UNIVERSIDAD NACIONAL DEL NORDESTE

FACULTAD DE CIENCIAS EXACTAS Y NATURALES Y AGRIMENSURA

CARRERA DE INGENIERÍA EN ELECTRÓNICA

MEMORIA DEL TRABAJO FINAL

Diseño e implementación de un filtro CFAR con automatismos para un procesador de señales radar con tecnología SoC-FPGA

Autor: González, Fernando Augusto
Nombre del Autor

Director: Ing. Oscar Guillermo Lombardero
Nombre del Director

Jurados:
Nombre del jurado 1 (pertenencia)
Nombre del jurado 2 (pertenencia)
Nombre del jurado 3 (pertenencia)

Este trabajo fue realizado en la Ciudad Corrientes y Resistencia, entre noviembre de 2019 y agosto de 2020.

Resumen

Acá va el resumen del trabajo. Debe ser lo más breve posible. No más de dos o tres párrafos, de unas cuatro o cinco oraciones cada uno. Leyendo esto debe quedar muy claro en qué consiste el trabajo realizado, por qué el trabajo es importante, por qué el trabajo muestra que el estudiante aplicó correctamente lo aprendido en la Carrera y qué información va a encontrar el lector en esta Memoria.

No usar en este resumen ninguna referencia bibliográfica del tipo [1], ni tampoco notas a pie de página ni siglas que no estén aclaradas como parte de este texto, ni tipografía en negritas, subrayada o cursiva. Dicho de otra forma, el texto en este resumen debe ser escrito de forma tal que si se recorta el mismo y se lo pega en un archivo .txt entonces este conserve su formato y sea perfectamente entendible sin ningún agregado adicional, es decir, quede autocontenido.

Agradecimientos

Agradecimientos personales. **[OPCIONAL]**

No olvidarse de agradecer al tutor.

No vale poner anti-agradecimientos (este trabajo fue posible a pesar de...)

Índice general

Índice de figuras

Índice de Tablas

Dedicado a... [OPCIONAL]

Capítulo 1

Introducción

1.1. Conceptos generales de radares

1.1.1. Radar

El radar (acrónimo del inglés Radio Detection and Ranging), de acuerdo a la definición dada por ENACOM (Ente Nacional de Comunicaciones), es un sistema que permite determinar la localización y/o la velocidad de un objeto a través del uso de radiaciones electromagnéticas. Estas ondas una vez emitidas en cierta dirección, cuando impactan en algún objeto, se reflejan en distintas direcciones. Una onda reflejada regresa al radar y contiene una pequeña parte de la energía emitida originalmente. De esta manera, conociendo la velocidad de propagación de la onda y midiendo el tiempo de retardo, se puede conocer datos del mismo, por ejemplo la posición relativa del objetivo y su velocidad.

En general se utiliza el sistema de coordenadas polares: proporciona el alcance y la orientación de los objetivos encontrados con respecto a la posición de la antena. Cabe mencionar que el rango es la distancia inclinada desde la antena y no la distancia horizontal, como se ilustra en la figura 1.1. El rango inclinado (1) es la hipotenusa del triángulo representado por la altitud de la aeronave y la distancia entre la antena del radar y la trayectoria en tierra de la aeronave (punto (3) en la tierra directamente debajo de la aeronave). En ausencia de información de altitud, la ubicación de la aeronave se trazaría más lejos (2) de la antena que su trayectoria real en tierra.

El patrón de radiación de la antena se ilustra en la figura 1.2 un haz estrecho cuando se ve desde arriba y, con cierta aproximación, puede considerarse como un trapecio si se ve de lado.

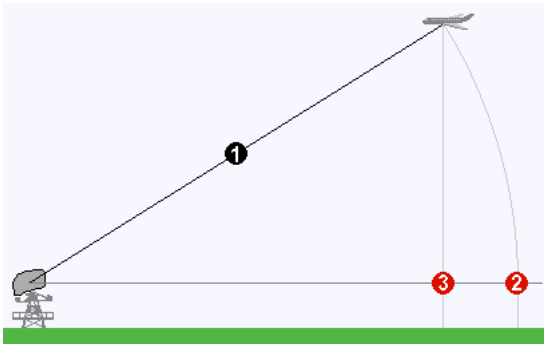


FIGURA 1.1: Rango, altura y distancia a nivel de tierra **[WolfgangW]**.

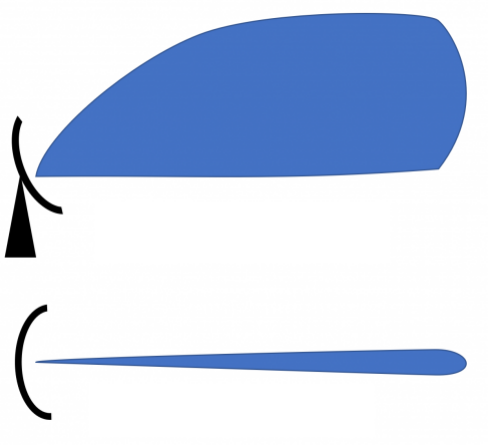


FIGURA 1.2: Patrón de radiación de la antena.

Los radares que identifican objetos mediante la detección de reflexiones que estos producen de señales de radiofrecuencia se denominan radares primarios. Estos utilizan una antena que gira continuamente montada en una torre para transmitir ondas electromagnéticas que se reflejan, o se dispersan, desde la superficie de la aeronave hasta cierta cantidad de millas náuticas desde el radar. El sistema de radar mide el tiempo necesario para que el radar repita el eco y la dirección de la señal. A partir de esto, el sistema puede medir la distancia de la aeronave a la antena del radar y el azimut, o dirección, de la aeronave en relación con la antena.

Existen otro tipo de radares de vigilancia, denominados secundarios, que utilizan una segunda antena de baliza de radar unida a la parte superior de la antena del radar principal para transmitir y recibir datos del área de la aeronave para la altitud barométrica, el código de identificación y las condiciones de emergencia. Se emiten pulsos codificados, para realizar interrogaciones mediante trenes de pulsos. Las aeronaves militares, comerciales y algunas de aviación general tienen transpondedores que responden automáticamente a una señal del radar secundario informando un código de identificación y altitud. Los centros de control de tráfico aéreo utilizan estos datos del sistema para verificar la ubicación de la aeronave dentro de un radio determinado del sitio del radar. El radar secundario también proporciona una identificación rápida de aeronaves en peligro.

Se menciona a continuación algunas ventajas y desventajas de los radares primarios:

- Ventajas
  - El radar primario es el único sensor de vigilancia utilizado en la aviación civil que no requiere ningún equipo a bordo para localizar la aeronave. A diferencia de radar secundario, puede descubrir una aeronave que experimente una falla de transpondedor o un intruso.
  - Se utiliza para captar objetivos de larga distancia. Puede detectar aviones desde una distancia de cientos de kilómetros.
  - Mantiene 360 grados de vigilancia desde la superficie hasta grandes altitudes. Determina la orientación de los objetivos en un área más grande.
- Desventajas

- Cono de silencio. Debido al patrón de radiación, hay una parte del espacio aéreo sobre la antena que no se puede inspeccionar. Este efecto se mitiga colocando una serie de radares de tal manera que el cono de silencio de cada radar quede cubierto por otro radar.
- Los objetivos con el mismo rango de inclinación (en diferentes niveles) son difíciles de distinguir (las señales recibidas se superpondrán). Esto se mitiga combinando el radar primario con un radar secundario que puede reconocer las diferentes aeronaves por sus códigos de transpondedor.
- Límite de rango mínimo. El PSR opera en una frecuencia, lo que significa que no puede emitir y recibir señal al mismo tiempo. Si el objetivo está demasiado cerca de la antena del radar, la señal reflejada puede recibirse antes del final de la transmisión. Si eso sucede, el objetivo no será detectado. Tenga en cuenta que acortar el pulso también reducirá la cantidad de energía emitida, lo que limitará el alcance máximo del radar. Esto se mitiga ajustando la longitud del pulso y la velocidad de rotación de la antena.

1.1.2. Tipos de radares

La tecnología de radar ha experimentado muchos cambios desde su invención. Actualmente existe diversa variedad de sistemas de radar que se pueden clasificar en varias categorías según la finalidad de uso, cantidad de antenas, dependencia del blanco (radares primarios y secundarios mencionados anteriormente), forma de onda, ámbito de aplicación, etc. A continuación se destacan algunos de los sistemas de radar más comunes:

- Radar biestático: Consta de un transmisor y un receptor que están separados por una distancia que es igual a la distancia del objetivo esperado. Un radar en el que el transmisor y el receptor están ubicados en el mismo lugar se conoce como radar monoestático. La mayoría de los misiles tierra-aire y aire-aire de largo alcance emplean el uso de radar biestático.
- Radar de onda continua: En este tipo de sistema, la energía de una onda continua de radio, de frecuencia estable conocida, se transmite y luego se recibe de cualquiera de los objetos que reflejan las ondas. Un radar de onda continua utiliza tecnología Doppler, lo que significa que el radar será inmune a cualquier forma de interferencia de objetos grandes que estén estacionarios o se muevan lentamente.
- Radar doppler: Es una forma especial de radar que emplea el efecto Doppler para producir datos de velocidad sobre un objeto a una distancia determinada. Esto se logra enviando señales electromagnéticas hacia un objetivo y luego analizando cómo el movimiento del objeto ha afectado la frecuencia de la señal devuelta. Esta variación tiene la capacidad de proporcionar mediciones extremadamente precisas del componente radial de la velocidad de un objetivo en relación con el radar. Los radares Doppler tienen aplicaciones en diferentes industrias, incluida la aviación, la meteorología, la salud y muchas otras.
- Radar monopulso: Compara la señal recibida de un solo pulso de radar contra sí mismo con el objetivo de comparar la señal como se ve en múltiples

polarizaciones o direcciones. La forma más común de radar monopulso es la adaptación de un radar de exploración cónico que compara el retorno de dos direcciones para medir directamente la ubicación del objetivo.

- Radar pasivo: Está diseñado para detectar y rastrear objetos procesando reflejos de fuentes de iluminación no cooperativas en el entorno. Estas fuentes incluyen por ejemplo señales de comunicaciones y transmisiones comerciales.
- Radares de instrumentación: Son radares que están diseñados para probar cohetes, misiles, aviones y municiones en campos de prueba gubernamentales y privados. Proporcionan una variedad de información que incluye espacio, posición y tiempo, tanto en tiempo real como en el análisis de postprocesamiento.
- Radar meteorológico: Este radar utiliza ondas de radio junto con polarización horizontal o circular. La selección de frecuencia del radar meteorológico depende de un compromiso de rendimiento entre el reflejo de la precipitación y la atenuación como resultado del vapor de agua atmosférico. Algunos radares meteorológicos están diseñados para utilizar cambios Doppler para medir la velocidad del viento y la polarización dual para identificar los tipos de precipitación.
- Radares cartográficos: Se utilizan para escanear una gran región geográfica en busca de aplicaciones geográficas y de teledetección. Están limitados a objetos relativamente estáticos. Existen algunos sistemas de radar específicos que pueden detectar a los humanos detrás de las paredes gracias a las características reflectantes de los humanos que son más diversas que las que se encuentran en los materiales de construcción.
- Radares de navegación: Poseen longitudes de onda cortas son capaces de reflejarse desde la tierra y las piedras. En su mayoría, son comunes en barcos comerciales y otros aviones comerciales de larga distancia. Hay varios radares de navegación que incluyen radares marinos comúnmente montados en barcos para evitar colisiones y con fines de navegación.

1.1.3. Medición de distancia

Tradicionalmente se ha considerado que estas ondas electromagnéticas se propagan en línea recta en el espacio y esto puede variar ligeramente según las condiciones atmosféricas y meteorológicas. Mediante el uso de antenas de radar especiales, esta energía se puede enfocar en la dirección deseada. De esta forma, se puede medir la dirección (en azimut y ángulo de ubicación) de los objetos que reflejan.

El radar emite pulsos de radio cortos con una potencia de pulso muy alta. Este pulso se concentra solo en una cierta dirección de la antena y se propaga a la velocidad de la luz en esta dirección. Si hay un obstáculo en esta dirección, entonces parte de la energía del pulso se dispersa en todas las direcciones. Una parte muy pequeña también se refleja en el radar. La antena del radar recibe esta energía y un sistema electrónico evalúa la información contenida en esta señal de eco.

El rango está determinado por la diferencia de tiempo del pulso emitido y recibido (la velocidad de propagación es la velocidad de la luz) y la demora se obtiene

del azimut de la antena. La velocidad de rotación de la antena suele estar entre 5 y 12 rpm. Como las distancias de viaje y retorno deben tenerse en cuenta en la medición, se utiliza la siguiente ecuación:

$$R = \frac{c_0 t}{2} \tag{1.1}$$

La distancia se suele indicar en "millas náuticas"(en inglés abreviado NM, Nautical Miles). El factor de conversión es $1NM = 1{,}852km$.

1.1.4. Interferencias

El procesamiento de la señal de radar involucra filtrar distintos tipos de señales indeseadas que se superponen a la señal de eco recibida en el radar. La relación señal a ruido del sistema (Signal to Noise Ratio, SNR, por sus siglás en inglés) determina la capacidad del mismo para sobreponerse a la presencia de estas señales. Cuánto mayor sea la SNR del sistema, se puede aislar mejor los objetivos reales de las señales provenientes de fuentes de ruido del entorno. Las fuentes principales de estos ruidos se describen a continuación:

- Ruido: Es una fuente interna de interferencia que se origina en los componentes electrónicos. Como la potencia del eco recibido por el radar es muy baja, el receptor juega un papel clave en la minimización del ruido. La figura de ruido es una característica que permite conocer el nivel de ruido producido por el receptor. Además de las fuentes internas, la radiación térmica natural del entorno constituye una fuente externa de interferencias, en particular la que rodea al blanco.
- Clutter: Se denomina clutter a aquellos ecos recibidos por el radar que son, por definición, no deseados. Pueden estar causados por objetos del entorno, precipitaciones (lluvia, por ejemplo), tormentas de arena, animales (especialmente pájaros), turbulencias y otros efectos atmosféricos.

1.2. Procesamiento de la señal de radar

La señal de vídeo analógico proveniente del radar por lo general viene con cierta cantidad de ruido y clutter incorporado. Los ecos de clutter son más intensos en zonas cercanas al radar y están influenciados por las condiciones atmosféricas. Se debe realizar un procesamiento sobre ésta señal de video para poder extraer la información sobre los blancos que circulen por el espacio aéreo cubierto por dicho radar. Se emplea para ello un procesador monoradar.

1.2.1. Procesador monoradar

El procesador monoradar tiene por misión la obtención de un dato, denominado plot, por cada blanco detectado por el radar. Es por ello que se encuentran ubicados en la cadena de proceso después de los procesos de filtrado y detección. Normalmente los plots se proporcionan a través de un canal serie o sobre red en formato digital, donde cada blanco tiene asociados una serie de parámetros como la distancia, azimut, elevación (para radares tridimensionales), potencia, nivel de confianza, velocidad, etc. El procesador extrae estos parámetros a partir de las señales de vídeo proporcionadas por etapas previas. Utilizando estas señales, el

procesador monoradar tiene que aglutinar la información procedente de un mismo blanco, debido a que el radar, de cada blanco, recibe un número de hits (reflejo de la emisión de un ciclo radar (PRF)) durante varios PRT consecutivos. El número de hits varía dependiendo del ancho del lóbulo de radiación, de la velocidad de giro de la antena, de la frecuencia de repetición de pulsos (PRF, Pulse repeticion frequency) del radar, del nivel de señal, de las condiciones meteorológicas, etc. Debido a esto el procesador monoradar, aplicando los algoritmos y procesos necesarios (ventana deslizante, por ejemplo), determinará los parámetros de azimut y distancia del blanco respecto a la estación Radar.

Por tanto, la función del procesador monoradar es, correlacionar toda la información procedente del blanco, agruparla y extraer la información útil (distancia, azimut, nivel de potencia, etc.), y proporcionar como salida la información correspondiente a un solo blanco, codificarlo en formato digital y enviarlo a consolas (locales o remotas) o a Centros de Fusión Remotos, donde se recibe este tipo de información de varios radares para presentar un solo blanco visto por varios radares de forma simultánea. De esta forma se pueden visualizar en centros de control remotos coberturas muy amplias.

1.3. Técnicas CFAR

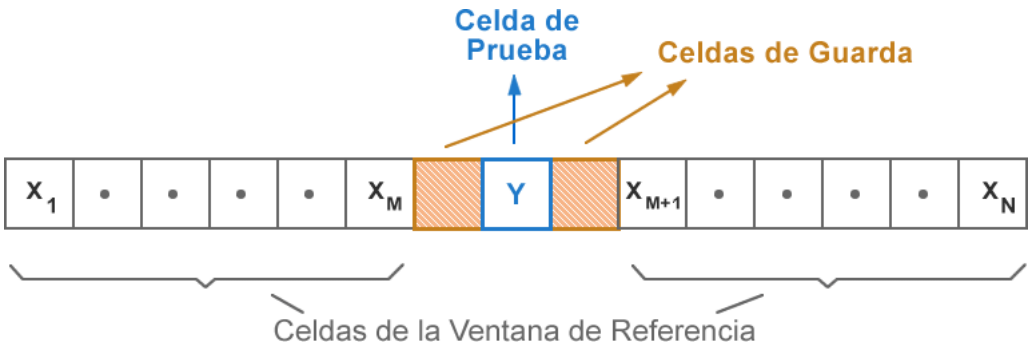
Para limitar la cantidad de plots generados es necesario establecer un umbral de comparación a partir del cual se considera a un eco recibido como válido. Para limitar la cantidad de plots generados, no es suficiente establecer un umbral fijo de comparación para la señal recibida por el radar. Esto es debido a que la distribución de clutter en el espacio aéreo es variante con el tiempo y espacio, y con las condiciones atmosféricas. Por tanto la predicción de dicha distribución se vuelve difícil y es necesario emplear técnicas más avanzadas para mejorar la relación señal a ruido.

El presente trabajo adopta una solución basada en la técnica CFAR (acrónimo inglés de Constant False Alarm Rate), el cual emplea un umbral de comparación variable con el objetivo de tener una tasa de falsa alarma constante, que se adapta a las condiciones presentes en una determinada zona del espacio aéreo.

En general el sistema CFAR basa su funcionamiento en almacenar datos de vídeo en un cierto entorno para caracterizar una distribución de ruido en ese recinto (podemos considerar ruido al clutter y a otros tipos de interferencias). Esto luego es usado para evaluar si el eco recibido en ese entorno puede ser considerado como un blanco o no, dependiendo de la relación existente entre el ruido medio y la amplitud de tensión del eco. Como es de esperar, a mayor cantidad de datos de video almacenado, mejor caracterización del ruido. Para realizar esta función, el CFAR dispone de una celda bajo testeo, un cierto número de celdas, denomindas celdas de entrenamiento o de referencia, que almacenan el dato de amplitud del video digital en el entorno de la celda bajo testeo y una etapa que opera sobre estos datos para generar un umbral cuyo valor se comparará con la celda bajo testeo.

1.3.1. CA-CFAR

Existen diferentes técnicas CFAR. Empleando la técnica CFAR con promediado de celdas (CA-CFAR, por Cell-Averaging CFAR) se realiza un promediado de



celdas llamadas celdas de referencia que se ubican a ambos lados de la celda bajo testeo. Es una técnica de aplicación general, ya que sirve para la mayoría de los casos. El ruido estimado se puede calcular como:

$$P_n = \frac{1}{N} \sum_{m=1}^{N} x_m \tag{1.2}$$

donde N es la cantidad de celdas utilizadas y x_m es el valor de la muestra en cada celda. Si x_m resulta ser la salida de un detector de ley cuadrática, entonces P_m será la potencia de ruido estimada.

Para evitar introducir en la promediación la potencia proveniente de la celda bajo testeo, se establece alrededor de ella unas celdas de guarda, cuyo valor almacenado no se computa, sólo se transfiere a las celdas siguientes, como se ilustra en la figura **??**.

1.3.2. Técnicas CA-CFAR relacionadas

Existen variaciones inmediatas del algoritmo CA-CFAR, en las cuales en lugar de considerar en la promediación de ruido los valores de ambas celdas de referencia, sólo se considera uno de ellos. Este es el caso de los algoritmos GOCA-CFAR (greatest of Cell-Averaging - CFAR) y SOCA-CFAR (smallest of Cell-Averaging - CFAR). Con estas técnicas se comparan los promedios parciales de las celdas de referencia y se considera el mayor o el menor respectivamente.

1.3.3. Otras técnicas CFAR

Existen otros algoritmos no considerados en este trabajo, que incorporan otras herramientas estadísticas, como ser OS-CFAR (por Ordered Statistic CFAR) y (TM-CFAR, por Trimmed Mean - CFAR):

- OS-CFAR En la técnica CFAR de estadística ordenada, OS-CFAR, se ordenan las muestras de las celdas de referencia en orden ascendente. El elemento $n/2$ es la mediana de los datos. El $k - simo$ elemento de la lista ordenada representa un determinado nivel de interferencia y el umbral es un múltiplo de este valor.
- TM-CFAR La técnica CFAR de constante media recortada (TM-CFAR), es una extensión de OS-CFAR en el que las celdas ordenadas en la ventana de referencia se recortan desde el extremo superior e inferior. El umbral es formado sumando las celdas restantes. En algunos TM-CFAR solo las celdas

del extremo superior (celdas con mayor potencia) se descartan, por lo que si hay varios objetivos presentes en la ventana de referencia, el recorte elimina el efecto de la interferencia objetivo.

1.4. FPGA

Las FPGA (por sus siglas en inglés, Field Programmable Gate Array) son una matriz de compuertas lógicas programables cuyos elementos se componen un conjunto de bloques lógicos configurables, conectados mediante conexiones programables. Se crearon en el año 1984 como evolución de los dispositivos lógicos programables complejos (CPLD) y comparten ciertas similitudes con los Circuitos Integrados de Aplicación Específica, ASIC (acrónimo en inglés), sin embargo la diferencia con respecto a este viene marcada por su característica de ser reprogramable. También se pueden realizar comparaciones con otros dispositivos lógicos similares como GPP, DSP, ASIC estructurado, etc. Las FPGA se emplean en campos como la Medicina, procesamiento de imágenes y video, Aeroespacio y defensa, comunicaciones cableadas e inalámbricas, audio, entre otros.

A diferencia con un microcontrolador, las FPGA no tiene una función específica incorporada, sino que poseen recursos para reproducir un circuito dentro del chip. Esto es posible debido los elementos lógicos individuales denominados CLBs (Configuble Logic Module) o ALMs (Adaptative Logic Module), como se ilustra en la figura **??**. Las FPGA modernas incluyen en estos bloques elementos básicos como Flip Flops, Look-Up-Tables, Digital Signal Prossencing Blocks, relojes dedicados, PLLs, etc. Tambien tiene bloques de entrada y salida que se conectan con pines individuales de las ALMs, que pueden ejercer no sólo funciones de buffers sino también estados de alta impedancia.

Cabe mencionar que son dispositivos volátiles es decir que no tienen memoria, por sí solos no pueden almacenar su configuración de conexión y por tanto cuando se desconecta la fuente de energía se pierde esa configuración. Generalmente se utiliza una memoria por fuera de la FPGA para almacenar esa configuración.

1.4.1. SoC-FPGA

Los procesadores y las FPGA son dispositivos importantes en la mayoría de los sistemas embebidos. Los procesadores poseen la funcionalidad de gestión de alto nivel y las FPGA las estrictas operaciones en tiempo real, procesamiento de datos o funciones de interfaz.

Los dispositivos SoC-FPGA empezaron a producirse para brindar una alternativa de mayor desempeño y menor consumo y coste a sistemas que utilizan por separado una FPGA y un microprocesador o DSP. En los dispositivos SoC-FPGA se integra la arquitectura del procesador y FPGA en un sólo dispositivo. Esto brinda mayor integración, menor consumo de potencia, menor tamaño de placa y un mayor ancho de banda de comunicación entre el procesador y la FPGA.

Como las señales entre el procesador y la FPGA ahora residen en el mismo chip, la comunicación entre los dos consume sustancialmente menos potencia en comparación con el uso de chips separados. Además la integración de miles de conexiones internas entre el procesador y la FPGA proporciona un ancho de banda mucho mayor y una latencia más baja que usar ambos dispositivos por separado.

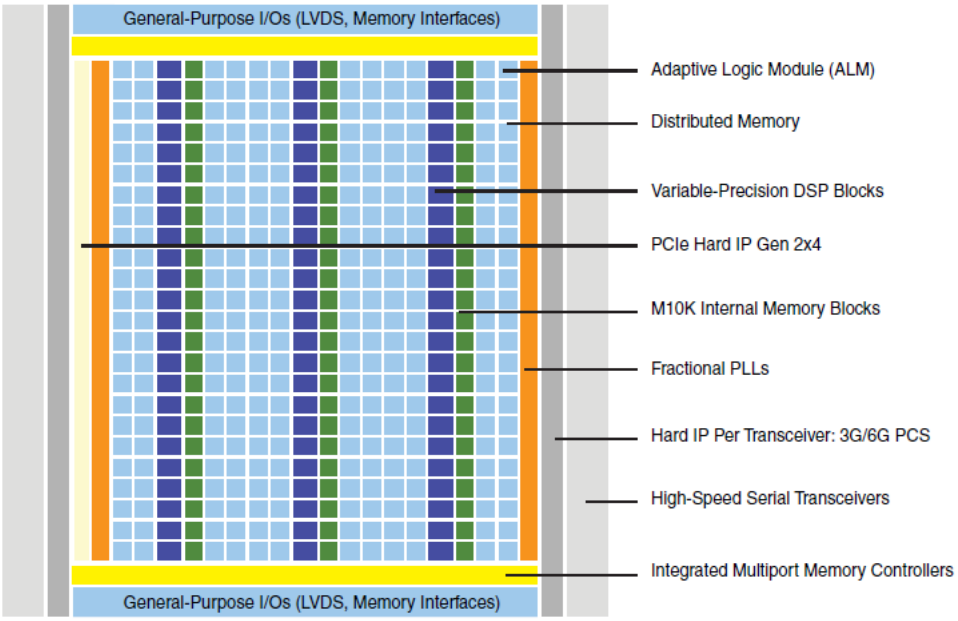


FIGURA 1.3: Composición de una FPGA Cyclone V de Altera

El paso siguiente es incluir el chip dentro de una placa de circuito impreso (PCB, por el acrónimo inglés Printed Circuit Board). Para ellos hay empresas que ofrecen en el mercado PCB con diferentes tecnologías FPGA y variados periféricos. Un ejemplo es la placa tipo ADC-SoC, que incorpora una SoC-FPGA con un convertidor analógico a digital (ADC, por Analog to Digital Conveter) de alta velocidad.

1.5. Bloques básicos de una FPGA

En el presente trabajo se utilizó una FPGA de la firma Intel. A continuación se describe los componentes básicos que por generalmente incorpora una FPGA de esta empresa.

1.5.1. ALM

El bloque básico de una FPGA Cyclone V se denomina Adaptative Logic Module (ALM), el cual se ilustra en la figura 1.4. Un ALM contiene cuatro registros programables, con los siguiente puertos:

- Data
- Clock
- Synchronous and asynchronous clear
- Load

Las señales globales, los pines de entrada y salida de propósito general (GPIO)o cualquier señal interna de lógica pueden comandar las señales de control de clock y de clear de un registro ALM. Además los pines GPIO o la lógica interna controlan la señal de activación del reloj. Para las funciones combinacionales, los registros se omiten y la salida de la LUT es conducida directamente a las salidas de un ALM.

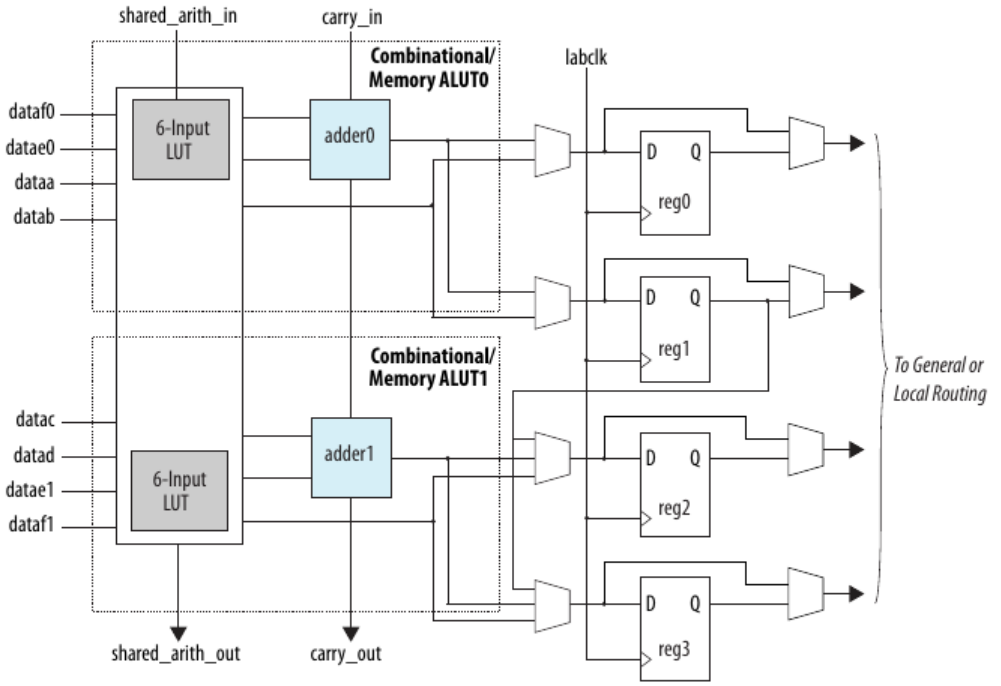


FIGURA 1.4: Diagrama en bloques de un ALM

1.5.2. Bloques de memoria

Posee además dos tipos de memorias embebidas, M10K y MLAB:

- M10K: Las de tipo M10K son de 256 hasta 8000 bits de profundidad aproximadamente, están diseñadas para arrays extensos de memoria con puertos independientes.
- MLAB: Las de tipo MLAB sin embargo son de hasta 32 bits de profundidad y están optimizadas para la implementación de registros de desplazamientos, buffers tipo FIFO anchos y líneas de retardo. Cada MLAB se compone de diez ALMs y hasta el 25 % de las ALM se pueden configurar como memoria distribuida usando los bloques MLABs.

1.5.3. Relojes y PLLs

Posee 16 redes de reloj globales de 550 Mhz cuya arquitectura está basada en estructuras de Intel tipo globales, cuadrantes y periféricos. Esta estructura de reloj es compatible con pines de entrada de reloj dedicados y PLL fraccionales. Una característica importante del funcionamiento de los relojes es que en la herramienta Quartus identifica las secciones de la red de reloj sin usar y las desconecta, mejorando el consumo.

1.5.4. Hard Processor System

La FPGA posee integración con un sistema de procesamiento dentro del mismo chip. Este último se compone de una unidad de microprocesador con procesador córtex-A9 de ARM, controladores de memoria flash, soporte de periféricos y de interfaz, PLLs, interconexión SDRAM L3, entre otros.

Las partes HPS y FPGA del dispositivo SoC-FPGA son diferentes. El HPS arranca desde cualquiera de las múltiples fuentes de arranque, incluida la estructura FPGA y los dispositivos flash externos, y la FPGA se configura a través del HPS o cualquier fuente externa compatible con el dispositivo.

1.6. Niveles de abstracción en el diseño digital

En el ámbito del diseño digital se pueden distinguir diferentes niveles de abstracción que se distinguen por su impacto en el diseño y complejidad. Las decisiones a nivel conceptual son menos complejas pero tienen un impacto importante en el diseno final y a su vez la complejidad del diseno crece a medida que avanzamos en el ciclo del diseno. Podemos distinguir, por ejemplo, entre los siguientes niveles:

- Nivel conceptual o de sistema: Es el nivel más alto en el diseño. Consiste en captar los requerimientos y especificaciones del sistema y, a partir de los mismos, reducir opciones de diseño y algoritmos.
- Nivel algorítmico: Consiste en implementar los algoritmos en lenguajes de alto nivel con la intención de determinar resultados sobre la viabilidad del diseño. Por ejemplo implementar una simulación usando lenguaje Python, lenguaje C/C++ o entorno MATLAB.
- Nivel de transferencia de registros: En este nivel, también denominado RTL (por el acrónimo inglés, Register Transfer Level), se considera que el algoritmo ya está decidido y se procede a describir los detalles de cómo se mueven los datos entre y dentro de los subsistemas, además de cómo se manipulan los datos en función de las entradas del sistema. Su comportamiento se describe mediante un lenguaje de descripción de hardware (HDL, por el acrónimo de Hardware Description Language).
- Nivel de compuertas: El codigo escrito en RTL se sintetiza para la implementacion a nivel de compuerta. El proceso de síntesis toma el RTL y lo traduce, elaborando una lista de conexiones a nivel de puerta (netlist). Para la síntesis logica, el usuario especifica restricciones de diseno y la tecnologa de destino en forma de una biblioteca de celdas estandar (cell library). La biblioteca tiene compuertas logicas basicas estandar, como AND y OR, o macroceldas como sumadores, multiplicadores, flip-flops, multiplexores, etc. La herramienta convierte completamente el diseno descrito en RTL en un diseno que contiene celdas estandar. Para mapear de forma optima la descripcion de alto nivel en hardware real, la herramienta realiza varios pasos:
    - Convierte primero la descripcion RTL en logica booleana no optimizada.
    - Realiza varias transformaciones para optimizar la logica sujeto a restricciones de usuario, donde esta optimizacion es independiente de la tecnologa de destino.
    - Finalmente, la herramienta mapea la lógica optimizada a celdas estándar específicas de la tecnología.

- Nivel de transistor Es el nivel más bajo de abstracción, se describe el funcionamiento de las puertas y registros básicos utilizando transistores, cables y otros componentes eléctricos como resistencias y condensadores.

1.7. Lenguaje de descripción de hardware

Los lenguajes de descripción de hardware (HDL) se utilizan a nivel RTL para describir la estructura y comportamiento de sistemas digitales en forma textual. No son lenguajes de programación ya que a diferencia de estos, los HLDs pueden manejar múltiples procesos paralelos (como flip-flops y sumadores) que se ejecutan automáticamente de forma independiente entre sí. Cualquier cambio en la entrada del proceso activa automáticamente una actualización en la pila de procesos. En el proceso de diseño es importante tener en cuenta que cada línea de código representa uno o más componentes de hardware.

El uso de un lenguaje de descripción de hardware (HDL) para diseñar en dispositivos FPGA tiene las siguientes ventajas:

- Enfoque de arriba hacia abajo (top-down) para proyectos grandes. Este enfoque para el diseño de sistemas funciona bien para grandes proyectos que requieren que muchos diseñadores trabajen juntos. Una vez que el equipo de diseño determina el plan de diseño general, los diseñadores individuales pueden trabajar de forma independiente en secciones de código separadas.
- Simulación funcional al principio del flujo de diseño. Probar la funcionalidad del diseño antes de que se implemente en el nivel RTL o en el nivel de puerta permite realizar los cambios necesarios desde el principio.
- Síntesis de código HDL para puertas. Sintetizando la descripción del hardware para apuntar a la implementación del dispositivo FPGA:
    - Disminuye el tiempo de diseño al permitir una especificación de diseño de mayor nivel, en lugar de especificar el diseño a partir de los elementos base del dispositivo FPGA.
    - Reduce los errores que pueden ocurrir durante una traducción manual de una descripción de hardware a un esquema de diseño.
    - Permite que la herramienta de síntesis aplique técnicas de automatización (como los estilos de codificación y la inserción automática de entradas y salidas) durante la optimización del código original. Esto se traduce en una mayor optimización y eficiencia.
- Permite probar diferentes implementaciones de diseño al principio del flujo de diseño. Utilizando la herramienta de síntesis para realizar la síntesis lógica y la optimización en puertas. Dado que el tiempo de síntesis es corto, se pueden explorar diferentes posibilidades arquitectónicas en el nivel de transferencia de registros (RTL).
- Reutilización del código de nivel de transferencia de registro (RTL). Se puede reorientar el código de nivel de transferencia de registro (RTL) a nuevos dispositivos FPGA con mínimos cambios.
-

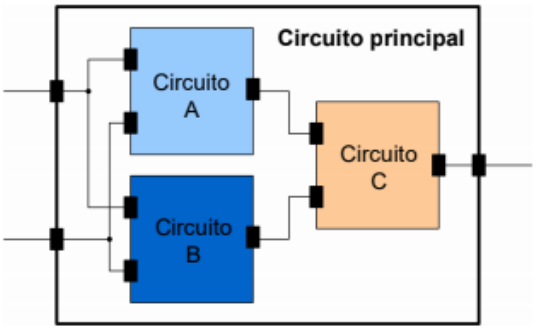


FIGURA 1.5: Ejemplo de subsistema

Los lenguajes HDL soportados por la IEEE son Verilog-HDL y VHDL. VHDL es un lenguaje que inicialmente usaban los contratistas del Departamento de Defensa de EE.UU; ahora se usa comercialmente y en universidades de investigación. Verilog nació como un HDL exclusivo, promovido por una compañía llamada Cadence Data Systems, pero Cadence transfirió el control de Verilog a un consorcio de empresas y universidades llamado Open Verilog International (OVI). Actualmente OVI y Open VHDL Internacional se unieron para formar Accellera. Para este proyecto se empleó ambos lenguajes, los cuales se describirán a continuación.

1.7.1. Aspectos básicos del lenguaje VHDL

Un sistema digital está descrito por sus entradas y sus salidas y la relación que existe entre ellas. VHDL tiene una estructura que separa la declaración de puertos de entrada y de salida por un lado y por el otro la descripción del comportamiento del componente. En la entidad (entity), se declaran los puertos de entrada y salida a utilizar, y en la sección arquitectura (architecture) se describe el comportamiento de la entidad, como se ilustra en la figura 1.6. Cada arquitectura tiene asociada una entidad. Si es necesario, se pueden declarar bibliotecas a utilizar, por ejemplo para cálculo aritmético o para operación entre señales lógicas.

Entidad de diseño

La entidad de diseño es la principal abstracción en VHDL. Una entidad puede representar un sistema entero, un subsistema,una plaqueta, un chip, una macrocelda, una compuerta lógica o cualquier nivel de abstracción que se encuentre entre los antes mencionados.

Una entidad de diseño también puede describirse en términos de componentes interconectados. Cada componente de una entidad de diseño puede estar vinculada a una entidad de diseño de nivel inferior para definir la estructura o el comportamiento de ese componente, como se ilustra en la figura 1.5. La separación de una entidad de diseño en componentes y la interconexión de esos componentes a otras entidades de diseño que pueden separarse de manera similar, da como resultado una jerarquía de entidades de diseño representando un diseño completo. Esta colección de entidades de diseño se denomina jerarquía de diseño. El bloque superior, denominado top-level, es un bloque externo y los bloques anidados en la jerarquía son bloques internos. El uso de jerarquías permite crear código reutilizable y crear componentes que realizan una función específica.

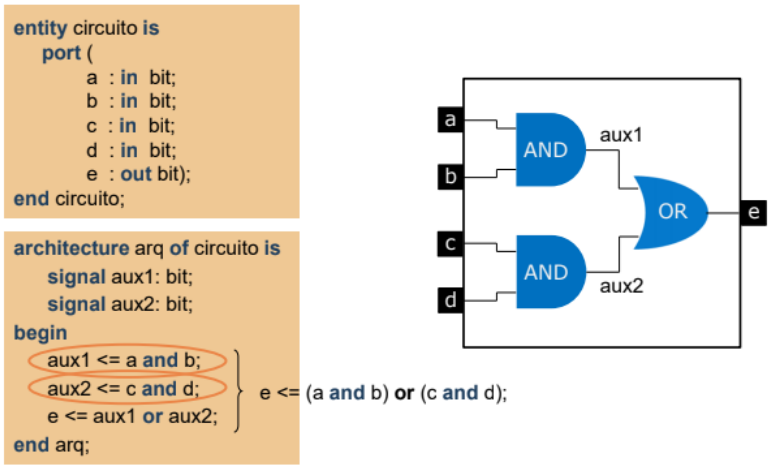


FIGURA 1.6: Ejemplo de un componente descrito con VHDL.

```
module FA(
    input a,
    input b,
    input c_in,
    output sum,
    output c_out);
    assign {c_out, sum}
    = a+b+c_in;
endmodule
```

FIGURA 1.7: Ejemplo de un componente descrito con Verlog.

Arquitectura de diseño

La arquitectura define el cuerpo de una entidad de diseño. Especifica las relaciones entre las entradas y las salidas de esta, y puede estar expresada en términos de estructura, flujo de datos o comportamiento.

Dentro de la arquitectura se pueden crear las señales internas a esa entidad, que efectúan el proceso de las señales de entrada para proporcionar un cierto resultado en los puertos de salida. Para ello se utilizan procesos, asignaciones, instancias de componentes, registros y operaciones lógicas.

1.7.2. Verilog

Es un lenguaje de descripción de hardware que fue diseñado con una sintaxis similar a C. Puede operar a nivel RTL y de compuertas. El diseño se estructura mediante módulos,como se ilustra en la figura 1.7, que incluyen tanto declaración de puertos como el comportamiento.

Hay similitudes y diferencias en comparación con VHDL, sin embargo al parecer no hay una clara ventaja de usar uno u otro en términos de implementación.

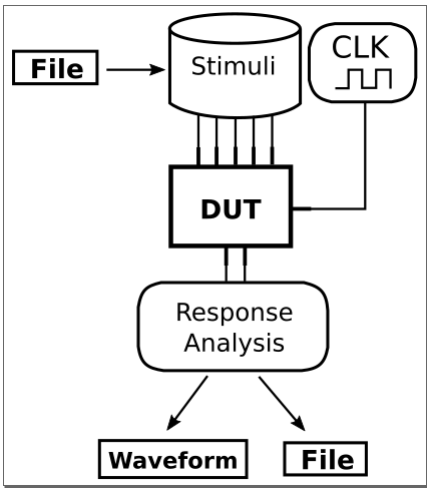


FIGURA 1.8: Diagrama en bloques de un test bench

1.8. Flujo de diseño

El diseño de un circuito digital involucra diferentes fases que se describen a continuación:

1.8.1. Diseño

Es la primer etapa del proceso. Consiste en ejecutar una toma de decisiones sobre el diseño a implementar y la manera de hacerlo. Esto implica tanto como determinar el tipo de placa, lenguaje HDL y herramientas a utilizar así como establecer jerarquías en el diseño y elaborar las especificaciones que deben cumplirse.

1.8.2. Simulación

Se utiliza una computadora para representar la estructura y comportamiento del sistema lógico digital. Mediante la simulación lógica, se interpreta la descripción HDL para predecir el comportamiento del hardware. De esta manera se pueden detectar errores en el funcionamiento del circuito antes de que se fabrique físicamente. Estos errores son corregidos modificando las sentencias del lenguaje empleado y sometiendo el diseño a un proceso de simulación. Una de las formas más usadas para simular diseños es el banco de pruebas (Testbench), que proporciona una forma gráfica de producir y visualizar las formas de onda de las señales de entrada y salida.

El testbench no posee puertos de entrada ni de salida, sino señales de estimulación que se conectan a cada puerto de entrada del dispositivo bajo testeo o DUT (por el acrónimo inglés Device Under Test) y señales de observación que se conectar a los puertos de salida para observar la respuesta del DUT a esos estímulos, como se muestra en la figura 1.8.

Así, el proceso de simulación se resume en las siguientes etapas:

1. Diseño del componente con HDL
2. Diseño del marco de pruebas
3. Verificación

4. Corrección del componente

1.8.3. Síntesis

En este proceso se emplea una herramienta de síntesis, proporcionada en general por el fabricante del chip, que elabora una lista de primitivas y sus interconexiones (netlist) a partir del diseño descrito en mediante HDL. La síntesis depende del dispositivo utilizado y de los recursos lógicos de los que disponga; diferentes dispositivos pueden implementar una misma función de distintas formas sin cambiar la funcionalidad del diseño.

El grado de optimización del proceso de síntesis a la hora de convertir el código HDL al un circuito equivalente, depende de los siguientes factores:

1. La descripción del circuito. Este punto es el más importante porque impacta en los recursos a utilizar y en las directivas a incluir. En la descripción además de decir qué función realiza el circuito, se describe la forma en la que debe realizarlo. Cuando se describe el funcionamiento de un circuito, existen muchas formas de hacer las mismas operaciones, y todas ellas darán lugar a distintas formas de implementación.
2. Los recursos disponibles en el dispositivo seleccionado. Los recursos afectan a la forma en que las funciones descritas son interpretadas e implementadas en los bloques lógicos existentes en el dispositivo. Por ejemplo, un circuito que realice una división entre un número que no sea potencia de dos no podría hacerse en ciertos dispositivos, que no cuentan con esta característica.
3. Las directivas de síntesis seleccionadas por el diseñador. Las directivas de síntesis son aquellos parámetros que se configuran para que la herramienta de síntesis los tenga en cuenta en el proceso de implementación. Por ejemplo, incluir restricciones de tiempo a ciertas señales permite que la implementación de los bloques que se conecten a dichas señales se realice de manera de ubicarlos en una sección del chip que permita que esta restricción se cumpla. En caso de que no se cumpla, la herramienta dará un error de síntesis para comunicar que el diseño debe ser modificado.

1.8.4. Implementación

La implementación, también denominado como Place And Route (P & R)en las FPGAs, consiste en situar el diseño sintetizado usando las celdas lógicas del dispositivo. La implementación transforma las Netlist de nivel medio creadas en la síntesis, y las mapea en la superficie de la FPGA, usando la lógica y los recursos internos disponibles. El archivo final es un Bitstream (.bit) que será el fichero final que se cargará en la FPGA.

1.8.5. Configuración

La configuración incluye la transferencia del fichero bitstream a la FPGA. Éste puede residir en una memoria no volátil como una PROM, o dentro de la FPGA (Muchas FPGA vienen con una memoria interna capaz de mantener el fichero

de configuración). El bitstream puede ser cargado en la FPGA mediante programación JTAG, a través de un procesador, microcontrolador u otro dispositivo externo.

Capítulo 2

Materiales y métodos

El presente trabajo formó parte de un proyecto más grande. Consistió en la realización de dos sistemas, CFAR y configurador, los cuales eran subsistemas de un Procesador Monoradar.

La partición del diseño se realizó teniendo en cuenta las distintas funciones a realizar. Para este proyecto, el sistema CFAR se diseñó para generar el umbral adaptativo propio de la técnica mencionada. Luego, teniendo en cuenta que el clutter se concentra en general en las inmediaciones del radar y que se podría tener mejor desempeño en el filtrado admitiendo distintos factores de escala CFAR en distintas zonas de la cobertura, se diseño el sistema configurador para poder sectorizar la cobertura en un determinado número de sectores independientes. Además, se incorporó al configurador ciertas funciones para realizar un ajuste automático en el valor de escala CFAR para cumplir una determinada cantidad de presencia requerida por el usuario.

Para la realización del presente trabajo se utilizó:

- Placa ADC-SoC de TerasIC.
- Software Quartus Prime.
- Software ModelSim.
- Software VHDL Style Guide.
- Sistema de control de versiones GIT.
- Repositorio remoto en GitHub.
- Procesador monoradar preexistente.
- Simulador de radar.

En las secciones siguientes se menciona la utilización de las mismas.

2.1. Placa ADC-SoC de TerasIC

Se utilizó una placa tipo SoC-FPGA de la firma TeraSiC. Esta placa posee un circuito de conversión analógica a digital que utiliza conectores SMA como interfaz de entrada y proporciona dos canales de conversión, cada uno de 14-bits de resolución y una frecuencia de muestreo de hasta 150 MSPS (Megasamples per Second).

El siguiente hardware se proporciona en la placa:

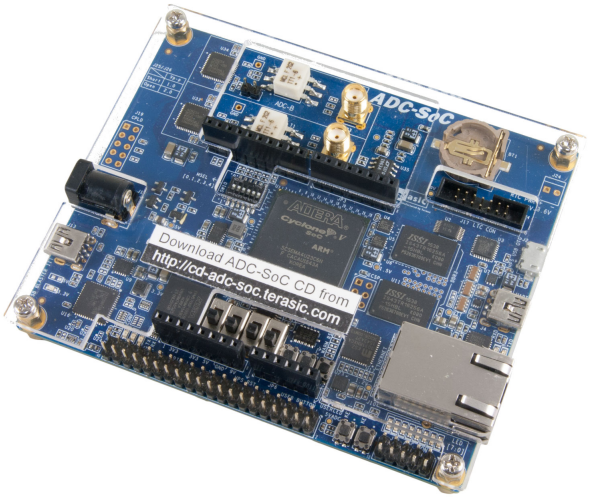


FIGURA 2.1: Placa ADC-SoC de TerasIC

- FPGA
  - Dispositivo Altera Cyclone V.
  - Dispositivo de configuración en serie.
  - USB-Blaster II integrado para programación; Modo JTAG
  - 2 pulsadores
  - 4 interruptores deslizantes
  - 8 LED de usuario verdes
  - Tres fuentes de reloj de 50 MHz del generador de reloj
  - Un cabezal de expansión de 40 pines
  - Un encabezado de expansión Arduino (compatibilidad con Arduino Uno R3) donde se puede conectar los 'shields' Arduino.
  - Un encabezado de expansión de entrada analógica de 10 pines (compartido con la entrada analógica Arduino).
  - Convertidor A / D, interfaz SPI de 4 pines con FPGA
  - Dos convertidores AD de 14 bits con 150 MSPS (megamuestras por segundo)
- HPS (Hard Processor System)
  - Procesador ARM Cortex-A9 de doble núcleo de 925 MHz
  - 1GB DDR3 SDRAM (bus de datos de 32 bits)
  - 1 Gigabit Ethernet PHY con conector RJ45
  - Puerto USB OTG, conector USB Micro-AB
  - Toma de tarjeta micro SD
  - Acelerómetro (interfaz I2C + interrupción)
  - UART a USB, conector USB Mini-B

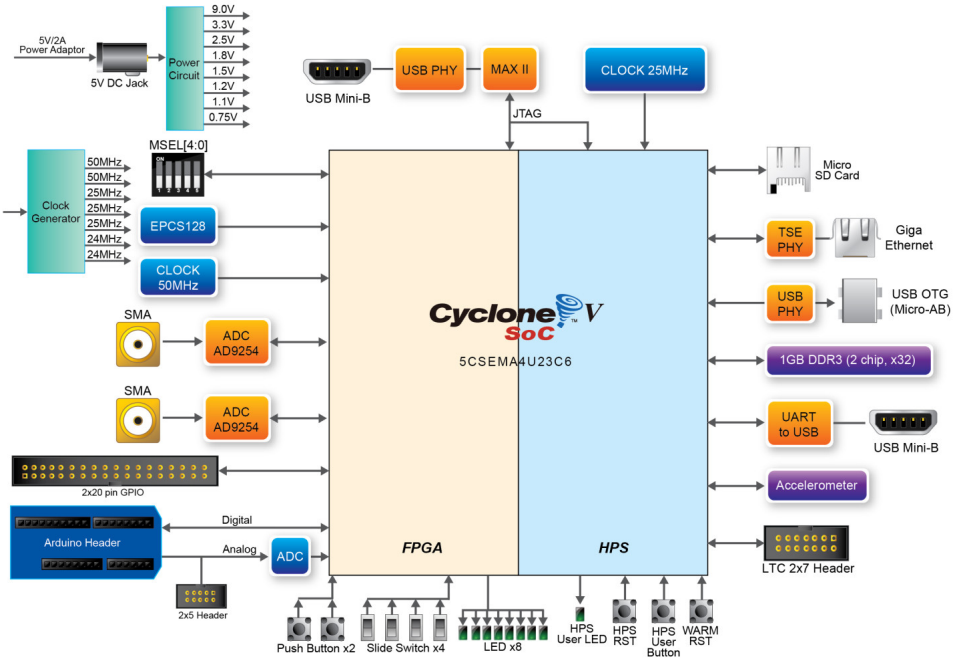


FIGURA 2.2: Diagrama en bloques de la placa ADC-SoC de Terasic

- Botón de reinicio en caliente y botón de reinicio en frío
- Un botón de usuario y un LED de usuario
- Cabecera de expansión LTC 2x7
- RTC integrado (reloj en tiempo real)

En el chip Cyclone V de Altera (propiedad de Intel) se integra una FPGA y un HPS unidos por un puente HPS. Por defecto, la tarjeta micro SD tiene instalado el SO Linux Yocto Poky 8.0, el cual permite correr programas compilados en lenguaje C/C++ entre otros.

2.2. Quartus Prime

Como herramienta de compilación se utilizó *Quartus Prime*, software producido por Altera para el análisis y la síntesis de diseños realizados en HDL. Permite compilar diseños, realizar análisis temporales, examinar diagramas RTL y configurar el dispositivo de destino con el programador.

Quartus proporciona herramientas para trabajar en diferentes fases del diseño en FPGA como la creación del diseño, el agregado de restricciones, la compilación, el análisis de tiempos y la configuración de la FPGA con un programador. Se describen las características de *Quartus* a continuación:

- Creación del diseño: Es posible diseñar en nivel RTL con lenguajes VHDL, Verilog o SystemVerilog. Además posee una herramienta llamada *Platform Designer* que crea automáticamente la lógica de interconexión a partir de la conectividad de alto nivel que se especifique. La automatización de interconexión elimina la laboriosa tarea de especificar conexiones HDL a nivel

del sistema. De esta manera se pueden especificar los requisitos de la interfaz e integrar componentes de IP dentro de una representación gráfica del sistema. En el presente proyectó se utilizó *Platform Designer* (Diseñador de plataformas) para configurar el puente HPS.

- Permite el agregado de restricciones al diseño con herramientas denominadas *assigment editor* (Editor de asignaciones) y *pin planner* (Planificador de pines).
- Permite compilar el diseño, compuesto por las siguientes etapas:
  - Análisis y Síntesis Se evalúa el código para detectar la correcta escritura del mismo y se verifica que el mismo sea sintetizable, es decir que pueda ser implementado con lógica.
  - *Fitter* (Consiste en la colocación y Ruteo, *Place & Route*): Se asignan recursos específicos de la FPGA para cumplir con el diseño. En etapa etapa además la herramienta utiliza la información sobre las restricciones de tiempo evaluar qué celdas utilizar (en términos de propagación de señales). Por otro lado, luego de la colocación, se emplean técnicas de optimización en la asignación de recursos.
  - Generación de archivos de programación: Se generan los archivos necesarios para programar la FPGA.
  - Análisis de tiempo: Verifica mediante la herramienta *Timing Analyzer* que se cumplan con las restricciones de tiempo especificadas en el diseño. Por ejemplo los referidos a la generación de señales de reloj internas, derivadas de fuentes de señales de reloj externas.
- Posee una herramienta llamada *Signal Tab Logic Analyzer* que permite realizar un debug mientras el sistema se ejecuta en la FPGA.

Visores de netlist

A medida que los diseños de FPGA crecen en tamaño y complejidad, la capacidad de analizar, depurar, optimizar y restringir el diseño es fundamental. *Quartus* posee visores llamados *RTL Viewer*, *State Machine Viewer* y *Technology Map Viewer* (visores de nivel de transferencia de registros, de máquina de estados y de mapa tecnológico, respectivamente). Cada uno permite ver representaciones esquemáticas de la estructura interna del diseño. Cada visor muestra una vista única del *netlist* (lista de redes) que se producen en diferentes etapas de la compilación, como se ilustra en la figura 2.3. Se describen a continuación:

- *RTL Viewer*: Permite ver un esquema de la lista de conexiones de diseño después de la etapa de Análisis y elaboración, y de la extracción de la lista de conexiones, pero antes de la síntesis y las optimizaciones de ajuste. Esta vista no es la estructura final del diseño, porque no se incluyen todas las optimizaciones; en cambio, es la vista más cercana posible al diseño original. Si el diseño utiliza síntesis integrada, esta vista muestra cómo *Quartus* interpreta los archivos de diseño.
- *State Machine Viewer* Proporciona una vista de alto nivel de las máquinas de estados finitos en el diseño y muestra la estructura interna de las máquinas

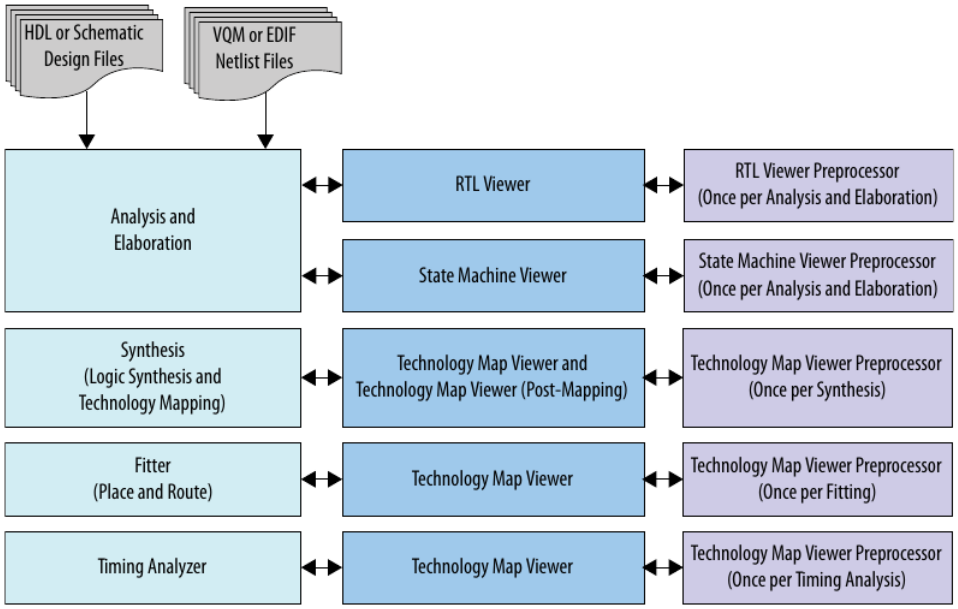


FIGURA 2.3: Ubicación de los visores de netlist en el flujo de diseño de Quartus

de estados en el diseño, incluida una vista más detallada de la entrada y salida de los nodos de estado individuales. También muestra las transiciones de los nodos en formato de tabla.

- *Technology Map Viewer* Permite ver un esquema de bajo nivel específico de la tecnología de la lista de redes de diseño después del ajuste o después de Análisis y síntesis. Se puede acceder a la vista del esquema *post-fitting* (posterior al ajuste) o *post-mapping* (posterior al mapeo), independientemente de la herramienta de síntesis que se utilice.

2.2.1. ModelSim

La simulación es un paso crítico en el diseño para FPGA. Permite al diseñador estimular su diseño y ver cómo el código que escribió reacciona al estímulo. Una gran simulación contendrá todos los estados posibles del diseño para garantizar que todos los escenarios de entrada se manejen de manera adecuada. Este ejercicio permite descubrir si en alguna parte del diseño, por ejemplo en procesos combinacionales, no se contempló todos casos posibles y en caso de descubrir un comportamiento no esperado se procede a corregirlo.

Para este proyecto se utilizó ModelSim, un entorno de simulación creado por Mentor Graphics. ModelSim Está diseñado para trabajar con Verilog y VHDL. Además cuenta con un debugger, el cual se puede emplear, por ejemplo, para descubrir comportamientos más difíciles de dilucidar que no son determinados a simple vista.

2.2.2. Sentencias VHDL tipo *report* y *assert*

Las sentencias de *report* y *assert* son sentencias de VHDL que se utilizaron para verificar la correcta ejecución de los procesos. La sentencias tipo *report* arroja un

mensaje en la consola de acuerdo a nivel de severidad, cuando el programa pasa por dicha sentencia, se puede imprimir un mensaje en la consola y agregar el valor que tiene una señal o variable determinada. Las sentencias tipo *assert* evalúan el valor de una variable o señal en función de una condición y reportan el resultado en la consola de acuerdo al nivel de severidad.

La sintaxis básica de las declaraciones tipo *report* en VHDL es:

```
1  report <message_string> [severity <severity_level>];
```

ALGORITMO 2.1: "Sintaxis tipo *report*"

Message string es la cadena del mensaje. *Severity level* es el nivel de gravedad del reporte. Los valores posibles de niveles de gravedad son: nota, advertencia, error, falla.

Las declaraciones tipo *report* son declaraciones secuenciales. Esto significa que solo pueden estar en regiones secuenciales, como dentro de las declaraciones que forman parte de un *process*. No pueden estar solos en una arquitectura.

Por otro lado, las declaraciones tipo *assert* pueden ser secuenciales o concurrentes. Es decir, se pueden escribir en un proceso o en una arquitectura. Existen varias formas de escribir una declaración de este tipo, donde la condición es de tipo booleano:

```
1  assert <condition>;
2  assert <condition> severity <severity_level>;
3  assert <condition> report <message_string>;
4  assert <condition> report <message_string> severity <severity_level>;
```

ALGORITMO 2.2: "Sintaxis tipo *assert*"

2.2.3. Buenas prácticas en la codificación VHDL

Guía de estilo de codificación VHDL

Como se mencionó anteriormente, los sistemas desarrollados se diseñaron como subsistemas de un EDDR. Al ser el EDDR un proyecto más grande, involucró el trabajo de tres desarolladores y para unificar criterios de escritura del código VHDL se utilizó una guía de estilo de codificación.

Durante el proceso de revisión de código puede suceder que un problema real sea enmascarado por un problema de estilo de codificación. Dependiendo del proceso, los problemas de estilo pueden tardar mucho en resolverse. Por ejemplo, una serie de pasos a ejecutar podría ser:

- Crear una prueba.
- Encontrar el problema.
- Corregir el problema.
- Verificar solución del problema.

Mantener un estilo de codificación permite por un lado, disminuir la probabilidad de errores y mejorar la lectura para uno mismo y para otros, especialmente durante el desarrollo del código en donde se comparte el trabajo con otros desarrolladores. Por el otro, dedicar menos tiempo a cuestiones de estilo deja más tiempo para analizar la estructura del código.

La eliminación de problemas de estilo reduce la cantidad de tiempo que se tarda en realizar las revisiones de código. Esto da como resultado una base de código de mayor calidad. Por ello, en este trabajó se utilizó un software, desarrollado por Jeremiah C Leary, denominado VSG (del acrónimo VHDL Style Guide).

Las características más destacadas de este programa son:

- Definición explícita de los estándares de codificación VHDL.
- Configurable. Esto permite la actualización del código a los estándares actuales.
- Definición de un estilo del código y aplicación en partes o en toda la base del código.

El programa analiza el archivo cargado evaluando si el código cumple con todas las reglas de estilo, por ejemplo que todos los process tengan una etiqueta o que haya una correcta identación en las diferentes secciones del código. En caso de no hacerlo, VSG informará la regla que se viola y el número de línea o grupo de líneas donde ocurrió la violación. También da una sugerencia sobre cómo corregir la infracción.

La utilización de esta herramienta consistió en analizar cada entidad de diseño con el software y luego, realizar las correcciones pertinentes según el reporte de reglas incumplidas en cada archivo.

Para cuestiones que no están al alcance del software VSG se utilizó una convención para nombrar a puertos y señales

Reglas de codificación

Se adoptó las reglas de codificación propuestas por el departamento de sistemas informáticos de la Universidad Tecnológica de Tampere con el objetivo de aumentar la legibilidad para fines de revisión y evitar formas de codificación dañinas o poco prácticas. Se mencionan las más importantes a continuación:

- Usar solamente puertos tipo IN (de entrada) y tipo OUT (de salida).
- Usar salidas registradas.
- Realizar un *test bench* por cada entidad de diseño.
- Usar el flanco ascendente como flanco activo.
- Hacer que los procesos síncronos sean sensibles sólo al *clock* y *reset*.
- Hacer que los procesos combinacionales o asíncronos sean sensibles a todas las señales leídas en su interior.
- Definir vectores en sentido descendente, tipo *downto*.
- Evitar números mágicos. Es decir, números que sólo el diseñador entiende de donde se originan. Emplear uso de constantes en su lugar.
- Completar con todos los casos posibles las declaraciones secuenciales (por ejemplo *if*, *case*, etc).

Otras convenciones

Adicionalmente a lo mencionado anteriormente en ésta sección, se adoptaron convenciones propuestas por el equipo de desarrolladores del EDDR para nombres. Se mencionan a continuación:

- *i_ nombre* para puertos de entrada.
- *o_ nombre* para puertos de salida.
- *nombre_ s* para señales (cabe recordar que son de uso interno de la entidad).
- *nombre_ v* para variables (función similar a señales).
- *i_ nombre_ numeración* para instancias.
- *nombre_ g* para genéricos.

2.3. Buenas prácticas en el diseño digital

Al diseñar con código HDL, se debe comprender cómo una herramienta de síntesis interpreta las diferentes técnicas de diseño HDL y qué resultados esperar. Las diferentes técnicas de diseño pueden afectar la utilización de la lógica y el desempeño en cuanto al tiempo, así como la confiabilidad del diseño. Las buenas prácticas de diseño sincrónico están hechas para evitar la dependencia de los retrasos de propagación en un dispositivo, lo que puede provocar análisis de tiempos incompletos y posibles fallos. Se adoptó para este proyecto, las prácticas de diseño recomendadas por Altera (empresa fabricante del chip utilizado).

Se utilizó un diseño síncrono. En un diseño de este tipo, una señal de reloj controla la actividad de todas las entradas y salidas. El diseño se comporta de manera predecible y confiable para todas las condiciones de proceso, voltaje y temperatura (PVT) siempre que se asegure de que se cumplen todos los requisitos de temporización de los registros. Además es posible migrar fácilmente diseños síncronos a diferentes familias de dispositivos o grados de velocidad.

Se utilizó en todos los componentes los flancos activos del reloj parar disparar eventos. En cada flanco activo del reloj (que normalmente es el flanco ascendente), las entradas de datos de los registros se muestrean y se transfieren a las salidas. Siguiendo un flanco de reloj activo, las salidas de la lógica combinacional que alimentan las entradas de datos de los registros cambian de valor. Este cambio desencadena un período de inestabilidad debido a los retrasos de propagación a través de la lógica, ya que las señales pasan por varias transiciones y finalmente se establecen en nuevos valores. Los cambios que ocurren en las entradas de datos de los registros no afectan los valores de sus salidas hasta después del siguiente flanco de reloj activo. Debido a que el circuito interno de los registros aísla las salidas de datos de las entradas, la inestabilidad en la lógica combinacional no afecta el funcionamiento del diseño siempre que cumpla con los siguientes requisitos de tiempo:

- Antes de un flanco de reloj activo, la entrada de datos haya sido estable durante al menos el tiempo de configuración del registro (*setup time*).
- Después de un flanco de reloj activo, la entrada de datos permanezca estable durante al menos el tiempo de retención del registro (*hold time*).

Al especificar todas las frecuencias de reloj y otros requisitos de tiempo, se puede utilizar la herramienta *Timing Analyzer* descrita anteriormente para obtener un informe de los requisitos de hardware reales para los tiempos de configuración y tiempos de retención de cada pin del diseño. Al cumplir con estos requisitos de pines externos y seguir las técnicas de diseño sincrónico, se asegura el cumplimiento de los tiempos de configuración y retención para todos los registros del dispositivo utilizado.

Para cumplir con los requisitos de configuración y tiempo de espera en todos los pines de entrada, cualquier entrada a la lógica combinacional que alimenta un registro debe tener una relación sincrónica con el reloj del registro. Si las señales son asíncronas, se puede registrar las señales en las entradas del dispositivo para ayudar a prevenir una violación de los tiempos de retención y de configuración requeridos. Cuando no se cumplen estos requisitos, la salida de un registro puede oscilar la salida o establecer la salida en un nivel de voltaje intermedio entre los niveles alto y bajo llamado estado *metaestable*. En este estado inestable, pequeñas perturbaciones, como el ruido en los rieles de alimentación, pueden hacer que el registro asuma el nivel de voltaje alto o bajo, dando como resultado un estado válido impredecible. Pueden producirse varios efectos indeseables, incluidos mayores retrasos de propagación y estados de salida incorrectos. En algunos casos, la salida puede incluso oscilar entre los dos estados válidos durante un período de tiempo relativamente largo.

Lógica combinacional

Las estructuras lógicas combinacionales constan de funciones lógicas que dependen únicamente del estado actual de las entradas. En las FPGA de Altera, estas funciones se implementan en LUT's con elementos lógicos o módulos lógicos adaptativos (ALM).

Los bucles combinacionales se encuentran entre las causas más comunes de inestabilidad y falta de fiabilidad en los diseños digitales. Los bucles combinacionales generalmente violan los principios de diseño síncrono al establecer un bucle de retroalimentación directa que no contiene registros. En un diseño síncrono, los circuitos de retroalimentación deben incluir registros. Por ejemplo, un bucle combinacional ocurre cuando el lado izquierdo de una expresión aritmética también aparece en el lado derecho en el código HDL. Un bucle combinacional también ocurre cuando retroalimenta la salida de un registro a un pin asincrónico del mismo registro a través de la lógica combinacional. En la sección siguiente se describirá una metodología que previene este tipo de bucles.

Otras recomendaciones tenidas en cuenta en el proceso de diseño fue la de evitar *latches* inintencionales. Un *latch* es un circuito pequeño con retroalimentación combinatoria que mantiene un valor hasta que se asigna un nuevo valor. Es común que los errores en el código HDL provoquen una inferencia de *latch* no intencionada. A diferencia de otras tecnologías, un *latch* en la arquitectura FPGA no es significativamente más pequeño que un registro. La arquitectura no está optimizada para la implementación de *latches* y estos generalmente tienen un rendimiento de temporización más lento en comparación con los circuitos registrados equivalentes.

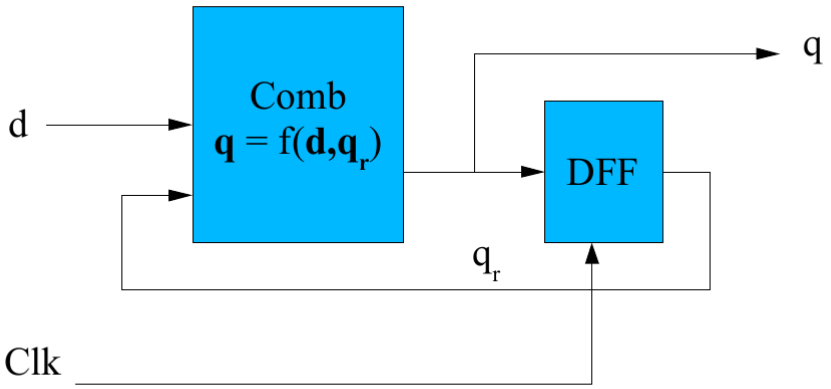


FIGURA 2.4: Diseño sincrónico abstraído en dos partes, combinacional (izquierda) y secuencial (derecha)

2.3.1. Metodología estructurada

Para algunos componentes se empleó la metodología propuesta por Jiri Gaisler de *Gaisler Research*. Se realizó de esa manera para asegurar la síntesis en aquellos componentes donde había más procesamiento del tipo combinacional.

De acuerdo a autor, en el diseño tradicional se presentan a menudo muchas declaraciones concurrentes, muchas señales y procesos de pocas líneas, entre otros. Esto puede ocasionar problemas, por ejemplo, en cuanto a:

- Tiempo de ejecución (dependiendo del número de procesos).
- Dificultad para entender el flujo de datos y/o el algoritmo.
- No distinguir entre señales del tipo secuencial y combinacional y señales relacionadas.

Para que el código sea fácil de entender y mantener, apto para rápida simulación, sintetizable y con menor posibilidad de discrepancias entre simulación y síntesis, el autor propone una abstracción de la lógica digital usando dos partes, como se ilustra en la figura 2.4 , una secuencial y otra combinacional.

Se implementó ésta metodología en VHDL de la siguiente manera:

- La arquitectura albergó dos procesos: uno secuencial y otro combinacional.
- Se declaró dos señales locales, para entrada y salida del registro.
- Todo el algoritmo se volcó en el proceso combinacional.
- La lista de sensibilidad del proceso combinacional se realizó de modo que sea sensible a todos los puertos de entrada.
- La lista de sensibilidad del proceso secuencial se realizó de modo que sea sensible sólo a puerto de reloj.

2.4. Diseño detallado

2.4.1. Partición Software & Hardware

Uno de los primeros pasos para la realización de un diseño digital es la partición entre software y hardware. Es decir, decidir qué parte del diseñó se procesará con software y qué parte con hardware. A continuación se describe el detalle de la partición software/hardware para el CFAR y configurador:

- CFAR
  - Componente de Firmware FPGA:
    - Sistema CFAR de 64 celdas de referencia y 5 celdas de guarda.
    - Posibilidad de alternar entre algoritmos CFAR.
    - Celdas de 14-bit de resolución.
    - Uso de coeficiente multiplicador de 16-bit.
  - Componente de Software HPS:
    - Lectura de cuenta de targets (salida del CFAR).
- Configurador
  - Componente de Firmware FPGA:
    - 32 sectores reducidos que abarquen cada uno una porción de la cobertura.
    - 1 sector fijo que abarque toda la cobertura.
    - Posibilidad de alternar entre sector fijo y sectores reducidos.
    - Control del tipo de video a utilizar (normal o integrado).
  - Componente de Software HPS:
    - Control de coeficiente multiplicador.
    - Control de coeficiente de ventana.
    - Control del algoritmo.
    - Control de tipo de filtrado.
    - Control del modo de operación.
    - Control del submodo de operación.
    - Lectura de cuenta de multiplicador, algoritmo y ventana.

2.4.2. Acciones del CFAR

Se implementó la siguiente jerarquía de entidades de diseño para el CFAR:

- CFAR
  - Celdas de referencia
    - FFD

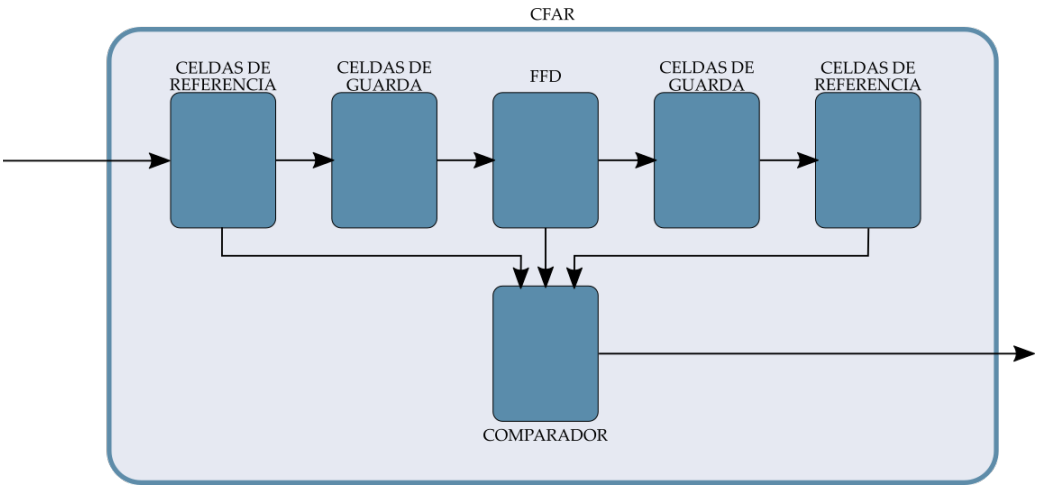


FIGURA 2.5: Relación entre componentes del CFAR

  - Celdas de guarda
    - FFD
  - Celda test
  - Comparador
- Contador CFAR

La relación entre entidades de diseño del CFAR y el flujo de información entre estos se ilustra en la figura 2.5.

Cada componente fue diseñado para realizar las acciones que se describen a continuación.

FFD

- Debe funcionar como un registro de 14-bit de carga y salida paralela.
- Debe transferir cada bit del vector en un flanco ascendente del reloj.
- Debe resetearse de forma asincrónica.
- Debe permitir habilitación o deshabilitación.

Celdas de guarda

- Debe comportarse como un registro de desplazamiento de 5 celdas.
- Debe transferir el dato de video en cada flanco ascendente del reloj.
- Debe resetearse de forma asincrónica.
- Debe permitir habilitación o deshabilitación.

Celdas de referencia

- Debe comportarse como un registro de desplazamiento de 64 celdas.
- Debe transferir el dato de video en cada flanco ascendente del reloj.

- Debe resetearse de forma asincrónica.
- Debe permitir habilitación o deshabilitación.
- Debe proporcionar en cada flanco ascendente de reloj el cálculo de la suma todas del valor de video almacenado en todas las celdas.

Celda Test

Mismo comportamiento que FFD.

Comparador

- Debe recibir el valor de video digital almacenado en la celda bajo testeo.
- Debe recibir la suma de las dos celdas de referencia.
- Debe calcular la suma de las dos celdas de referencia.
- Debe calcular el promedio de cada celda de referencia y el promedio total.
- Debe recibir una señal de configuración para seleccionar uno de tres algoritmos posibles: CA-CGAR, GOCA-CFAR ó SOCA-CFAR.
- Debe recibir un valor de multiplicador correspondiente a un factor de escala.
- Debe generar un umbral con el multiplicador recibido y el promedio correspondiente al algoritmo seleccionado.
- Además del umbral generado con el multiplicador recibido x, debe generar dos umbrales adicionales, uno con el multiplicador decrementado en una unidad $x-1$ y otro con el multiplicador incrementado en una unidad $x+1$.
- Debe comparar el valor almacenado en la celda bajo testeo con cada umbral. Si el valor de la celda bajo testeo lo supera, deberá establecer cada salida correspondiente en un '1' lógico, de lo contrario deberá establecer cada salida correspondiente en un '0' lógico.

CFAR

- Debe recibir las señales de video digital proveniente de dos convertidores analógico a digital (ADC). Corresponde a los videos normal e integrado.
- Debe permitir la selección de la fuente de video.
- Debe contener dos celdas de referencia, dos celdas de guarda y una celda bajo testeo.
- Debe permitir la selección de uno de tres algoritmos posibles: CA-CGAR, GOCA-CFAR ó SOCA-CFAR.
- Debe obtener un umbral a partir de los multiplicadores(x, x+1 y x-1).
- Debe proporcionar a su salida 3 señales de targets correspondiente a los umbrales (x, x+1 y x-1).

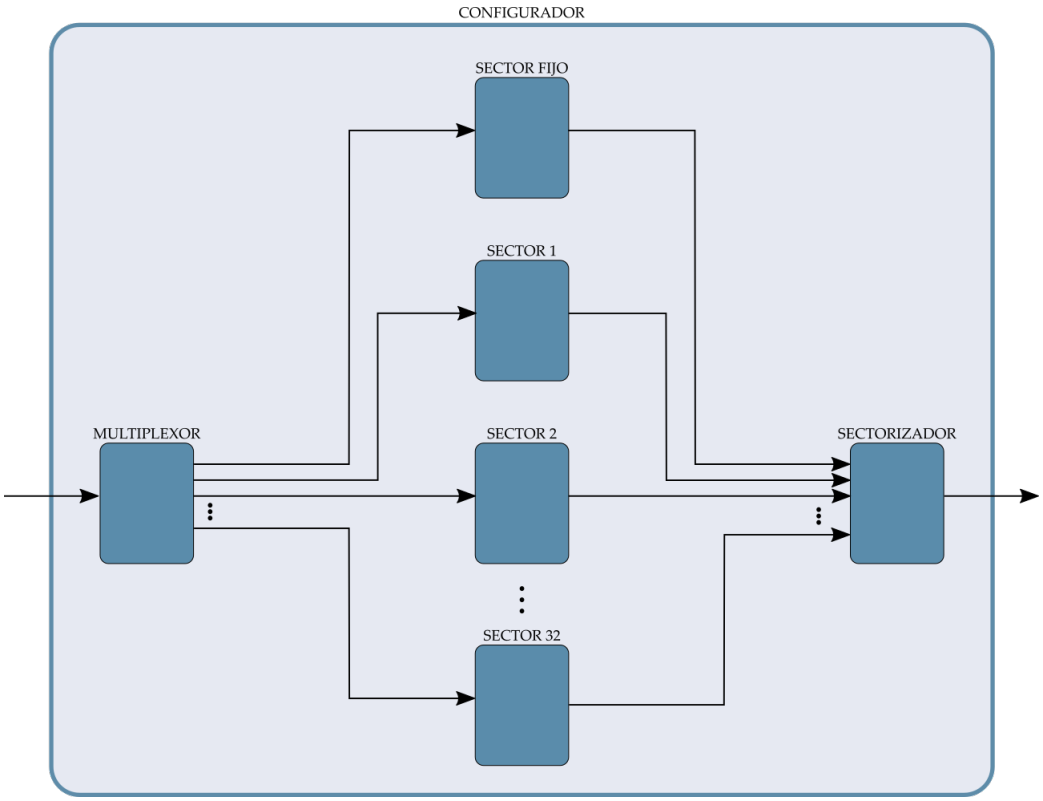


FIGURA 2.6: Relación entre componentes del Configurador

Contador CFAR

- Debe recibir los *targets* (señal de salida del CFAR).
- Debe utilizar la misma señal de reloj que el CFAR.
- Debe incrementar una cuenta cada vez que se detecta un valor '1' lógico.
- Debe proporcionar a la salida la cuenta en forma binaria.

2.4.3. Acciones del configurador

Se implementó la siguiente jerarquía de entidades de diseño para el configurador.

- Configurador
  - Sector fijo
  - Sector
    - Registro de entrada
    - Registro de salida
    - Procesador de presencias
    - Ajuste de multiplicador
  - Sectorizador

La relación entre entidades de diseño del configurador y el flujo de información entre estos se ilustra en la figura 2.6. En esta figura multiplexor de entrada no es una entidad de diseño, sino forma parte de la arquitectura del configurador.

Cada componente fue diseñado para realizar las acciones que se describen a continuación.

Sector fijo

- Debe almacenar la configuración existente en sus puertos de entrada cuando se detecte un flanco ascendente de reloj. La salida debe quedar disponible a partir de ese momento.

Registro de entrada

- Debe almacenar la configuración existente en sus puertos de entrada cuando se detecte un flanco ascendente de la señal de reloj y este coincida con un nivel alto de la señal correspondiente a la habilitación de carga de configuración. La salida debe quedar disponible a partir de ese momento.

Registro de salida

- Debe almacenar la configuración existente en sus puertos de entrada cuando se detecte un flanco ascendente de la señal de reloj y este coincida con un nivel alto de la señal correspondiente al paso por el norte del radar. La salida debe quedar disponible a partir de ese momento.

Sector

- Debe poder operar en dos submodos:
  - Submodo fijo
    - Debe operar únicamente con los registro de entrada y salida
    - La carga de configuración en el registro de entrada estará habilitada sí y solo sí el código de sector que se encuentro dentro de la configuración sea igual al código de identificación de ese sector.
  - Submodo automático
    - Debe contar las presencias
    - En cada paso por el norte debe actualizar el multiplicador almacenado en el registro de salida en función de la cantidad de presencia requerida.
    - En cada paso por el norte debe resetear los contadores.

Procesador de presencias

- Debe poder configurarse con el submodo en submodo fijo o automático.
- En submodo fijo debe quedar inactivo con su salida en la decisión correspondiente a "no incrementar".
- En submodo automático:

- Debe contar las presencias en tres contadores diferentes.
- Debe calcular el error absoluto de cada cuenta con la configuración de la cantidad de presencia requerida por el usuario.
- Debe decidir si para cada tipo de dato (correspondiente a cada contador) es necesario aumentar o disminuir el multiplicador para la siguiente vuelta.

Ajuste de multiplicador

- Debe recibir el valor de decisión proveniente del procesador de presencias.
- Debe contener una máquina de estados con dos estados, fijo y automático.
- En modo automático debe modificar el valor del multiplicador almacenado de acuerdo a la decisión del procesador de presencias; puede aumentar o disminuir el valor del multiplicador o no modificarlo.
- En modo fijo debe quedar inactivo proporcionando a la salida el último multiplicador modificado.

Sectorizador

- El grupo de puertos de entrada correspondientes a un sector determinado debe contemplar al menos los datos de selección de tipo de video, multiplicador, algoritmo y coeficiente de ventana.
- Debe contener tantos grupos de entradas como salidas de configuración de los sectores se utilicen.
- Debe generar el código de sector actual con las cuentas de rango y azimut.
- Su salida debe funcionar como un multiplexor cuya llave de selección debe ser el código del sector actual.

Configurador

- Debe recibir las cuentas de rango y azimut de los subsistemas encargados de realizar esa cuenta.
- Debe contar las presencias utilizando tres contadores diferentes.
- Debe contemplar 2 modos de funcionamiento:
  - Modo fijo (o manual): Debe proporcionar a la salida, un valor fijo de multiplicador, algoritmo, tipo de video y ventana deslizante para toda la cobertura.
  - Modo automático: Debe generar el código de identificación de cada sector predefinido en base a la combinación de los datos de las cuentas de rango y azimut. Debe proporcionar a la salida la configuración almacenada en el sector correspondiente al sector por el que en ese intervalo de tiempo transita el radar.
- Los sectores instanciados deben ser independientes.

- Debe recibir la configuración del puente HPS y redireccionar la misma al sector correspondiente si el modo de operación es automático o al sector fijo si el modo es fijo.
- Su salida debe conectarse al CFAR para configurar su forma operación.
- Debe contar con un multiplexor a la salida que, dependiendo del modo, direccione a los puertos de salida la configuración almacenada en el sector fijo o la que proviene del sectorizador.

Capítulo 3

Ensayos

3.1. Ensayos con marcos de prueba

Se realizó el *testbench* (marco de prueba) para cada entidad de diseño dentro del proyecto, para ensayar la respuesta de cada módulo a señales estímulo. Estas señales fueron representadas de manera de emular las señales correspondientes de aquellas para las que cada módulo se diseñó para manejar. Este tipo de ensayos se utilizó para observar la respuesta a estímulos de las diferentes entidades de la jerarquía de diseño yendo desde la entidad en el nivel más bajo a la entidad en el nivel más alto en la jerarquía.

- Jerarquía del CFAR:
  1. FFD/Celda test
  2. *a*) Celdas de guarda
     b) Celdas de referencia
  3. Comparador
  4. CFAR
  5. Contador CFAR
- Jerarquía del Configurador
  1. *a*) Registro de entrada
     b) Registro de salida
  2. Sector fijo
  3. Procesador de presencias
  4. Ajuste de multiplicador
  5. Sector
  6. Sectorizador
  7. Configurador

Flujo de simulación básico

El diagrama de la figura 3.1 muestra los pasos básicos para simular un diseño en ModelSim.

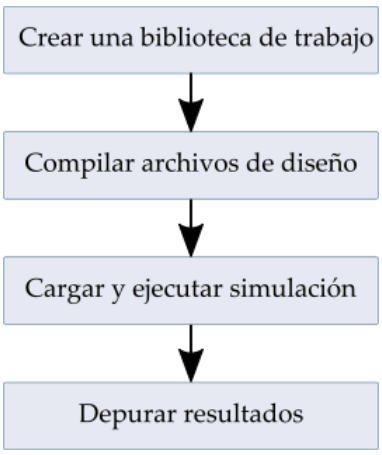


FIGURA 3.1: Flujo de simulación básico

- Creación de la biblioteca de trabajo: En *ModelSim*, todos los diseños se compilan en una biblioteca. Normalmente, una nueva simulación en *ModelSim* comienza creando una biblioteca de trabajo llamada "trabajo", que es el nombre de biblioteca por defecto que utiliza el compilador como destino predeterminado para las unidades de diseño compiladas. Se creó, de este modo, dos bibliotecas, una para el cfar y otra para el configurador, cada una con todas las entidades de la jerarquía y sus correspondiente *testbench*.
- Compilación del diseño: Después de crear la biblioteca de trabajo, se compiló las unidades de diseño en ella. Esta compilación analiza cada archivo de forma individual y reporta, por ejemplo, errores de sintaxis o de falta de entidades en la librería que fueron instanciados en la entidad compilada. Estos reportes permiten realizar las correcciones necesarias en la entidad y/o arquitectura de un determinado archivo fuente. Sin embargo la compilación de todas las entidades no asegura el funcionamiento de todas ellas trabajando en conjunto (como un sistema) debido a que el análisis es a nivel individual.
- Carga del simulador con el diseño y ejecución de la simulación con el diseño compilado: Una vez que la compilación resultó exitosa para todas las entidades de cada jerarquía se cargó el simulador con el *testbench* asociado a cada entidad según el orden propuesto de ensayos descrito en la sección 3.1. Luego se estableció el tiempo de simulación en cero y se ejecutó en cada caso la simulación.
- Depuración sus resultados: Se evaluó los resultados para cada simulación. En algunos casos se escribió reportes dentro de la arquitectura VHDL por ejemplo, para determinar si en un *process* se ingresó correctamente en todos los casos contemplados de alguna sentencia condicional. En otros se visualizó las formas de onda de las señales estímulo (entrada) y de respuesta (salida). En casos más complejos en donde no se obtuvo los resultados esperados y no se pudo determinar de "primera mano"la explicación a un determinado comportamiento, se utilizó el *debugger* incorporado en *ModelSim* para recorrer la ejecución en cada sentencia de las arquitecturas involucradas en el archivo fuente en cuestión.

3.2. Ensayos con visores de netlist

Se utilizó los visores de netlist *RTL Viewer* y *Technology Map Viewer*, descritos en la seccion 2.2 del capítulo 2, para verificar la arquitectura de diseño de las entidades mencionadas en la sección 3.1.

El visor *RTL Viewer* se utilizó para ver los resultados de la síntesis inicial y determinar si se creó la lógica necesaria y si el software interpretó correctamente la lógica y las conexiones. De esta manera se verificó el diseño visualmente.

En los casos en los que la simulación RTL arrojó un comportamiento inesperado, este visor fue útil para rastrear la lista de conexiones y asegurarse que las conexiones y la lógica en el diseño sean las esperadas. En caso de verificar la correcta interpretación de la herramienta de síntesis a este nivel, se analizó las etapas posteriores del proceso de diseño para investigar posibles violaciones de tiempo o problemas en el flujo de verificación en sí.

El visor *Technology Map Viewer* se utilizó para ver los resultados al final de la compilación completa del diseño y determinar qué recursos fueron asignados en aquellos bloques en los que se observó un comportamiento erróneo durante la simulación en *ModelSim*. Además, se utilizó para rastear conexiones entre entidades de diseño, sobre todo para verificar aquellas conexiones dedicadas a transportar la señal de un reloj a varios componentes.

3.3. Ensayos con simulador y decodificador

Para verificar el funcionamiento del Configurador se utilizó un simulador. Dicho equipo se integró dentro de la FPGA con el objetivo de poder tener dos modos de configuración: uno en donde la señal a analizar proveniera del radar y otro en el que la señal proveniera del simulador. En el segundo modo de configuración, el equipo simuló todas las señales correspondientes a los radares primario y secundario, incluidos los ecos de estos sensores. Por defecto, en este modo se simulan 8 plots primarios y 8 plots secundarios por cada giro de antena, como se ilustra en la figura 3.2. Este modo permite efectuar ensayos de mantenimiento y calibración sobre las distintas partes del Procesador Monoradar cuando no se dispone físicamente del sistema de radar.

Para ensayar el funcionamiento del configurador se diseñó un decodificador para recibir los datos de salida del mismo. Se dividió la cobertura en dos partes (agrupando 16 sectores en cada parte) y cada una se configuró con valores diferentes de algorítmo, multiplicador y coeficiente de ventana deslizante para verificar visualmente la confirmación (o no) de estos valores mediante el encendido (o apagado) de unos leds integrados en la placa.

Para determinar el crecimiento de la cantidad de targets detectados por el CFAR en un giro de antena, se utilizó el contador descrito en la sección 2.4.2. Los cuatro bits más significativos de la cuenta se conectaron a cuatro leds integrados en la placa para verificar visualmente su variación a medida que el radar barrió la cobertura.

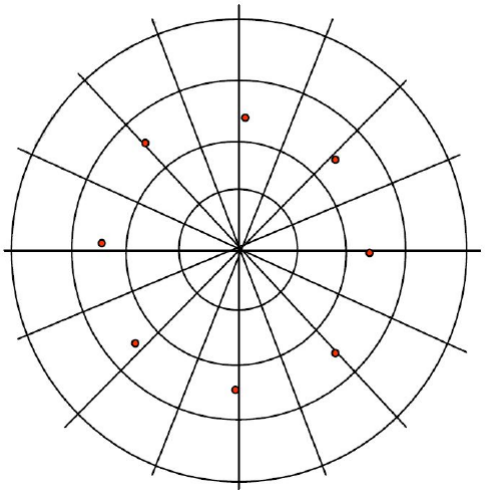


FIGURA 3.2: Distribución de plots de prueba

3.4. Ensayos con Procesador Monoradar preexistente

Se utilizó un Procesador Monoradar preexistente para contrastar los datos arrojados por el Procesador Monoradar del que tanto el CFAR como el configurador descritos en este trabajo fueron parte. Para el CFAR se contrastó las ubicaciones de los plots detectados por ambos sistemas y se observó además la distribución de clutter. En el caso del configurador, se configuró varios sectores con diferentes valores de multiplicador para observar variación de clutter debido a los diferentes umbrales generados en estos sectores de la cobertura. Además para el configurador se varió el nivel de multiplicador para algunos sectores, desde el nivel mínimo al máximo con el objetivo de visualizar en pantalla los límites de estos sectores.

Capítulo 4

Resultados

4.1. Resultados

4.1.1. Resultados de ensayos con marcos de prueba

FFD. Ensayo 1

Se introdujo los siguientes estímulos:

- Señal de reloj de 1 ns de periodo.
- Señal de video de entrada `d_ffd_tb` de `14bit` cuyo valor crece cada 1 ns.
- Señal de habilitación `en_ffd_tb` con valor permante en '1' lógico (siempre habilitado).
- Señal de reset `rst_ffd_tb` con valor permanente en '0' lógico (nunca resetea).

Los resultados se ilustran en la parte derecha de la figura 4.1. Se observó en la señal de salida `q_ffd_tb` que:

- En los primeros 5 ns la salida está indefinida. Lo cual es esperable.
- En cada flanco ascendente del reloj el valor de la señal de entrada se transfiere al puerto de salida.

Los comportamientos son esperados.

FFD. Ensayo 2

Se introdujo los siguientes estímulos:

- Señal de reloj de 1 ns de periodo.
- Señal de video de entrada `d_ffd_tb` de 14-bit cuyo valor crece cada 1 ns.
- Señal de habilitación `en_ffd_tb` con valor '1' lógico durante los primeros 1.25 ns, '0' lógico entre 1.25 y 2.25 ns, y '1' lógico luego de 2.25 ns.
- Señal de reset `rst_ffd_tb` con valor '0' lógico hasta 3.25 ns y '1' lógico luego de 3.25 ns.

Los resultados se ilustran en la parte izquierda de la figura 4.1. Se observó en la señal de salida `q_ffd_tb` que:

- En los primeros 5 ns la salida está indefinida. Lo cual es esperable.

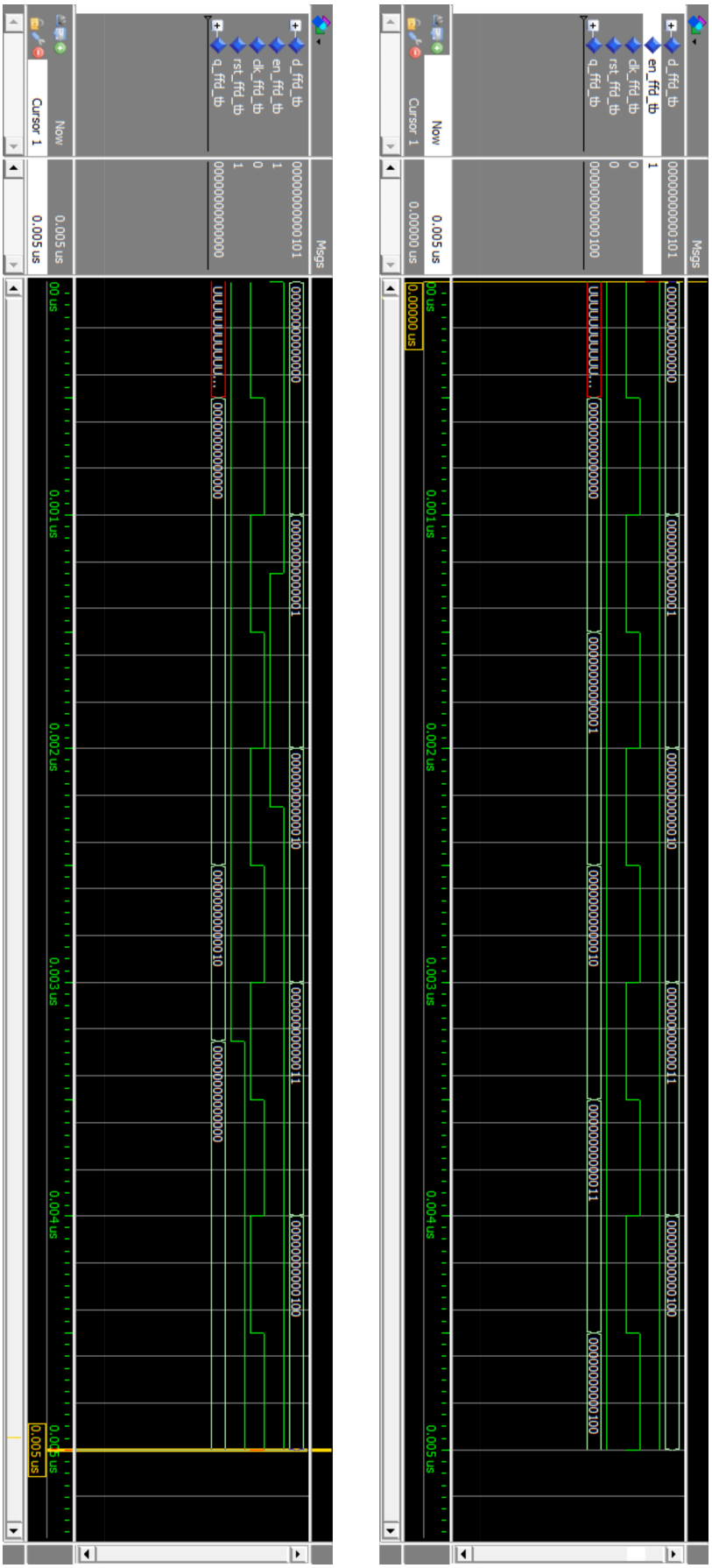


FIGURA 4.1: Ensayo 1 (derecha) y 2 (izquierda) sobre el módulo "ffd"

- En primer flanco ascendente del reloj (5 ns) el valor de la señal de entrada se transfiere al puerto de salida.
- Entre 5 ns y 2.25 ns la salida se mantiene en el valor anterior ("00000000000000"). Es decir, en el segundo flanco ascendente del reloj el valor de la señal de entrada **no** se transfiere a la salida. En este mismo intervalo se observa un nivel bajo de la señal de habilitación, lo que significa que *no* está habilitado la tranferencia del dato de la entrada a la salida y un nivel bajo de la señal de reset (reset inactivo).
- En el tercer flanco ascendente del reloj (2.5 ns) ocurre la transferencia del dato de la entrada a la salida, correspondiente al nivel alto de la señal de habilitación y al nivel bajo de la señal de reset.
- Desde 3.5 ns hasta el final de la simulación, lo que incluye el cuarto y quinto flanco ascendente del reloj, se observa que el dato de salida es "00000000000000". Esto se corresponde con el cambio de nivel bajo a nivel alto de la señal de reset. Esto provoca que la salida se resetee.

Los comportamientos son esperados.

Celdas de guarda. Ensayo 1

Se introdujo los siguientes estímulos:

- Señal de reloj de 1 ns de periodo.
- Señal de video de entrada `d_ffd_tb` de 14-bit cuyo valor crece cada 1 ns.
- Señal de habilitación `en_ffd_tb` con valor permante en '1' lógico (siempre habilitado).
- Señal de reset `rst_ffd_tb` con valor permanente en '0' lógico (nunca resetea).

Los resultados se ilustran en la parte izquierda de la figura 4.2. Se observó en la señal de salida `q_ffd_tb|` que:

- En los primeros 5 ns la salida está indefinida. Lo cual es esperable.
- En cada flanco ascendente del reloj el valor de la señal de entrada se transfiere al puerto de salida.

Los comportamientos son esperados.

Celdas de guarda. Ensayo 2

Se introdujo los siguientes estímulos:

- Señal de reloj de 1 ns de periodo.
- Señal de video de entrada `d_ffd_tb` de 14-bit cuyo valor crece cada 1 ns.
- Señal de habilitación `en_ffd_tb` con valor '1' lógico durante los primeros 1.25 ns, '0' lógico entre 1.25 y 2.25 ns, y '1' lógico luego de 2.25 ns.
- Señal de reset `rst_ffd_tb` con valor '0' lógico hasta 3.25 ns y '1' lógico luego de 3.25 ns.

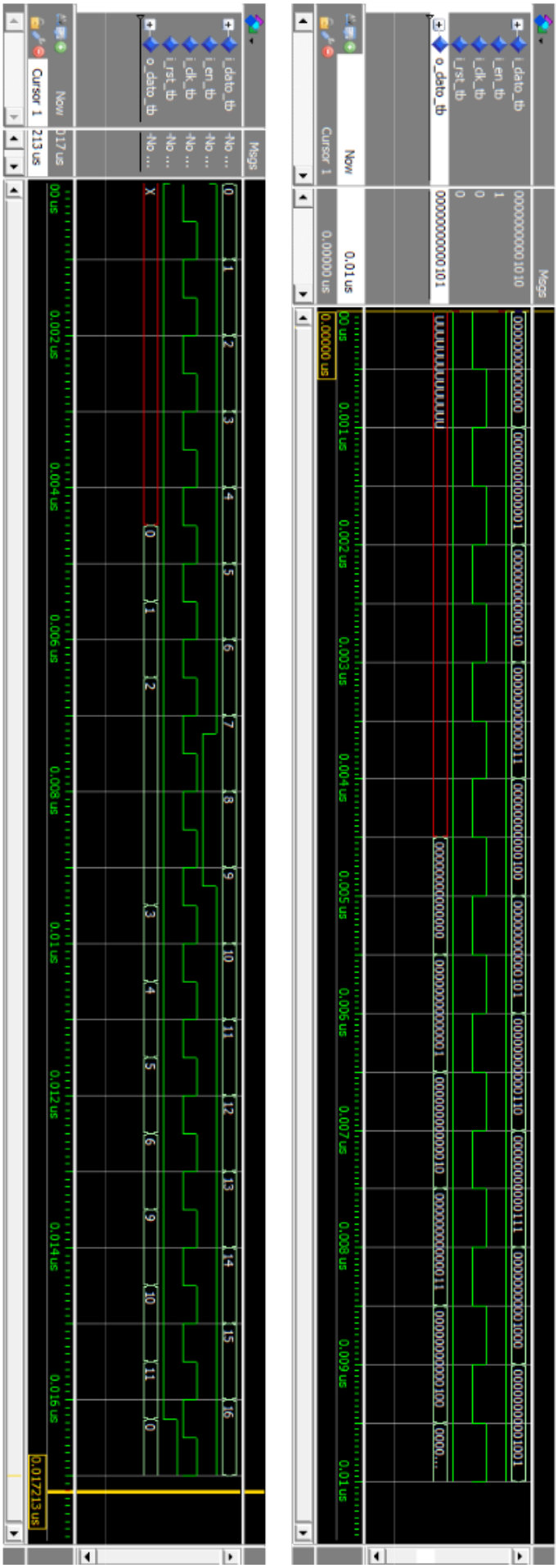


FIGURA 4.2: Ensayo 1 (izquierda) y 2 (derecha) sobre el módulo 'celdas de guarda'

Los resultados se ilustran en la parte derecha de la figura 4.2. Se observó en la señal de salida `q_ffd_tb` que:

- En los primeros 5 ns la salida está indefinida. Lo cual es esperable.
- En primer flanco ascendente del reloj (5 ns) el valor de la señal de entrada se transfiere al puerto de salida.
- Entre 5 ns y 2.25 ns la salida se mantiene en el valor anterior ("00000000000000"). Es decir, en el segundo flanco ascendente del reloj el valor de la señal de entrada **no** se transfiere a la salida. En este mismo intervalo se observa un nivel bajo de la señal de habilitación, lo que significa que *no* está habilitado la tranferencia del dato de la entrada a la salida y un nivel bajo de la señal de reset (reset inactivo).
- En el tercer flanco ascendente del reloj (2.5 ns) ocurre la transferencia del dato de la entrada a la salida, correspondiente al nivel alto de la señal de habilitación y al nivel bajo de la señal de reset.
- Desde 3.5 ns hasta el final de la simulación, lo que incluye el cuarto y quinto flanco ascendente del reloj, se observa que el dato de salida es "00000000000000". Esto se corresponde con el cambio de nivel bajo a nivel alto de la señal de reset. Esto provoca que la salida se resetee.

Los comportamientos son esperados.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.