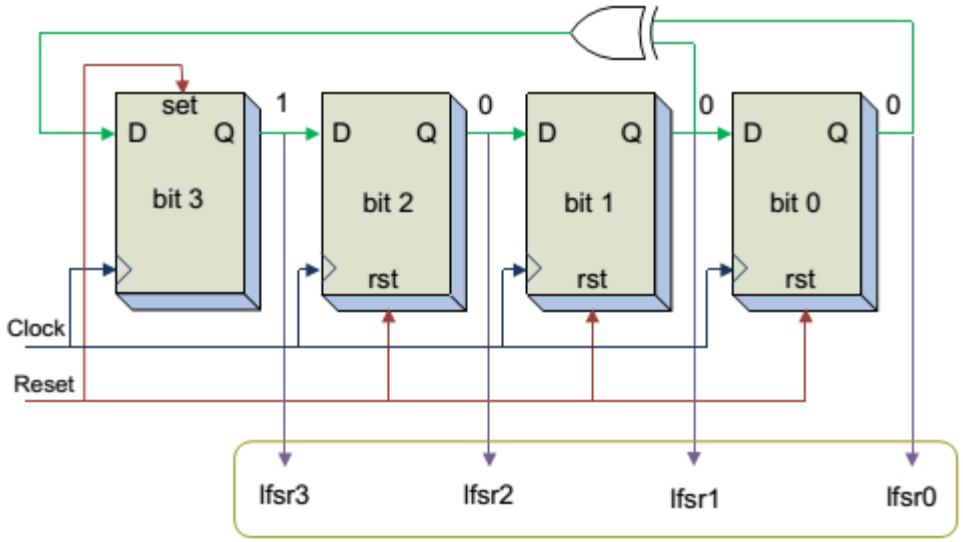

Figura A.1 Implementación de un LFSR de 4 bits.

En primer lugar, los registros de desplazamiento se numeran desde la izquierda, lugar del bit más significativo (MSB) a la derecha bit menos significativo (LSB). El circuito de realimentación está compuesto por una compuerta lógica XOR cuyas entradas son el bit 1 y el bit 0. La salida de la XOR es la entrada serie del LFSR, entrada al bit 3. Ahora pasamos a explicar cómo funciona este circuito.

Suponemos que el estado inicial del registro de desplazamiento es “1000”. En el flanco de subida de sucesivos ciclos de reloj, las siguientes secuencias estarán disponibles en las salidas del registro de:

"0100", "0010", "1001", "1100", "0110", "1011", "0101", "1010 "," 1101 "," 1110 "," 1111 ","

0111 "," 0011 "," 0001 ", …. “1000”, “0100”, ...

De este modo, la salida circula a través de 2^4-1, estados y vuelva a comenzar la misma secuencia nuevamente. La secuencia generada por el LFSR es pseudo-aleatoria, lo que significa que la secuencia muestra una propiedad estadística determinada y parece ser aleatorio.

Se debe tener en cuenta, que el estado "0000" no está incluido en la secuencia y es el único estado que falta de los 16 posibles. Si por alguna razón el LFSR toma el valor "0000", el circuito se quedará en ese estado, pues no tiene forma de salir del mismo.

El valor inicial de la secuencia se denomina comúnmente "semilla" (seed), y se define generalmente como un genérico en la entidad del código VHDL, o una constante en la arquitectura, para facilitar el cambio de la semilla según el valor inicial deseado. En realidad este valor de inicio puede ser cualquier valor excepto el cero, aunque existe un modo de agregar el cero a la secuencia, esto se explica más adelante.

Los resultados de la simulación del código VHDL describiendo un LFSR de 4 bits (detallado arriba), pueden verse en Figura A.2

Figura A.2 Resultados de la simulación de un LFSR de 4 bits.

En las formas de onda de la simulación se puede ver claramente la secuencia pseudo-aleatoria de la salida, llamada q_lfsr_4b, y como comienza el ciclo nuevamente una vez que pasan los 2^n-1 ciclos de reloj.

Otra manera de describir el LFSR de 4 bits, de un modo más compacto es usando un foor-loop.

Este ejemplo es una modificación del descrito en el libro “RTL GHardware Design Using VHDL”, de Pong Chu[2].

El siguiente código detalla el código VHDL respectivo.

```
---------------------------------- ejemplo 2 de 4-bits LFSR ------------------------------------

library ieee;

use ieee.std_logic_1164.all;

---------------------------------------- Entiy ----------------------------------------------------

entity lfsr_loop is

generic(initial_value: std_logic_vector(3 downto 0):= "1000";

lfsr_width  : integer := 4);

port(

clk  : in std_logic;

rst  : in std_logic;

q_lfsr_4b: out std_logic_vector(lfsr_width-1 downto 0)

);

end lfsr_loop;

------------------------ Architecture ------------------------------------------------------------------

architecture beh of lfsr_loop is

signal q_lfsr_4b_i: std_logic_vector(lfsr_width-1 downto 0);

begin
```

```
-- Shifter Process

lfsr_cnt_proc: process(rst, clk)

begin

if(rst= '1' ) then

q_lfsr_4b_i <= initial_value;

elsif (rising_edge(clk)) then

-- shift operation: b3->b2, b2->b1, b1->b0

shifter_loop: for i in 3 downto 1 loop

q_lfsr_4b_i(i-1) <= q_lfsr_4b_i(i);

end loop shifter_loop;

-- Serial Input to the b3 of the LFSR

q_lfsr_4b_i(3) <= q_lfsr_4b_i(1) xor q_lfsr_4b_i(0);

end if;

end process lfsr_cnt_proc;

q_lfsr_4b <= q_lfsr_4b_i;

end architecture beh;
```

A.1.2 Implementación en FPGA

Una gran ventaja en la implementación de un LFSR es la reducida cantidad de recursos necesarios para la construcción del respectivo circuito. Por ejemplo, para implementar un generador pseudoaleatorio de 32-bits solo se necesitan 32 registros y 3 compuertas lógicas XOR. De este modo, se puede construir, con muy bajos recursos, un LFSR de 32-bit con ciclos de $2^{(32}-1)$ estados. Por otra parte, si se quisiera construir un contador binario de 32 bits la cantidad de recursos es grande.

A.1.3 Bits a Realimentar del Registro de Desplazamiento (TAPs)

El circuito de realimentación del LFSR depende del número de bits del LFSR, y los bits a tomar como entrada a la compuerta lógica XOR ya están determinados sobre una base ad hoc (de la teoría de campo finito). La expresión de la realimentación comúnmente es muy simple y puede implicar hasta cuatro operaciones lógicas XOR o XNOR.

La siguiente tabla detalla los bits del LFSR que forman parte de la realimentación. Siempre la salida de la XOR o XNOR está conectada a la entrada serie del LFSR.

n	XNOR from	n	XNOR from	n	XNOR from	n	XNOR from
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	10,7	52	52,49	94	94,73	136	136,135,11,10
11	11,9	53	53,52,38,37	95	95,84	137	137,116
12	12,6,4,1	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,94	143	143,142,123,122
18	18,11	60	60,59	102	102,101,36,35	144	144,143,75,74
19	19,6,2,1	61	61,60,46,45	103	103,94	145	145,93
20	20,17	62	62,61,6,5	104	104,103,94,93	146	146,145,87,86
21	21,19	63	63,62	105	105,89	147	147,146,110,109
22	22,21	64	64,63,61,60	106	106,91	148	148,121
23	23,18	65	65,47	107	107,105,44,42	149	149,148,40,39
24	24,23,22,17	66	66,65,57,56	108	108,77	150	150,97
25	25,22	67	67,66,58,57	109	109,108,103,102	151	151,148
26	26,6,2,1	68	68,59	110	110,109,98,97	152	152,151,87,86
27	27,5,2,1	69	69,67,42,40	111	111,101	153	153,152
28	28,25	70	70,69,55,54	112	112,110,69,67	154	154,152,27,25
29	29,27	71	71,65	113	113,104	155	155,154,124,123
30	30,6,4,1	72	72,66,25,19	114	114,113,33,32	156	156,155,41,40
31	31,28	73	73,48	115	115,114,101,100	157	157,156,131,130
32	32,22,2,1	74	74,73,59,58	116	116,115,46,45	158	158,157,132,131
33	33,20	75	75,74,65,64	117	117,115,99,97	159	159,128
34	34,27,2,1	76	76,75,41,40	118	118,85	160	160,159,142,141
35	35,33	77	77,76,47,46	119	119,111	161	161,143
36	36,25	78	78,77,59,58	120	120,113,9,2	162	162,161,75,74
37	37,5,4,3,2,1	79	79,70	121	121,103	163	163,162,104,103
38	38,6,5,1	80	80,79,43,42	122	122,121,63,62	164	164,163,151,150
39	39,35	81	81,77	123	123,121	165	165,164,135,134
40	40,38,21,19	82	82,79,47,44	124	124,87	166	166,165,128,127
41	41,38	83	83,82,38,37	125	125,124,18,17	167	167,161
42	42,41,20,19	84	84,71	126	126,125,90,89	168	168,166,153,151
43	43,42,38,37	85	85,84,58,57	127	127,126		
44	44,43,18,17	86	86,85,74,73	128	128,126,101,99		

Tabla A.1 Bits a Realimentar del Registro de Desplazamiento (TAPs)

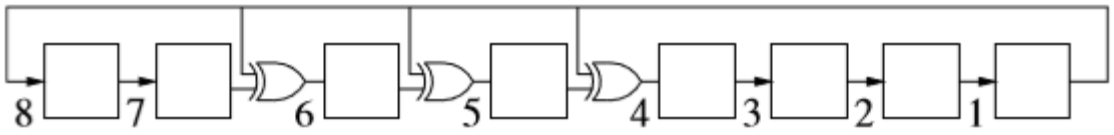

Figura A.3 LFSR de secuencia binaria pseudoaleatoria de ciclo 255, este LFSR tiene (bits a realimentar) TAPS en las posiciones 8,6,5 y 4.

Las expresiones detalladas en la tabla de arriba para cada LFSR proveen lo que se llama “máxima longitud del LFSR”, por lo que utilizando la expresión correspondiente en el circuito de realimentación, generará la secuencia binaria pseudo aleatoria de (2^n) -1 valores, cuyo ciclo dura (2^n) -1 ciclos de reloj, y se repite indefinidamente. Se debe aclarar que por cada determinado número de registros puede haber otra expresión que origine una máxima longitud del LFSR, pero con mayor cantidad de bits en la expresión. También, puede haber otras expresiones que generen solo un número determinado de valores, mucho menos que los (2^n) -1 posibles.

Cualquiera de los generadores LFSR generan una secuencia pseudo-aleatoria de unos y ceros, la secuencia no exactamente aleatoria, ya que se repite, y sigue una cierto comportamiento explicado en la teoría de los campos finitos. Sin embargo a los fines prácticos, la secuencia generada puede considerarse aleatoria. Por ejemplo, un LFSR de 60 bits con un reloj corriendo a 50MHz, repite la secuencia después de 731 años, tiempo suficientemente lago como para ser considerado aleatorio. En el caso de un LFSR de 63 bits, con un reloj de 50MHz, el ciclo se repite luego de 5849 años.

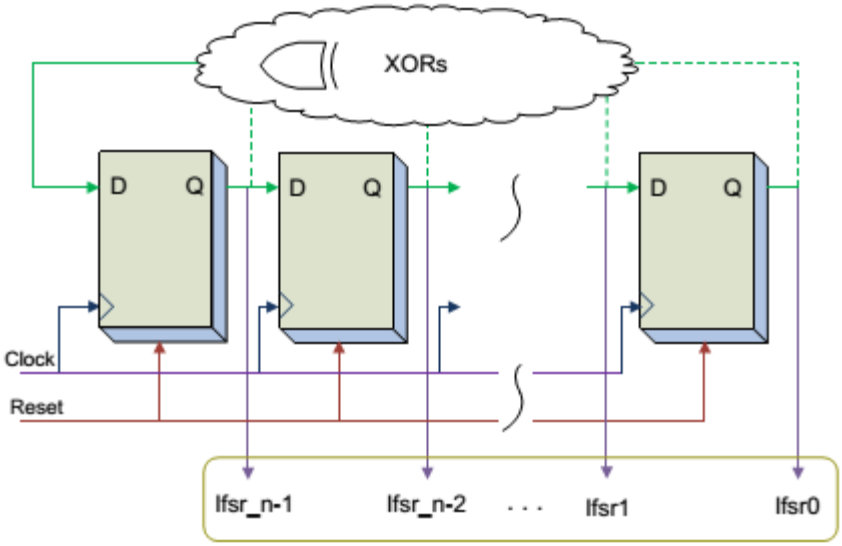

Figura A.4 Diagrama general de un LFSR

A.1.4 XOR or XNOR Feeback:

El circuito de realimentación puede estar basado en compuertas lógicas XOR o XNOR. Estas compuertas son intercambiables, aunque el LFSR genere los mismos valores, la secuencia sería distinta.

APÉNDICE B: MULTIPLICADOR GENÉRICO

En este apéndice pasamos a explicar el código del multiplicador genérico.

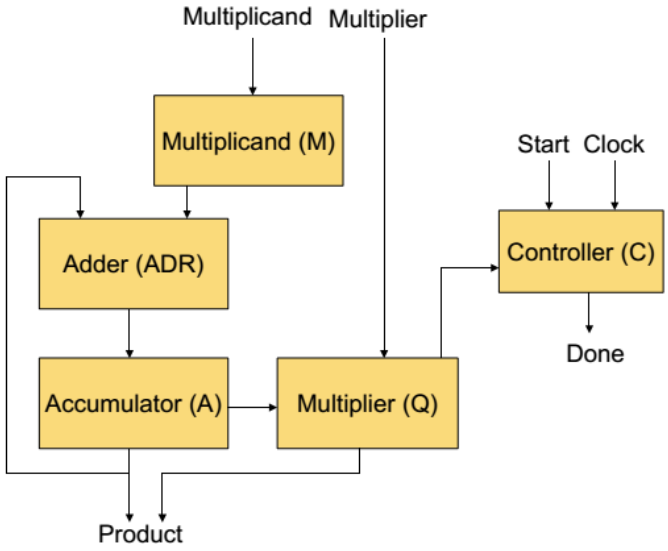

Figura B.1 Diagrama de bloques del multiplicador binario genérico.

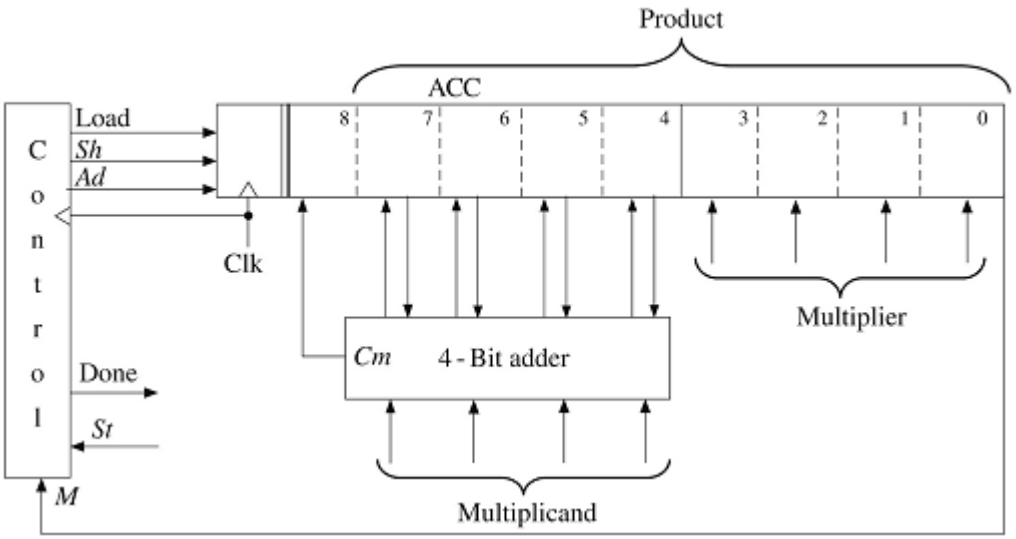

Figura B.2 Diagrama de bloques del multiplicador binario con señales de control.

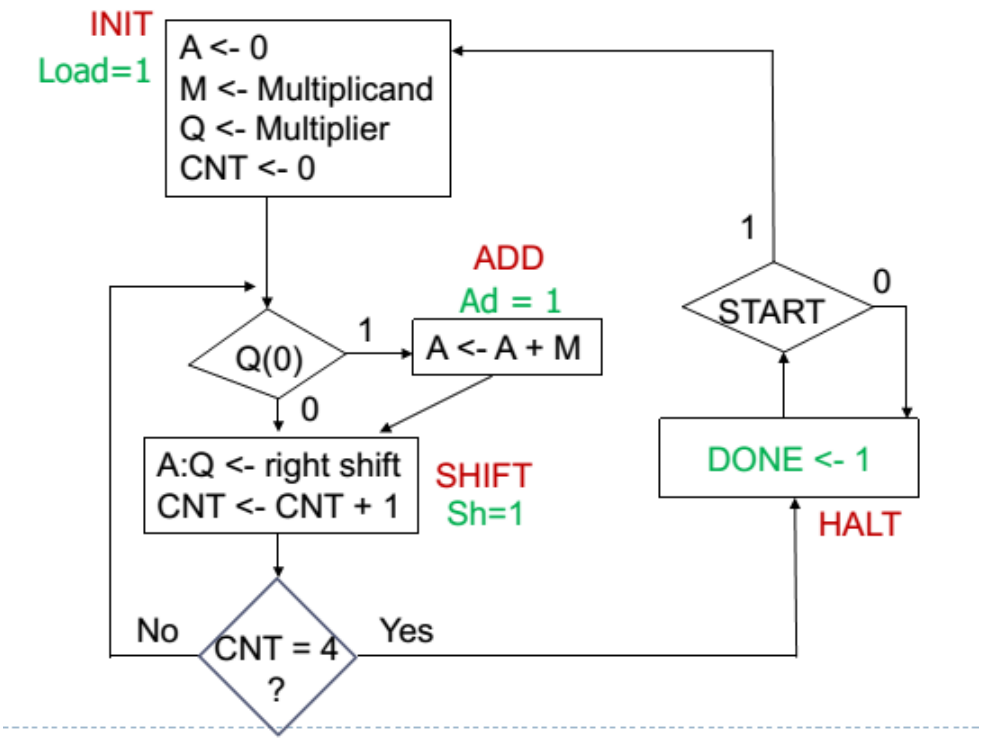

Figura B.3 Algoritmo de suma y desplazamiento.

A continuación el código del Multiplicador por cada Bloque, empezamos por el TOP.

B.1 CÓDIGO DEL TOP:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.mult_components.all;
-- Montaje total del Multiplicador y el Generador de números Aleatorios.
Entity MONTAJE_TOP is
generic( nn_bits: integer := 7);
PORT(
START_out_sal:  out std_logic;
RESET_out_sal:  out std_logic;
DONE_out_sal: out std_logic;
MM1_sal: out std_logic_vector(nn_bits-1 downto 0); -- se muestra el numero asignado al Operando M1
```

```
MM2_sal: out std_logic_vector(nn_bits-1 downto 0); -- se muestra el numero asignado al Operando
M2

NUM_GEN_sal: out std_logic_vector(50 downto 0);

Clk : in  STD_LOGIC;

RESULTADO: out std_logic_vector(2*nn_bits-1 downto 0) );

End Entity;

architecture Behavioral of MONTAJE_TOP is

----------------------------------------------------

-- definición de señales

-- SEÑALES LFSR

signal reset :   STD_LOGIC;

signal random_signal :  STD_LOGIC_VECTOR (70 downto 0);

signal contador: integer := 0;

-- FIN señales de LFSR

-- SEÑALES DE MultTop

--------------

signal START_aux:  std_logic;

signal START_out:  std_logic;

signal RESET_out:   std_logic;

signal DONE_out:  std_logic;

signal RESULTAD:  std_logic_vector(2*nn_bits-1 downto 0);

-- fin definicion señales

--- Definicion de componentes

component MultTop is

generic( NN:integer:= 7 );

port (

Clk:  in std_logic;

Multiplier: in std_logic_vector(NN-1 downto 0);

Multiplicand: in std_logic_vector(NN-1 downto 0);

Product: out std_logic_vector(2*NN-1 downto 0);

Start: out std_logic;
```

```
Done: out std_logic );
end component;
----------------------------------------------
component LFSR is
generic(
 --lfsr_width : natural := 10;
 lfsr_width : natural := 71;
 all_zeros_state: boolean:= false);
    Port (
clk : in std_logic;
rst : out std_logic;
q_lfsr_4b: out std_logic_vector(lfsr_width-1 downto 0) );
end component;
----------------------------------------------
-- Fin Definición de componentes
BEGIN
-- AREA DE INSTANCIACIONES
-- Instanciación para el LFSR
LFSR1: LFSR
    Port map(
 clk => DONE_out,  --La señal de DONE  es el clock del LFSR
 rst => RESET_out,
 q_lfsr_4b => random_signal   );
-- FIN de Instanciacion para el LFSR
-- Instanciacion para MultTop
MultTop1: MultTop
Port map(
Clk => Clk,
Multiplier => random_signal(2*nn_bits-1 downto nn_bits),
Multiplicand => random_signal(nn_bits-1 downto 0) ,
Product => RESULTAD,
```

```
Start => START_out,
Done => DONE_out  );
-- Fin Instanciación para MultTop
-- FIN ÁREA DE INSTANCIACIONES
-- AREA DE PROCESS
P1: process(Clk, random_signal, DONE_out,  RESET_out, START_out, contador, START_aux )
BEGIN
if((Clk'event)and (Clk = '1')) then
RESULTADO <= RESULTAD;
START_out_sal <= START_out;
RESET_out_sal <= RESET_out;
DONE_out_sal <= DONE_out;
MM1_sal <= random_signal(2*nn_bits-1 downto nn_bits) ; -- nn_bits
MM2_sal <= random_signal(nn_bits-1 downto 0);
NUM_GEN_Sal(50 downto 0) <= random_signal(50 downto 0);
end if;
end process;
end Behavioral;
```

B.2 CÓDIGO DEL ADDER DEL MULTIPLICADOR:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity AdderN is
generic (N: integer := 7);
port(
AddA_in: in std_logic;
Clk : in std_logic;
op1: in std_logic_vector(N-1 downto 0):=(others=>'0'); -- N bit Addend
op2: in std_logic_vector(N-1downto 0):=(others=>'0'); -- N bit Augend
resultado_out: out std_logic_vector(N downto 0):=(others=>'0') -- N+1 bit result, includes carry
);
end AdderN;
```

```
architecture Behavioral of AdderN is
------ funcion suma
    FUNCTION suma (op1, op2: std_logic_vector) RETURN std_logic_vector IS
      VARIABLE parcial: std_logic_vector (2 downto 0);
      VARIABLE llevada: std_logic := '0';
     VARIABLE result: std_logic_vector (N downto 0);
 VARIABLE resultado: std_logic_vector(N-1 downto 0);
    BEGIN
          for i IN 0 to N-1 loop
        parcial := op1 (i) & op2 (i) & llevada;
        CASE parcial IS
           when "000" => llevada := '0'; resultado (i) := '0';
            when "001" => llevada := '0'; resultado (i) := '1';
            when "010" => llevada := '0'; resultado (i) := '1';
            when "011" => llevada := '1'; resultado (i) := '0';
            when "100" => llevada := '0'; resultado (i) := '1';
            when "101" => llevada := '1'; resultado (i) := '0';
            when "110" => llevada := '1'; resultado (i) := '0';
            when "111" => llevada := '1'; resultado (i) := '1';
     when others => llevada := '0'; resultado (i) := '0';
          END CASE;
       end loop;
      result := llevada&resultado;
     return result;
    END FUNCTION suma;
  ------ funcion suma
begin --begin de la arquitectura

P101: process(Clk, op1, op2, llevadi, AddA_in )
begin -- begin del process
IF (Clk'event and Clk = '1') THEN
resultado_out <= suma(op1, op2);
end if;
```

```
end process;
end Behavioral;
```

B.3 CÓDIGO DEL DEL REGISTRO DE DESPLAZAMIENTO DEL MULTIPLICADOR:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity RegN is
generic (N: integer := 7);
port ( --Din: in std_logic_vector(N-1 downto 0); --N-bit input   -- :=(others=>'0');
Din: in std_logic_vector(N-1 downto 0):=(others=>'0');
Dout: out std_logic_vector(N-1 downto 0):=(others=>'0'); --N-bit output
Clk: in std_logic; --Clock (rising edge)
Load: in std_logic; --Load enable
Shift: in std_logic; --Shift enable
Clear: in std_logic; --Clear enable
SerIn: in std_logic --Serial input
);
end RegN;
architecture Behavioral of RegN is
signal Dinternal: std_logic_vector(N-1 downto 0):=(others=>'0'); -- Internal state
begin
process (Clk)
begin
if (rising_edge(Clk)) then
if (Clear = '1') then
Dinternal <= (others => '0'); -- Clear
elsif (Load = '1') then
Dinternal <= Din; -- Load
elsif (Shift = '1') then
Dinternal <= SerIn & Dinternal(N-1 downto 1); -- Shift
end if;
end if;
end process;
```

```
Dout <= Dinternal; -- Drive outputs**
end Behavioral;
```

B.4 CÓDIGO DEL CONTROL DE SEÑALES DEL MULTIPLICADOR:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Controller is
generic (N: integer := 2); -- # of counter bits
port ( Clk: in std_logic; -- Clock (use rising edge)
Q0: in std_logic; -- LSB of multiplier
Start: out std_logic := '1'; -- Algorithm start pulse
Load: out std_logic; -- Load M,Q and Clear A
Shift: out std_logic; -- Shift A:Q
AddA: out std_logic; -- Load Adder output to A
Done: out std_logic -- Indicate end of algorithm
);
end Controller;

architecture Behavioral of Controller is
type states is (HaltS,InitS,QtempS,AddS,ShiftS);
signal state: states := HaltS;
signal CNT: unsigned(N-1 downto 0):=(others=>'0');
signal contador: integer := 0;
signal Starti:  std_logic := '0';
begin
-- Moore model outputs to control the datapath
Done <= '1' when state = HaltS else '0'; -- End of algorithm
Load <= '1' when state = InitS else '0'; -- Load M/Q, Clear A
Starti <= '0' when state = InitS else '1';
AddA <= '1' when state = AddS else '0'; -- Load adder to A
Shift <= '1' when state = ShiftS else '0'; -- Shift A:Q
Start <=  Starti;
process(Clk, Starti, contador)
```

```
begin
if rising_edge(Clk) then
case state is
when HaltS => if Starti = '1' then -- Start pulse applied
state <= InitS; -- Start the algorithm
end if;
when InitS => state <= QtempS; -- Test Q0 at next clock**
when QtempS => if (Q0 = '1') then
state <= AddS; -- Add if multiplier bit = 1
else
state <= ShiftS; -- Skip add if multiplier bit = 0
end if;
when AddS => state <= ShiftS; -- Shift after add
when ShiftS => if (CNT = 2**N - 1) and (contador = 7) then --
state <= HaltS; -- Halt after 2^N iterations
else
state <= QtempS; -- Next iteration of algorithm: test Q0 **
end if;
end case;
end if;
end process;
P2: process(Clk, Starti)
begin
if rising_edge(Clk) then
if state = InitS then
CNT <= to_unsigned(0,N); -- Reset CNT in InitS state
contador <= 0;
elsif state = ShiftS then
CNT <= CNT + 1; -- Count in ShiftS state
contador <= contador + 1;
end if;
end if;
end process;
end Behavioral;
```

B.5 CÓDIGO DEL DEL LFSR:

En el Apéndice A se desarrolla la teoría sobre el código del generador de números aleatorio

```
---------------------------------------------- LFSR Parametrizado ----------- 71 bits ----------------------------------
--------------library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

entity LFSR is

 generic(

lfsr_width : natural := 71;

all_zeros_state: boolean:= false);

port(

 clk : in std_logic;

 rst : out std_logic;

 q_lfsr_4b: out std_logic_vector(lfsr_width-1 downto 0)

);

end LFSR;

-----------------------------------------------------

 -- Architecture -- ------------------------------

-----------------------------------------------

architecture beh of LFSR is

-- signals declarations

signal reset :std_logic := '1';

-- fin signals declarations

-- constant declarations

constant max_width : natural := 72; -- max # of registers in the LFSR

constant initial_value: std_logic_vector (lfsr_width-1 downto 0) :=
"10111111011111111111111111111111111111111100001000100000100010000010001000";

type feedback_array_type is array (2 to max_width) of

 std_logic_vector(max_width-1 downto 0);

constant feedback_equation: feedback_array_type :=

(2 => (1 | 0 => '1', others => '0'),
```

```
3 => (3 | 2 => '1', others => '0'),

4 => (4 | 3 => '1', others => '0'),

5 => (5 | 3 => '1', others => '0'),

6 => (6 | 5 => '1', others => '0'),

7 => (7 | 6 => '1', others => '0'),

8 => (8 | 6 | 5 | 4 => '1', others => '0'),

9 => (5 | 4  => '1', others => '0'),

10 => (9 | 5  => '1', others => '0'),

11 => (10 | 7  => '1', others => '0'),

12 => (13 | 4 | 3| 1 => '1', others => '0'),

13 => (13 | 4 | 3 | 1 => '1', others => '0'),

14 => (14 | 5 | 3 | 1 => '1', others => '0'),

15 => (15 | 14  => '1', others => '0'),

16 => (16 | 15 | 13 | 4 => '1', others => '0'),

17 => (17 | 14  => '1', others => '0'),

18 => (18 | 11  => '1', others => '0'),

19 => (19 | 6 | 2 | 1 => '1', others => '0'),

20 => (20 | 17  => '1', others => '0'),

21 => (21 | 19  => '1', others => '0'),

22 => (22 | 21  => '1', others => '0'),

23 => (23 | 18  => '1', others => '0'),

24 => (24 | 23 | 22 | 17 => '1', others => '0'),

25 => (25 | 22  => '1', others => '0'),

26 => (26 | 6 | 2 | 1 => '1', others => '0'),

27 => (27 | 5 | 2 | 1 => '1', others => '0'),

28 => (28 | 25  => '1', others => '0'),

29 => (29 | 27  => '1', others => '0'),

30 => (30 | 6 | 4 | 1 => '1', others => '0'),

31 => (31 | 28  => '1', others => '0'),

32 => (32 | 22 | 2 | 1 => '1', others => '0'),
```

```
33 => (33 | 20  => '1', others => '0'),
34 => (34 | 27 | 2 | 1 => '1', others => '0'),
35 => (35 | 33  => '1', others => '0'),
36 => (36 | 25  => '1', others => '0'),
37 => (37 | 5 | 4 | 3| 2 | 1  => '1', others => '0'),
38 => (38 | 6 | 5 | 1 => '1', others => '0'),
39 => (39 | 35  => '1', others => '0'),
40 => (40 | 38 | 21 | 19 => '1', others => '0'),
41 => (41 | 38  => '1', others => '0'),
42 => (42 | 41 | 20 | 19 => '1', others => '0'),
43 => (43 | 42 | 38 | 37 => '1', others => '0'),
44 => (44 | 43 | 18 | 17 => '1', others => '0'),
45 => (45 | 44 | 42 | 41 => '1', others => '0'),
46 => (46 | 45| 26 | 25 => '1', others => '0'),
47 => (47 | 42  => '1', others => '0'),
48 => (48 | 47| 21 | 20 => '1', others => '0'),
49 => (49 | 40  => '1', others => '0'),
50 => (50 | 49| 24 | 23 => '1', others => '0'),
51 => (51 | 50| 36 | 35 => '1', others => '0'),
52 => (52 | 49 => '1', others => '0'),
53 => (53 | 52| 38 | 37 => '1', others => '0'),
54 => (54 | 53| 18 | 17 => '1', others => '0'),
55 => (55 | 31  => '1', others => '0'),
56 => (55 | 35| 34 | 17 => '1', others => '0'),
57 => (57 | 50  => '1', others => '0'),
58 => (58 | 39  => '1', others => '0'),
59 => (59 | 58| 38 | 37 => '1', others => '0'),
60 => (60 | 59  => '1', others => '0'),
61 => (61 | 60| 46 | 45 => '1', others => '0'),
62 => (62 | 61| 6 | 5 => '1', others => '0'),
63 => (63 | 62  => '1', others => '0'),
```

```
64 => (64 | 63| 61 | 60 => '1', others => '0'),
65 => (65 | 47  => '1', others => '0'),
66 => (66 | 65| 57 | 56  => '1', others => '0'),
67 => (67 | 66| 58 | 57  => '1', others => '0'),
68 => (68 | 59  => '1', others => '0'),
69 => (69 | 67| 42 | 40  => '1', others => '0'),
70 => (70 | 69| 55 | 54  => '1', others => '0'),
71 => (70 | 65  => '1', others => '0'),
72 => (71 | 66| 25 | 19  => '1', others => '0')
--73 => (73 | 48  => '1', others => '0'),
--74 => (74 | 73| 59 | 58  => '1', others => '0'),
--75 => (73 | 74| 65 | 64  => '1', others => '0')
--76 => (75 | 74| 41 | 40  => '1', others => '0')
--77 => (76 | 75| 47 | 46  => '1', others => '0')
--78 => (77 | 77| 59 | 58  => '1', others => '0')
--79 => (77 | 70  => '1', others => '0')
--80 => (78 | 77| 43 | 42  => '1', others => '0')
--81 => (80 | 77  => '1', others => '0')
--82 => (81 | 79| 47 | 44  => '1', others => '0')
--83 => (82 | 81| 38 | 37   => '1', others => '0'),
--84 => (84 | 71  => '1', others => '0'),
--85 => (85 | 84| 58 | 57   => '1', others => '0'),
--86 => (86 | 85| 74 | 73   => '1', others => '0'),
--87 => (87 | 74  => '1', others => '0'),
--88 => (88 | 87| 17 | 16   => '1', others => '0'),
--89 => (89 | 51  => '1', others => '0'),
--90 => (90 | 89| 72 | 71   => '1', others => '0'),
--91 => (91 | 90| 8 | 7   => '1', others => '0'),
--92 => (92 | 91| 80 | 79   => '1', others => '0'),
--93 => (93 | 91  => '1', others => '0'),
```

```
--94 => (94 | 73  => '1', others => '0'),

--95 => (95 | 84  => '1', others => '0'),

--96 => (96 | 94| 49 | 47   => '1', others => '0'),

--97 => (97 | 91  => '1', others => '0'),

--98 => (98 | 87  => '1', others => '0'),

--99 => (99 | 97| 54 | 52   => '1', others => '0'),

--100 => (100 | 63  => '1', others => '0'),

--101 => (101 | 100| 95 | 94   => '1', others => '0'),

--102 => (102 | 101| 36 | 35   => '1', others => '0'),

--103 => (103 | 94  => '1', others => '0'),

--104 => (104 | 103| 94 | 93   => '1', others => '0'),

--105 => (105 | 89  => '1', others => '0'),

--106 => (106 | 91  => '1', others => '0'),

--107 => (107 | 105| 44 | 42   => '1', others => '0'),

--109 => (109 | 108| 103 | 102   => '1', others => '0'),

--110 => (110 | 109| 98 | 97   => '1', others => '0'),

--111 => (111 | 101  => '1', others => '0'),

--112 => (112 | 110| 69 | 67   => '1', others => '0'),

--113 => (113 | 104  => '1', others => '0'),

--114 => (114 | 113| 33 | 32   => '1', others => '0'),

--115 => (115 | 114| 101 | 100   => '1', others => '0'),

--116 => (116 | 115| 46 | 45   => '1', others => '0'),

--117 => (117 | 115| 99 | 97   => '1', others => '0'),

--118 => (118 | 85  => '1', others => '0'),

--119 => (119 | 111  => '1', others => '0'),

--120 => (120 | 113| 9 | 2   => '1', others => '0'),

--121 => (121 | 103  => '1', others => '0'),

--122 => (122 | 121| 63 | 62   => '1', others => '0'),

--123 => (123 | 121  => '1', others => '0'),

--124 => (124 | 87  => '1', others => '0'),

--125 => (125 | 124| 18 | 17   => '1', others => '0'),
```

```
--126 => (126 | 125| 90 | 89   => '1', others => '0'),
--127 => (127 | 126  => '1', others => '0'),
--128 => (128 | 126| 101 | 99   => '1', others => '0'),
--129 => (129 | 124  => '1', others => '0'),
--130 => (130 | 127  => '1', others => '0'),
--131 => (131 | 130| 84 | 83   => '1', others => '0'),
--132 => (132 | 103  => '1', others => '0'),
--133 => (133 | 132| 82 | 81   => '1', others => '0'),
--134 => (134 | 77  => '1', others => '0'),
--135 => (135 | 124  => '1', others => '0'),
--136 => (136 | 135| 11 | 10   => '1', others => '0'),
--137 => (137 | 116  => '1', others => '0'),
--138 => (138 | 137| 131 | 130   => '1', others => '0'),
--139 => (139 | 136| 134 | 131   => '1', others => '0'),
--140 => (140 | 111  => '1', others => '0'),
--141 => (141 | 140| 110 | 109   => '1', others => '0'),
--142 => (142 | 121  => '1', others => '0'),
--143 => (143 | 142| 123 | 122   => '1', others => '0'),
--144 => (144| 143| 75 | 74   => '1', others => '0'),
--145 => (145 | 93  => '1', others => '0'),
--146 => (146| 145| 87 | 86   => '1', others => '0'),
--147 => (147| 146| 110 | 109   => '1', others => '0'),
--148 => (148 | 121  => '1', others => '0'),
--149 => (149| 148| 40 | 39   => '1', others => '0'),
--150 => (150 | 97  => '1', others => '0'),
--151 => (151 | 148  => '1', others => '0'),
--152 => (152| 151| 87 | 86   => '1', others => '0'),
--153 => (153 | 152  => '1', others => '0'),
--154 => (154| 152| 27 | 25   => '1', others => '0'),
--155 => (155| 154| 124 | 123   => '1', others => '0'),
```

```
--156 => (156| 155| 41 | 40   => '1', others => '0'),
--157 => (157| 156| 131 | 130   => '1', others => '0'),
--158 => (158| 157| 132 | 131   => '1', others => '0'),
--159 => (159 | 128  => '1', others => '0'),
--160 => (160| 159| 142 | 141   => '1', others => '0'),
--161 => (160 | 143  => '1', others => '0')
--162 => (162| 161| 75 | 74   => '1', others => '0'),
--163 => (163| 162| 104 | 103   => '1', others => '0'),
--164 => (164| 163| 151 | 150   => '1', others => '0'),
--165 => (165| 164| 135 | 134   => '1', others => '0'),
--166 => (166| 165| 128 | 127   => '1', others => '0'),
--167 => (167 | 161  => '1', others => '0'),
--168 => (167| 166| 153 | 151   => '1', others => '0')
); -- final de matriz
-- signal declarations
signal q_lfsr_4b_i : std_logic_vector(lfsr_width-1 downto 0);
signal nor_detect_0s: std_logic;
signal serial_in : std_logic;
signal feedback : std_logic;
begin
rst <= reset;
-------------------------------------------------------------
-- Process to get feedback XOR from the feedback equation --
feedb_proc: process(q_lfsr_4b_i)
-- process local declarations
constant tap_constant: std_logic_vector(max_width-1 downto 0) := feedback_equation(lfsr_width);
variable tmp: std_logic;
begin
tmp := '0';
bit_xor: for i in 0 to lfsr_width-1 loop
-- tmp := tmp xor (q_lfsr_4b_i(i) and tap_constant(i));
```

```
--tmp := NOT( tmp xor (q_lfsr_4b_i(i) and tap_constant(i)) );
tmp :=  tmp xnor (q_lfsr_4b_i(i) and tap_constant(i));
end loop bit_xor;
feedback <= tmp;
end process feedb_proc;
-------------------------------------------------------------
-------------------------------------------------------------
-- Shifter Process
lfsr_cnt_proc: process(reset, clk)
begin
if(reset= '1' ) then
q_lfsr_4b_i <= initial_value;
reset <= '0';
elsif (rising_edge(clk)) then
 -- shift operation: b3->b2, b2->b1, b1->b0
--shifter_loop: for i in 3 downto 1 loop
shifter_loop: for i in lfsr_width-1 downto 1 loop
q_lfsr_4b_i(i-1) <= q_lfsr_4b_i(i);
end loop shifter_loop;
 -- Serial Input to the b3 of the LFSR
 q_lfsr_4b_i(q_lfsr_4b_i'high) <= serial_in;
end if;
end process lfsr_cnt_proc;
q_lfsr_4b <= q_lfsr_4b_i;
-------------------------------------------------------------
-- piece of code to be generated when the 'all zeros'
-- state is needed
-------------------------------------------------------------
gen_zero: if (all_zeros_state = TRUE) generate
nor_detect_0s <= '1' when q_lfsr_4b_i(lfsr_width-1 downto 1)=
```

```
(q_lfsr_4b_i(lfsr_width-1 downto 1)'range=>'0') else'0';
-- Serial Input to the LFSR
 serial_in <= not( feedback XOR nor_detect_0s);
 --serial_in <= feedback XOR nor_detect_0s;
end generate gen_zero;
-------------------------------------------------------------
-------------------------------------------------------------
-- when the 'all zeros' state is not needed
-------------------------------------------------------------
gen_no_zeros: if (all_zeros_state = FALSE) generate
-- Serial Input to the LFSR
 serial_in <= feedback;
end generate gen_no_zeros;
-------------------------------------------------------------
end architecture beh;
-------------------------------------------------------------
 -- EoF --
```

B.6 CÓDIGO DEL DEL TEST BENCH O BANCO DE PRUEBA:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
ENTITY testi IS
END testi;
 ARCHITECTURE behavior OF testi IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT MONTAJE_TOP
    PORT(
        START_out_sal : OUT  std_logic;
        RESET_out_sal : OUT  std_logic;
        DONE_out_sal : OUT  std_logic;
```

```
        MM1_sal : OUT  std_logic_vector(6 downto 0);
        MM2_sal : OUT  std_logic_vector(6 downto 0);
       NUM_GEN_sal : OUT  std_logic_vector(34 downto 0);
Clk : IN  std_logic;
RESULTADO : OUT  std_logic_vector(13 downto 0)
        );
    END COMPONENT;
   --Inputs
   signal Clk : std_logic := '0';
 --Outputs
   signal START_out_sal : std_logic;
   signal RESET_out_sal : std_logic;
   signal DONE_out_sal : std_logic;
   signal MM1_sal : std_logic_vector(6 downto 0);
   signal MM2_sal : std_logic_vector(6 downto 0);
  signal NUM_GEN_sal : std_logic_vector(34 downto 0);
   signal RESULTADO : std_logic_vector(13 downto 0);
   -- Clock period definitions
   constant Clk_period : time := 10 ns;
BEGIN
 -- Instantiate the Unit Under Test (UUT)
   uut: MONTAJE_TOP PORT MAP (
          START_out_sal => START_out_sal,
          RESET_out_sal => RESET_out_sal,
          DONE_out_sal => DONE_out_sal,
          MM1_sal => MM1_sal,
          MM2_sal => MM2_sal,
          NUM_GEN_sal => NUM_GEN_sal,
          Clk => Clk,
          RESULTADO => RESULTADO
```

```
        );
    -- Clock process definitions
    Clk_process :process
    begin
Clk <= '0';
wait for Clk_period/2;
Clk <= '1';
wait for Clk_period/2;
    end process;
    -- Stimulus process
    stim_proc: process
    begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
      wait for Clk_period*10;
      -- insert stimulus here
      wait;
    end process;
END;
```