UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Programa de Maestría y Doctorado en Ingeniería

Ingeniería Eléctrica – Sistemas Electrónicos

«DISEÑO DE UN SISTEMA DE CONTROL DIGITAL ESCALABLE Y RECONFIGURABLE PARA ANTENAS EN ARREGLOS DE FASE»

T E S I S

QUE PARA OPTAR POR EL GRADO DE

MAESTRO EN INGENIERÍA

P R E S E N T A

Ernesto Hernández Gastaldi

TUTORES

Dr. Oleksandr Martynyuk | Facultad de Ingeniería

Dr. Saúl de la Rosa Nieves | Facultad de Ingeniería

Ciudad de México. Marzo 2023.

AGRADECIMIENTOS

A mis papás, **Ernesto** y **Griselda**:

Una vez más, por apoyarme constante e incondicionalmente con las decisiones que he tomado a lo largo de mi formación académica, desde permitirme elegir la carrera que deseaba, aun con todas las implicaciones que conllevó durante esos 5 largos años, hasta este posgrado que no hubiera ni siquiera contemplado sin su apoyo y afecto. Los logros de toda mi vida siempre son y serán en nombre de ustedes.

A «boo», **Ximena A.**:

Por ser mi principal soporte emocional desde el inicio de este posgrado. Gracias por estar siempre ahí para mí en mis frustraciones, problemas emocionales y siempre inspirarme a seguir adelante, por toda la motivación y el incontable cariño que he recibido de ti.

A mi «main», **Mauricio Byrd**:

Por ser el amigo que, desde principios de la carrera, se convirtió en mi hermano y en un gran ejemplo a seguir. Gracias a ti hoy consigo ser un mejor profesionista, y sinceramente tú has sido mi fuente de inspiración para comenzar y terminar este grado de estudios.

A un profesor excepcional, **José Luis Barbosa**:

Porque gracias a él desarrollé un profundo interés por la electrónica digital, y heme aquí, unos años después presentando una tesis de maestría completamente de tal área. Más allá de eso, le agradezco crear ese gran interés a la investigación y al pensamiento crítico, que surgió y se hizo mayor en cada clase de cada curso que tuve con él.

Agradezco de forma sincera a los asesores del trabajo, los doctores **Saúl** y **Oleksandr**, por toda su guianza en la elaboración de este proyecto, por el tiempo dedicado, por sus conocimientos, consejos, regaños y por su apoyo en general a lo largo de todo este plan de estudios. De igual manera, doy las gracias a los proyectos DGAPA-PAPIIT **IN119420** y **IN114823**, que permitieron adquirir las tarjetas de desarrollo para las pruebas físicas del proyecto elaborado en este trabajo.

Finalmente, permito expresar mi agradecimiento al **Consejo Nacional de Ciencia y Tecnología** del Gobierno de México, pues sin el sustento y apoyo recibidos por la beca obtenida no hubiera podido completar exitosamente los estudios de maestría.

ÍNDICE

RESUMEN

En el siguiente trabajo de tesis se relata el diseño de un sistema de control digital para antenas tipo arreglos de fase. El sistema implementado cuenta con una arquitectura escalable y reconfigurable, de manera que puede servir para controlar antenas de diversos parámetros y geometrías.

La topología del diseño es de tipo «maestro-esclavos» y la implementación fue llevada a cabo en el lenguaje de descripción de hardware SystemVerilog. Los módulos/componentes que conforman el sistema fueron descritos para ser fácilmente añadidos o removidos, consiguiendo así la escalabilidad buscada. De igual manera, se cuenta con una interfaz de usuario que permite cargar los parámetros de la antena a controlar, logrando una fácil reconfigurabilidad.

Para la experimentación, el sistema fue implementado en dispositivos FPGA; las pruebas fueron analizadas a través de un subsistema externo encargado de la obtención de resultados. Los experimentos llevados a cabo mostraron un tiempo de respuesta promedio de 1.1 μs y un error de orientación máximo de 0.2°.

ESTRUCTURA

En el capítulo 1 se presenta una introducción al planteamiento del problema, así como una investigación sobre el estado del arte y se definen los objetivos del trabajo. En el segundo capítulo se encuentran explicados los conceptos requeridos sobre antenas en arreglos de fase, así como conceptos sobre electrónica digital que son frecuentados a lo largo del trabajo. Dentro del capítulo 3 se relata a detalle el diseño del sistema, desde las propuestas preliminares, la arquitectura general y hasta cada uno de los módulos que lo comprenden. Para el capítulo 4 se encuentra el planteamiento de los experimentos que fueron propuestos, así como el módulo prototipo para las pruebas; es en el capítulo 5 donde se estudian a detalle los resultados que fueron obtenidos. Se finaliza el trabajo escrito con el capítulo 6, dedicado a las conclusiones y el trabajo a futuro.

CAPÍTULO 1
INTRODUCCIÓN

Las comunicaciones inalámbricas han sido un sueño hecho realidad desde el año de 1901, cuando Guglielmo Marconi recibió exitosamente la primera transmisión transatlántica de forma inalámbrica [1]; desde aquel entonces, esta tecnología ha evolucionado a grandes pasos por la inmensa cantidad de aplicaciones que posee. Las antenas son los objetos de mayor relevancia para poder recibir y transmitir de esta manera. El ejemplo más común de este tipo de antenas son los reflectores, y de este lo son las antenas parabólicas, que son capaces de recibir y transmitir ondas electromagnéticas con considerable precisión gracias a su geometría y a que algunas de ellas poseen adicionalmente rotación física.

Por otro lado, desde la década de los 60 existe un concepto de un tipo de tecnología que gradualmente ha ganado popularidad e interés entre la comunidad científica de esta área, dicha tecnología consiste en antenas en arreglos reflectivos. Son una tecnología de diseño híbrido que imita las características de los reflectores como una alternativa electrónicamente reconfigurable.

Actualmente, las antenas tipo arreglo reflectivo han adquirido aplicaciones relacionadas con exploración espacial, comunicación satelital, sensores remotos, radares de escaneo, etc. Adicionalmente, se prevé que con los avances en las tecnologías de fabricación de Printed Circuit Boards (PCB) y en nanotecnología, se puedan implementar arreglos reflectivos ópticos a un costo accesible y, por ende, esto lleve a un incremento y popularización aún mayor en cuanto a su nivel de utilización [2].

Los arreglos reflectivos reconfigurables son, de manera general, antenas cuya función es enviar o recibir una onda incidente en ellos hacia o desde un ángulo deseado, con la característica particular de que no es necesario el movimiento físico de la antena, la orientación se lleva a cabo mediante el control de las diferencias de fase generadas por sus elementos conformados por desplazadores de fase. Los elementos que conforman los arreglos comúnmente contienen diodos tipo Positivo-Intrínseco-Negativo (PIN), por lo que el control de los desplazadores se realiza directamente mediante la polarización de dichos diodos, y esta a su vez se lleva a cabo mediante la alimentación de sus terminales.

Respecto a los elementos de los arreglos, los ángulos que son capaces de desfasar son limitados, es decir, son valores discretos y están sujetos a su estructura. La cantidad de diodos con los que cuenten incrementa el número de fases, sin embargo, también incrementa su complejidad. El arreglo reflectivo más básico consiste de un alimentador, un controlador, y un arreglo de un único elemento conformado por un solo diodo, el cual sería capaz de otorgar un desplazamiento de 0 ° y 180 ° mediante el sentido de su polarización (encendido y apagado) [3]. Las placas que forman las antenas reflectivas pueden tener distintas formas geométricas: cuadradas, rectangulares y circulares. En la Figura 1-1 se ilustra un arreglo de forma cuadrada con su alimentador y su respectivo controlador.

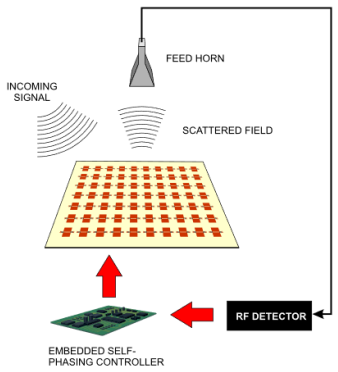


Figura 1-1. Estructura de un arreglo reflectivo electrónicamente reconfigurable con diodos PIN. Fuente: [4].

La manera en que se realizan las polarizaciones en los diodos proviene de un cálculo matemático que da como resultado un ángulo de desplazamiento, es decir, se debe realizar un cálculo para cada elemento del arreglo. Este ángulo de desplazamiento tiene un valor real ubicado entre 0° y 360°, y como se sabe que los elementos solo pueden desfasar en valores discretos debido a su número finito de diodos, para cada caso se debe realizar un redondeo del valor teórico al valor

más cercano que se puede tener de forma práctica. Posteriormente, existe una tabla de codificación que relaciona las polarizaciones de los diodos con diferentes ángulos de desfase, por lo que las señales de control que van a los diodos son como las salidas de esta etapa de codificación [5]. En la Figura 1-2 pueden observarse las diferencias de fase generadas por los elementos desplazadores de fase.

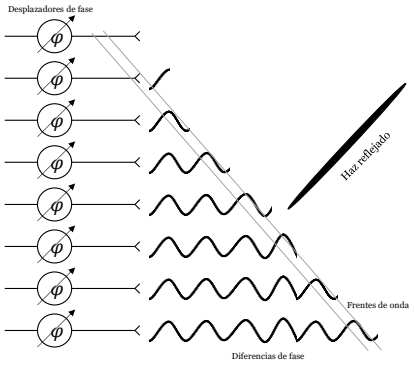


Figura 1-2. Ilustración del desfasamiento en conjunto mediante desplazadores de fase.

La estructura interna de un diodo PIN está presente en la Figura 1-3, los cuales se encuentran formados por una difusión tipo P y tipo N, con una capa de silicio puro en medio de ellos denominada región I (intrínseca). Estos diodos se caracterizan por tener una resistencia interna que varía inversamente a la polarización aplicada. A polarización nula, prácticamente no hay cargas libres en la región I para que el diodo conduzca, lo que hace que se comporte como una especie de capacitor cuyo valor ronda los femtofarads [6]; sin embargo, para asegurarse que las cargas libres sean las menores posibles, normalmente se polarizan inversamente en lugar de dejarse sin polarización. Si se les polariza de forma directa, la región I pasa a almacenar cierta cantidad de carga, lo cual conlleva al estado de conducción y a que los PIN se comporten como una resistencia cuyo valor disminuirá entre mayor sea el voltaje de polarización [7].

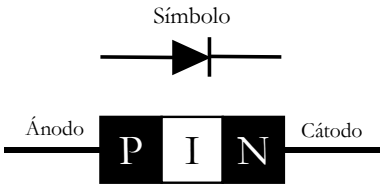


Figura 1-3. Estructura interna de un diodo PIN.

En lo que respecta al controlador de las antenas reflectivas, su mayor dificultad radica en el gran número de señales de control y en la velocidad de respuesta. Recordando que el número de señales de control es igual al número de diodos por elemento multiplicado por el número de elementos, es una cantidad que puede ir desde algunos cientos hasta las decenas de miles; sumado a ello, los controladores deben ser capaces de proporcionar todas las señales de control en un tiempo de respuesta deseado que esté el orden de los pocos microsegundos, para que así el haz reflejado por la antena se pueda desplazar casi de manera continua.

El proyecto a desarrollar en este trabajo surge, de forma general, a partir del requerimiento de controlar las antenas de arreglos reflectivos de manera eficiente. Como una manera de integrar una cooperación entre el módulo de radiofrecuencia y microondas y el de instrumentación electrónica, se ha optado por ver a este problema como una aplicación concreta de instrumentación, y específicamente como una aplicación del diseño de sistemas digitales.

Asimismo, la evidente falta de información en cuanto a tecnologías que se han implementado para el mismo propósito hace que el diseño de este sistema sea un reto aún más interesante, pues el presente trabajo de tesis será una de las contadas publicaciones que documenten de forma detallada el diseño de un sistema de control para dicho propósito. Mientras que, con respecto al diseño digital, se le ve como una aplicación específica que integra los retos de escalabilidad, reconfigurabilidad, almacenamiento de datos y procesamiento en paralelo.

1.1 OBJETIVOS

Objetivo general:

Consiste en el diseño e implementación de un sistema de control digital para la reconfiguración electrónica de antenas tipo arreglos reflectivos, que posea las capacidades de escalabilidad y reconfigurabilidad, además de que cumpla con los requerimientos establecidos de velocidad de respuesta, con el fin mejorar el funcionamiento de los sistemas de telecomunicaciones satelitales de última generación.

Objetivos específicos:

- Planteamiento de un modelo de arquitectura que funcione de forma escalable y reconfigurable.
- Diseño de un módulo prototipo que será la referencia de la arquitectura propuesta.
- Planteamiento de las etapas de funcionamiento de la arquitectura propuesta.
- Diseño del sistema encargado de calcular las fases para los elementos del arreglo.
- Diseño del sistema encargado de convertir las fases en señales de control para los desplazadores de fase.
- Sincronización y comunicación entre los dispositivos que conforman el sistema.
- Obtención de las simulaciones que validen el funcionamiento del modelo implementado.
- Implementación de un prototipo para pruebas físicas.

1.2 ESTADO DEL ARTE

El control de los elementos que contienen los arreglos de fase es, sin duda alguna, una parte indispensable de su funcionamiento, pues si no se realiza la correcta polarización de los desplazadores de fase, no hay forma de que se obtengan las ventajas que este tipo de antenas presentan con respecto a las antenas parabólicas convencionales. Los sistemas de control para tal propósito necesitan satisfacer los requerimientos establecidos de velocidad de respuesta, y del total de líneas de control necesario según la aplicación a la que se enfoque el arreglo. A continuación se mencionan descripciones breves sobre los controladores que han sido desarrollados y reportados en distintas referencias encontradas durante la investigación documental.

Xiaotian Pan et al. presentan la creación de un arreglo de geometría cuadrada para banda X, que cuenta con 160x64 elementos de un diodo PIN por elemento [8]. Con respecto al sistema de control, las 10,240 líneas totales son agrupadas en 40 subsistemas, y cada subsistema está controlado por cuatro Field-Programmable Gate Arrays (FPGA), donde cada una se encarga de enviar las señales de forma paralela a 64 líneas. Los autores mencionan que se utilizaron 160 tarjetas para cubrir el total y que, aunque parecen una cantidad muy grande, sirven para paralelizar al máximo la polarización de los elementos y asimismo minimizar la velocidad de respuesta. Aunque no se menciona a detalle el funcionamiento de los dispositivos, se comenta que las FPGA ya tienen precomputadas y almacenadas las polarizaciones de los diodos para los distintos ángulos de orientación. Sus resultados reportaron un tiempo de conmutación de 2 μs. De forma similar, en [9] se presenta un arreglo más pequeño, de 14x14 elementos, los cuales son controlados de igual forma que el artículo anterior, solo que en este caso únicamente se requirieron 4 FPGA Cyclone IV trabajando en conjunto de forma paralela.

Por otro lado, el trabajo de Huanhuan Yang et al. relata la creación de un arreglo compuesto de 1600 elementos de un solo diodo [10]. A través de una FPGA y de una computadora se realiza el control de los desplazadores: el operador envía los comandos de orientación a través de la computadora hacia la FPGA, esta los recibe y posteriormente envía las señales de control hacia registros de corrimiento que realizan la distribución de las señales de forma paralela. Los resultados obtenidos por los autores muestran un tiempo de conmutación que ronda los 16 μs. Una implementación bastante parecida es realizada en [11], donde se utiliza un microcontrolador para generar la señal de control codificada en 10 bits, que permite hasta 1024 patrones de radiación; la señal de 10 bits es enviada hacia una FPGA que contiene las polarizaciones de los 196 diodos del arreglo para los 1024 patrones de radiación, en este caso la FPGA solamente funciona como lo que ellos denominan un 'distribuidor de voltajes'.

Asimismo, en [12] se menciona el diseño de un pequeño arreglo de 100 elementos de un diodo para la banda Ku. El controlador que se diseña junto con el arreglo consiste en un microcontrolador y en registros de corrimiento comerciales de 8 bits. Se utilizan 20 registros que son distribuidos para controlar 5 diodos PIN cada uno, y las 20 señales seriales son

generadas de forma paralela a través del microcontrolador y enviadas de tal forma. Los experimentos de esta implementación mostraron un tiempo de conmutación de 12.5 μs.

Un diseño similar al anterior se reporta en el artículo [13], donde junto con el diseño de un arreglo de 10x24 elementos de un transistor de efecto de campo, se implementa un controlador en una computadora personal a través de una interfaz gráfica programada en C++. La computadora envía los comandos de orientación a través de Universal Serial Bus (USB) hacia un puente Serial Peripheral Interface (SPI) que funciona como una especie de expansor, el cual una vez recibidos los comandos los distribuye hacia columnas agrupadas en 16 elementos. La velocidad de respuesta con la implementación inicial rondó únicamente los 50 bps, así que la generación de señales de control después se programó en un microcontrolador Arduino, y esto incrementó la velocidad a 15 kbps, dando un tiempo aproximado de respuesta de 66.66 μs.

Como es posible observar de los trabajos encontrados en la respectiva investigación, las implementaciones de los sistemas de control se centran en dispositivos de lógica digital, ya sea a través de microcontroladores o a través de dispositivos lógicos reconfigurables, siendo estos últimos los más utilizados. También es de notarse que la mayoría de trabajos siguen una distribución progresiva de las señales de control hacia los elementos, es decir, se utiliza tanto procesamiento paralelo como distribución serial, esto mayormente en casos en los que la demanda de señales de control vuelve muy complicada la paralelización completa de la generación de señales. Por ejemplo, puede observarse cómo en el artículo [8] para lograr el control de todos los elementos de forma completamente paralela se necesitó una gran cantidad de dispositivos lógicos (160) trabajando en conjunto. En la Tabla 1-1 se encuentran resumidas las características de los sistemas de control previamente mencionados.

Tabla 1-1. Comparativas de los sistemas de control del estado del arte.

Autor	Características del arreglo	Señales de control	Características del sistema	Velocidad de respuesta
Xiaotian Pan et al. [8]	164×64 elementos de 1 bit para banda X	10,240 diodos PIN	Conformado por 160 FPGA	2 μs
Zhenglong Wang et al. [9]	14×14 elementos de 1 bit para banda Ku	196 diodos PIN	Conformado por 4 FPGA	No mencionado
Huanhuan Yang et al. [10]	40×40 elementos de 1 bit para banda X/Ku	1600 diodos PIN	Conformado por una computadora y una FPGA	16 μs
Hai Zhang et al. [11]	14×14 elementos de 1 bit para banda X	196 diodos PIN	Conformado por un microcontrolador y una FPGA	No mencionado
Huanhuan Yang et al. [12]	10×10 elementos de 1 bit para banda Ku	100 diodos PIN	Conformado por un microcontrolador y 20 registros de corrimiento	12.5 μs
R. L. Schmid et al. [13]	10×24 elementos de 1 bit para banda Ku	240 diodos PIN	Conformado por una computadora, un microcontrolador Arduino y un puente SPI	66.66 μs

El resto de sistemas encontrados en la investigación siguen una arquitectura más o menos similar, donde las señales de control son generadas inicialmente y después se distribuyen de distintas formas hacia todos los elementos que conforman el arreglo, y esto da un indicio de qué es lo que se debe hacer para el diseño del controlador que se realizará en el presente trabajo. También se considera importante mencionar que los sistemas de control que se hallaron forman parte del diseño de una antena y por ende son hechos a la medida, además de que en la mayoría de casos no se da una gran explicación de cómo es que funcionan o cómo fueron implementados, pues los documentos se centran principalmente en el diseño del arreglo.

De igual forma, no fue posible encontrar algún documento que relatara el diseño únicamente del controlador, ni para una aplicación en específico, ni para propósito general como el que se planea implementar en el presente proyecto. A partir de los casos encontrados en el estado del arte, se considera que el diseño de un sistema de control como modelo escalable y reconfigurable es una propuesta que puede contribuir de manera importante al desarrollo y aplicaciones para este tipo de antenas.

CAPÍTULO 2
MARCO TEÓRICO

El motivo del presente capítulo es proveer de un contexto teórico sobre los dos temas principales que abarca la tesis: las antenas y los sistemas digitales. Así bien, dentro del presente se encuentran dos subcapítulos, cada uno con sus secciones está enfocado a uno de los dos respectivos temas.

El marco teórico no tiene como propósito indagar con una gran profundidad sobre los temas antes mencionados. Más allá de los conceptos que así lo requieran, este capítulo sirve únicamente como guía general para comprender los términos que se mencionan de manera frecuente a lo largo del trabajo escrito.

2.1 ANTENAS

2.1.1 Descripción general

Las ***antenas*** son, a grandes rasgos, elementos sensibles cuya función es capturar cierto estímulo, como las antenas que poseen los insectos, sensibles a las vibraciones y que les permiten estar al tanto de su entorno. Por el lado de la electrónica, las antenas son dispositivos cuya función es recibir o transmitir ondas electromagnéticas. Normalmente se trata de dispositivos resonantes, lo que implica que son sensibles a sólo cierto rango de frecuencias [14]; esta característica es conocida como ***ancho de banda***.

Otra de las tantas características de las antenas es el ***patrón de radiación***, que se puede ver como las regiones del espacio en las que la antena es capaz de transmitir o recibir. Ya que ninguna puede tener una sensibilidad completamente omnidireccional, el patrón de radiación sirve para caracterizar la forma e intensidad en que las ondas electromagnéticas son transmitidas o recibidas por la antena en cuestión [15]. Por ejemplo, en la Figura 2-1 se encuentra el patrón de radiación de una antena tipo dipolo, que es una de las que más se acercan a tener un patrón omnidireccional.

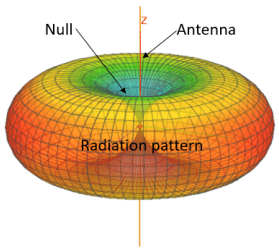


Figura 2-1. Patrón de radiación de una antena dipolo. Fuente: [16].

A partir del patrón de radiación es posible llegar a otro concepto también importante: este es conocido como la ***directividad***. Esta característica básicamente describe qué tanto del patrón de radiación se propaga hacia una dirección en específico [17], es decir, entre más concentrado esté hacia una dirección, más «directiva» será la antena. Obsérvese el ejemplo la Figura 2-2, que contiene el patrón de radiación de una antena parabólica y comparar con la Figura 2-1. Es fácil notar que la antena parabólica es mucho más directiva que la antena dipolo, pues la mayoría de su energía se está concentrando hacia un lado en específico (esto también es nombrado *lóbulo principal*).

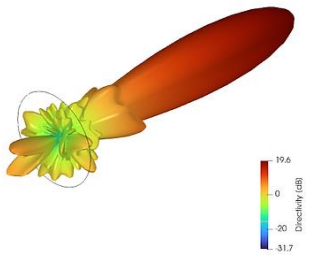


Figura 2-2. Patrón de radiación de un reflector parabólico. Fuente: [18].

2.1.2 Usos y aplicaciones

Existe una gran variedad de antenas distintas, por solo mencionar algunas se comentan las siguientes: dipolo, yagi, reflectores parabólicos, logperiódicas, de microcinta, en arreglos de fase, etc. [15]. Cada tipo antes mencionado tiene sus características propias y por ende sus respectivas aplicaciones. Para el caso de estudio en este trabajo, resulta importante mencionar cuáles son las aplicaciones de las antenas poco directivas y de las que tienen una alta directividad.

Para el caso de las antenas que mayormente son omnidireccionales, sus aplicaciones recaen principalmente en el envío de información hacia áreas amplias, de manera que no sea solo un objetivo al que se desea transmitir, sino más bien un área en la que puede haber varios receptores. Ejemplos de aplicaciones para este tipo de antenas son: transmisión de estaciones de radio, de estaciones de televisión, módems para internet inalámbrico [19], entre otros.

La prioridad de las antenas poco directivas es poseer cobertura en la mayor cantidad de espacio posible, con el objetivo de que cualquier receptor que esté dentro del área pueda recibir un poco de la energía emitida. En otras palabras, la energía distribuida por una antena con un patrón de radiación (casi) omnidireccional se distribuye (casi) equitativamente en todas direcciones, y la cantidad de energía recibida idealmente solo depende de la distancia entre el receptor y el transmisor, sin importar las posiciones exactas.

Por otro lado, las antenas directivas tienen un enfoque distinto. En este tipo de antenas se busca transmitir (o recibir) la mayor cantidad de energía posible hacia alguna dirección en concreto, es decir, se usa mayormente en comunicaciones donde se desea transmitir o recibir desde algún punto en particular (punto a punto). Las aplicaciones de estas principalmente se centran en comunicación satelital y tienen distintos usos en el ámbito militar, como el rastreo de objetivos por medio de radares [20]. Últimamente han ganado popularidad debido al incremento de los lanzamientos de micro y nanosatélites para distintos propósitos. Por ejemplo, el servicio de internet inalámbrico satelital es uno de los casos más mencionados en la actualidad, por la empresa Starlink [21] que es propiedad del magnate Elon Musk.

Es posible aseverar que el ejemplo más conocido de antenas directivas son los reflectores parabólicos, ya que estas son comúnmente vistas en el entorno cotidiano. En la Figura 2-3 se muestra una ilustración de una antena parabólica así como el flujo de las ondas electromagnéticas a través de ella. Debido a la geometría del disco en forma de parábola, todo haz recibido de forma perpendicular a este será reflejado hacia un punto focal (F), donde se coloca la parte sensible de la antena. Si se ve desde el enfoque de transmisión, todo haz emitido desde el punto focal hacia el plato será reflejado hacia una dirección perpendicular a la parábola, obteniendo una emisión directiva.

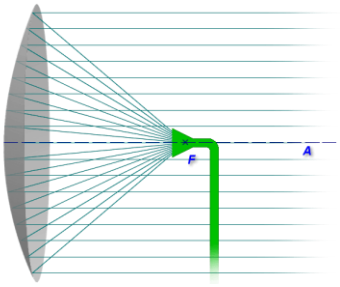


Figura 2-3. Diagrama de una antena parabólica. Fuente: [22].

Ahora bien, si por cualquier motivo[1] se desea ***reorientar*** el lóbulo hacia otra dirección, se debe realizar una rotación física de la antena y del plato para que apunten hacia la dirección deseada. Las antenas parabólicas con rotación física existen y son utilizadas pero, como cualquier sistema físico, tienen sus desventajas, por ejemplo: «baja» rapidez de reorientación; «poca» precisión; desgaste de los componentes físicos, fricción, etc., lo que implica tener que dar mantenimiento o reemplazar componentes; entre otros inconvenientes. Las antenas en arreglos de fase, que son explicadas en el siguiente apartado, solucionan estos entre tantos otros problemas.

[1] En muchos de los casos se requiere cambiar la orientación del lóbulo, ya sea para rastrear un objeto en movimiento o para mantener comunicación con un receptor o transmisor que no esté fijo (como los satélites de órbita baja [21]).

2.1.3 Antenas en arreglos de fase

Para comenzar, una antena en arreglo de fase es en esencia un grupo de antenas (de ahora en adelante llamadas «elementos») cuyo funcionamiento en conjunto sirve para actuar como una única antena. El haz y dirección de una antena en arreglo de fase puede ser modificado de forma electrónica, sin la necesidad de mover de manera física ninguno de los elementos que la componen [23]. Por supuesto, esto soluciona todos los problemas antes mencionados de tener un sistema de movimiento físico, además de que provee un tiempo de reorientación mucho menor y una alta precisión en comparación. En la Figura 2-4 se muestran algunos ejemplos de arreglos de fase.

Figura 2-4. Fotografías de algunas antenas en arreglos de fase. Fuente: [24].

Ahora se dará la explicación de su funcionamiento, ya que es de vital importancia tener claro cómo es que en conjunto forman un patrón de radiación altamente directivo, además de cómo consiguen una reorientación del haz sin un movimiento físico de los elementos [25]. Para comenzar el análisis, obsérvese el ejemplo en dos dimensiones de la Figura 2-5, donde una única antena emite ondas electromagnéticas a cierta frecuencia con muy poca directividad, lo que implica que el patrón de radiación es bastante amplio.

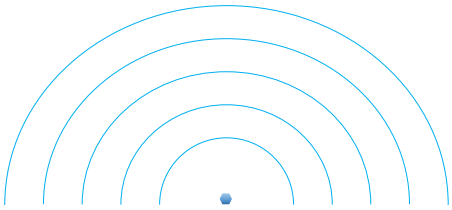


Figura 2-5. Propagación de ondas electromagnéticas emitidas por un elemento.

Por otro lado, obsérvese que en la Figura 2-6 se coloca otro elemento al lado que se tenía previamente. Ambos elementos transmiten a la misma frecuencia y al mismo tiempo, lo que ocasiona que ambas señales se interfieran mutuamente. Por el fenómeno de interferencia existen zonas en las que las ondas se anulan una con la otra (*interferencia destructiva*); aunque también hay aquellas en las que la energía de ambas señales se suma, formando así una zona principal de *interferencia constructiva* (delimitada con líneas grises).

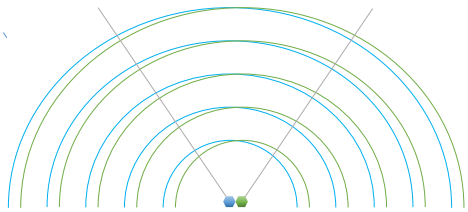


Figura 2-6. Propagación e interferencia de ondas electromagnéticas emitidas por dos elementos.

Siguiendo la misma filosofía, si se añaden aún más elementos como se ve en la Figura 2-7, se tiene una zona de interferencia constructiva bastante más estrecha y con mucha más energía (delimitada por líneas grises). Finalmente, es así como los elementos de un arreglo de fase en conjunto forman un lóbulo principal con una muy alta directividad.

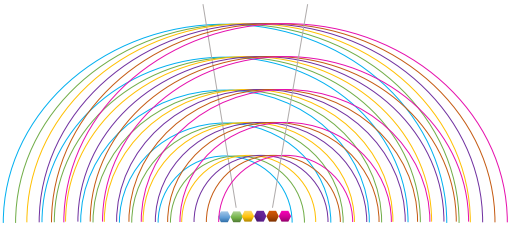


Figura 2-7. Propagación e interferencia de ondas electromagnéticas emitidas por seis elementos.

Ya que ha quedado claro cómo se genera un lóbulo principal altamente directivo, se discutirá cómo es posible modificar la orientación del haz formado por los elementos del arreglo, sin necesidad de un movimiento físico. Retomando el ejemplo de la Figura 2-6, el haz generado por la interferencia entre los dos elementos sigue una dirección completamente perpendicular, esto es debido a que las ondas emitidas por ambos elementos en esa ilustración tienen la misma ***fase***. Una *diferencia de fase* implica que una onda está viajando de forma adelantada o atrasada con respecto a otra, como se puede notar en la Figura 2-8, donde la onda de color rosa se encuentra desfasada 45° con respecto a la de color azul.

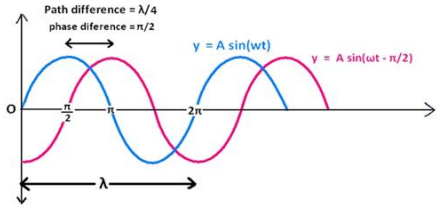


Figura 2-8. Dos ondas sinusoidales con diferencia de fase. Fuente: [26].

En la Figura 2-9 se ilustra qué sucede con la zona de interferencia constructiva si ocurre una diferencia de fase entre las ondas propagadas por ambos elementos. Tal como se observa, esta acción conlleva a una inclinación del haz generado, logrando así una modificación en la dirección del lóbulo principal.

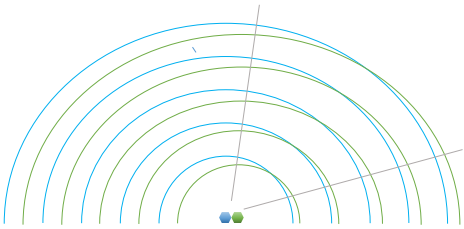


Figura 2-9. Propagación e interferencia de ondas electromagnéticas con desfase, emitidas por dos elementos.

El mismo fenómeno es igualmente aplicable para arreglos con más elementos, de tal manera que cada uno tenga una diferencia de fase y en conjunto se provoque un cambio en la orientación del haz hacia alguna dirección deseada (ilustrado en la Figura 2-10). El análisis se concluye entonces con la siguiente frase: ***"El control de la diferencia de fase de cada uno de los elementos del arreglo permite reorientar el lóbulo principal"***.

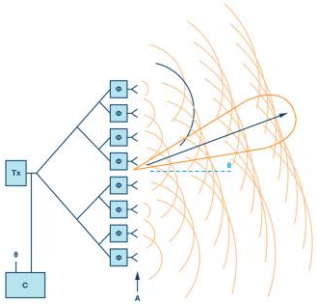


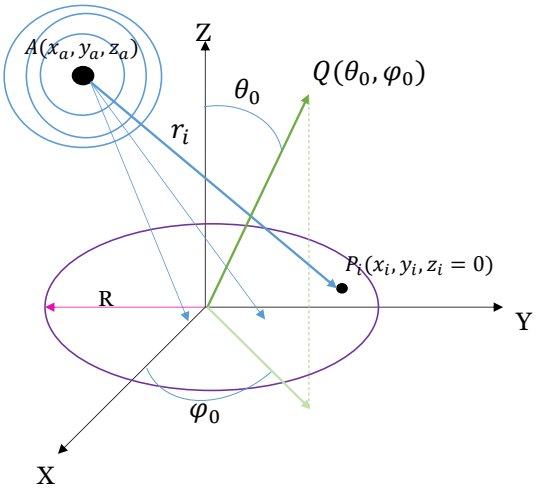
Figura 2-10. Diferencias de fase en varios elementos provocando una reorientación del haz. Fuente: [27].

Para finalizar esta sección, se menciona que hoy en día las antenas en arreglos de fase han adquirido gran popularidad, por las propias aplicaciones de las antenas directivas que fueron mencionadas anteriormente. Por tal razón en la actualidad se han diseñado diversos de tipos de arreglos: algunos que van desde una agrupación de antenas parabólicas de varios metros de tamaño [23], hasta arreglos de pocos centímetros cuyos elementos están hechos a partir de microsistemas electromecánicos [28] (MEMS, por sus siglas en inglés).

Para el presente trabajo, los arreglos de interés serán aquellos cuyos elementos son antenas de microcinta, específicamente aquellas controladas mediante diodos PIN [28] [29]. Los siguientes dos apartados proveen de un análisis más formal y matemático sobre este tipo de arreglos, aunque sin perder el enfoque al propósito del proyecto.

2.1.4 Expresiones matemáticas de funcionamiento

La Figura 2-11 muestra un diagrama simplificado en tres dimensiones de un arreglo de fase circular, en el cual se muestra un elemento desplazador (P_i), el alimentador (A), el haz incidente y el haz reflejado por los elementos en conjunto (Q), así como las nomenclaturas de algunos de los ángulos y parámetros asociados.



R: radio del arreglo
A: centro de fase del alimentador
$\boldsymbol{P_i}$**:** posición del elemento *i*
Q**:** haz reflejado con ángulos θ_0 y φ_0
$\boldsymbol{r_i}$**:** distancia entre el centro del alimentador y el elemento *i*.

$$\boldsymbol{r_i} = \sqrt[2]{(x_a - x_i)^2 + (y_a - y_i)^2 + (z_a - z_i)^2}$$

Figura 2-11. Diagrama de arreglo de fase y alimentador con parámetros.

La acción deseada en cualquier instante es reflejar la onda incidente hacia un vector cuya orientación está dada por el par de ángulos θ_0 y φ_0. θ_0 es el ángulo de la onda reflejada con respecto al eje Z, mientras que el ángulo φ_0 es con respecto al eje X. Matemáticamente hablando, la *distribución de fase* necesaria en cada elemento *i* para que la onda reflejada se dirija a los ángulos deseados θ_0 y φ_0 se define de la siguiente forma:

$$\varphi_{refle_i} = -kx_i \cos(\varphi_0)\, sen(\theta_0) - ky_i sen(\varphi_0) sen(\theta_0) \quad (2.a)$$

$k = \frac{2\pi}{\lambda}$, donde *k* es el número de onda y λ la longitud de onda.

Por otro lado, a la expresión (2.a) aún no considera la diferencia de *fase inducida* por la distancia que hay entre el alimentador y el elemento *i*, la cual se expresa como $\varphi_{ind_i} = -kr_i$. La posición del desplazador de fase del elemento *i* (P_{desp_i}) queda definida por la resta entre la *distribución de fase* y la *fase inducida* en el elemento:

$$\boldsymbol{P_{desp_i} = k[-x_i \cos(\varphi_0)\, sen(\theta_0) - y_i sen(\varphi_0) sen(\theta_0) + r_i]} \quad (2.b)$$

El resultado de la expresión (2.b) posteriormente es redondeado a un valor discreto, pues se debe tener presente que se trata de desplazadores con dispositivos de control finitos que le permiten una cantidad limitada de posiciones [30]. P_{desp_i} entonces se redondea al valor más cercano de $P_{cod_i}(2\pi/N)$, donde P_{cod_i} es el número o código de la posición discreta y N es el número de posiciones (fases) en función del número de bits del arreglo (b_a)[2].

[2] Por ejemplo, si se tiene un arreglo de $b_a = 2$, las $2^2 = 4$ posiciones de cada desplazador de fase serán 0°, 90°, 180° y 270°.

$$N = 2^{ba} \tag{2.c}$$

$$P_{cod} = \{0, 1, \dots, N-1\}$$

Respecto a las polarizaciones de los diodos PIN para las posiciones discretas del desplazador, estas están previamente definidas por el diseño y la estructura física del arreglo, por lo que para el sistema de control están dadas como una matriz constante que se puede denominar como «matriz/tabla de funcionamiento» o «matriz/tabla de polarización» [31].

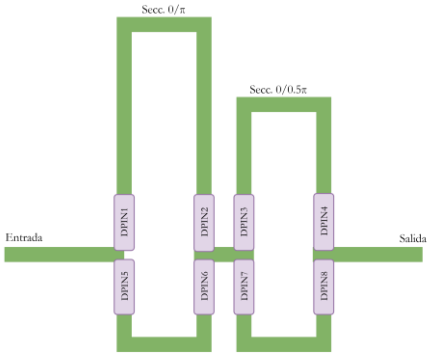


Figura 2-12. Desplazador de fase de dos bits con líneas conmutadas.

Por ejemplo, en la Figura 2-12 se tiene un desplazador de fase basado en líneas conmutadas; este contiene varias secciones, cada una genera un desfasamiento diferente. El control de los conmutadores (diodos PIN) permite cerrar o abrir los caminos necesarios para generar la fase deseada. Para este caso, la matriz de polarización tendría una estructura como la de la Tabla 2-1. Asimismo, en la Figura 2-13 se ilustran las polarizaciones y el recorrido para las cuatro posiciones discretas.

Tabla 2-1. Matriz de funcionamiento de un arreglo hipotético de dos bits.

Fase [°]	P_{cod}	DPIN 1	DPIN 2	DPIN 3	DPIN 4	DPIN 5	DPIN 6	DPIN 7	DPIN 8
0	0	OFF (0)	OFF (0)	OFF (0)	OFF (0)	ON (1)	ON (1)	ON (1)	ON (1)
90	1	OFF (0)	OFF (0)	ON (1)	ON (1)	ON (1)	ON (1)	OFF (0)	OFF (0)
180	2	ON (1)	ON (1)	OFF (0)	OFF (0)	OFF (0)	OFF (0)	ON (1)	ON (1)
270	3	ON (1)	ON (1)	ON (1)	ON (1)	OFF (0)	OFF (0)	OFF (0)	OFF (0)

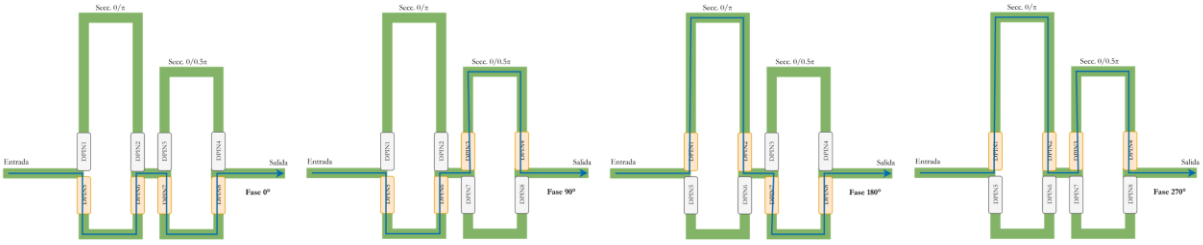


Figura 2-13. Posiciones para el desplazador de fase de dos bits con líneas conmutadas.

2.1.5 Distribución física de los elementos

La manera en que están organizados los elementos desplazadores dentro de un arreglo circular puede verse como una especie de malla. Obsérvese a continuación la Figura 2-14, la restricción matemática indica que el módulo de las coordenadas donde se ubica todo elemento (x_i, y_i) debe ser menor o igual al radio del arreglo (R) para que este forme parte del mismo, en caso contrario, el elemento en cuestión se encuentra fuera de las dimensiones y por ende no pertenece al arreglo.

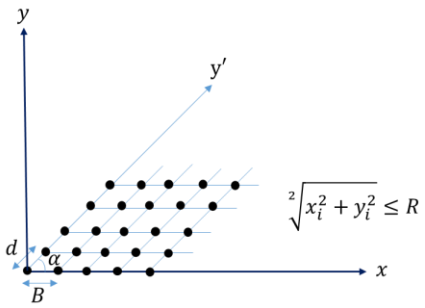


Figura 2-14. Ilustración de la distribución de algunos elementos dentro del arreglo.

A continuación se presenta una ilustración más detallada del arreglo completo (Figura 2-15), de esta puede observarse que la manera en que están organizados los elementos se puede ver también como un conjunto de hileras o columnas, las que tienen un ángulo de inclinación «**α**» respecto al eje x. Asimismo, la distancia que hay entre elemento y elemento dentro de una misma columna está dada por el parámetro «**d**», y la distancia que hay entre columna y columna es el parámetro «**B**».

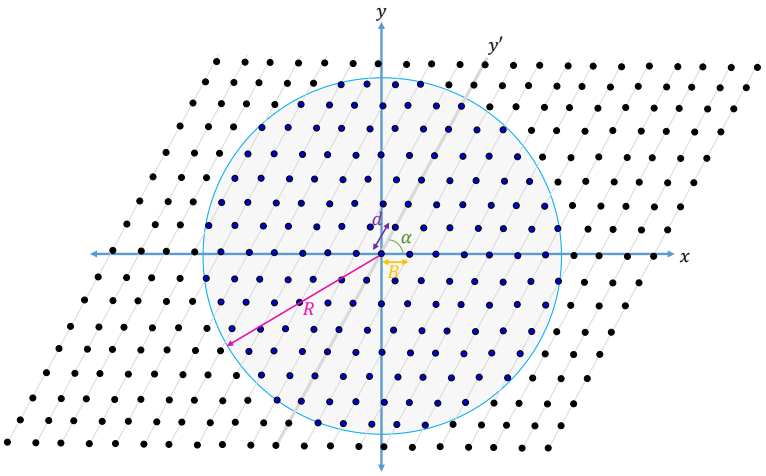


Figura 2-15. Parámetros y distribución de los elementos dentro del arreglo.

Con base en la Figura 2-15, se procede a hacer el análisis geométrico de uno de los puntos (elementos) más cercanos al origen, como lo ilustra la siguiente figura.

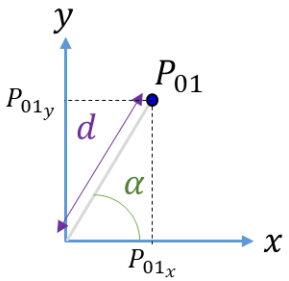


Figura 2-16. Coordenadas de un elemento cercano al origen.

Utilizando trigonometría para obtener las expresiones de la Figura 2-16, se puede llegar a que las coordenadas con respecto a cada eje son las siguientes:

$$P_{01_x} = dcos(\alpha)$$

$$P_{01_y} = dsen(\alpha)$$

Posteriormente se analizan los siguientes dos elementos pertenecientes a esa misma hilera de forma geométrica a través de la Figura 2-17:

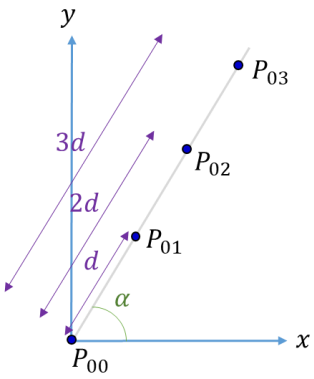


Figura 2-17. Columna de elementos que atraviesa el origen.

Las expresiones para las coordenadas de los elementos 2 y 3 de la hilera 0 también se pueden obtener con el uso de trigonometría y se muestran a continuación:

$$P_{02_x} = 2dcos(\alpha) \qquad P_{03_x} = 3dcos(\alpha)$$

$$P_{02_y} = 2dsen(\alpha) \qquad P_{03_y} = 3dsen(\alpha)$$

Se puede observar un patrón con respecto a las coordenadas de cada elemento perteneciente a una misma columna, por lo que las expresiones para las posiciones en ambos ejes se pueden generalizar, nombrando «**n**» al índice de fila, resultando en las expresiones siguientes:

$$P_{0n_x} = ndcos(\alpha)$$

$$P_{0n_y} = ndsen(\alpha)$$

Las expresiones anteriores sólo son válidas para la hilera 0 (que pasa por el origen), puesto que aún no se considera el desplazamiento horizontal que hay entre columna y columna, y para esto se hará un análisis geométrico de la columna 1 a partir de la Figura 2-18:

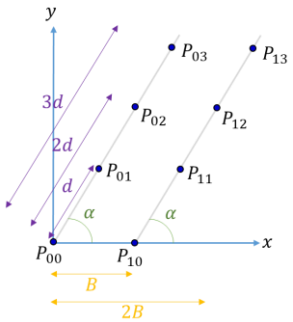


Figura 2-18. Ilustración de dos columnas de elementos.

De la Figura 2-18 se puede observar que el desplazamiento horizontal modifica los valores de las coordenadas en el eje *x*, mientras que las coordenadas en el eje *y* no se ven afectadas. Las expresiones para los elementos de la columna 1 se definen como:

$$P_{1n_x} = ndcos(\alpha) + B$$

$$P_{1n_y} = ndsen(\alpha)$$

Posteriormente, a través de la Figura 2-19 se hacen los análisis de las hileras 2 y 3 para intentar observar un patrón y poder generalizar las expresiones que definen las coordenadas para cualquier elemento ubicado en cualquier fila y columna. Se nombra al índice de columna como «m».

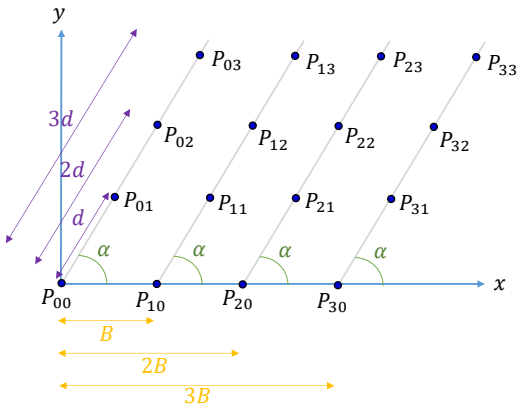


Figura 2-19. Ilustración de varias filas y columnas de elementos.

Las ecuaciones para las coordenadas de los elementos en las columnas 2 y 3 obtenidas a partir de la figura B-7 se definen a continuación:

$$P_{2n_x} = ndcos(\alpha) + 2B \qquad P_{3n_x} = ndcos(\alpha) + 3B$$

$$P_{2n_y} = ndsen(\alpha) \qquad P_{3n_y} = ndsen(\alpha)$$

Se observa que el incremento horizontal es proporcional al número de columna actual. Para generalizar, se nombrará «**m**» al índice de columna. Las expresiones finales para el cálculo de coordenadas del elemento $\boldsymbol{i}$ ubicado en los índices $\boldsymbol{m_i}$ y $\boldsymbol{n_i}$, quedan definidas por las expresiones (2.d) y (2.e), sujetos siempre a la condición (2.f).

$$\boldsymbol{x_i = x_{n_i m_i} = n_i dcos(\alpha) + m_i B} \tag{2.d}$$

$$\boldsymbol{y_i = y_{n_i m_i} = n_i dsen(\alpha)} \tag{2.e}$$

$$\sqrt{x_{n_i m_i}^2 + y_{n_i m_i}^2} \leq R \tag{2.f}$$

2.2 CONCEPTOS SOBRE DISEÑO DIGITAL

2.2.1 Tipos de circuitos digitales

Los circuitos de electrónica digital se dividen en dos tipos: combinacionales y secuenciales. En este apartado se explicarán sus respectivas características y diferencias.

Comenzando por los circuitos combinacionales, a grandes rasgos, son aquellos compuestos únicamente de compuertas lógicas si son analizados desde su nivel más bajo. Sin embargo, su diferencia principal con respecto a los circuitos secuenciales es que no poseen ningún tipo de «memoria». Dicho de otra manera, un sistema combinacional siempre tendrá una salida en función de las entradas actuales, sin importar lo que haya ocurrido con las entradas ni las salidas en ningún momento anterior.

Para ejemplificar a los sistemas de lógica combinacional, en la Figura 2-20 se muestra un sumador aritmético para dos números de 3 bits. El valor de la salida S siempre dependerá sólo de los valores actuales de A y B, sin importar lo que haya ocurrido anteriormente.

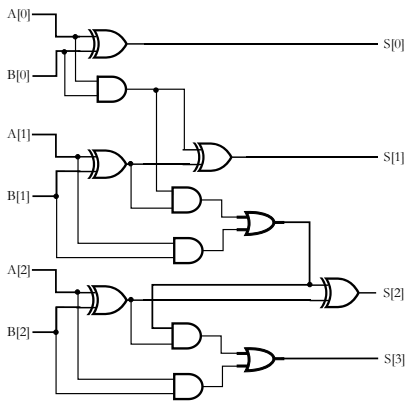


Figura 2-20. Sumador combinacional de 3 bits.

Por otra parte, los circuitos secuenciales pueden verse como un conjunto que puede contener circuitos combinacionales dentro de sí. La característica que los diferencia, como se mencionó, es que estos en concreto poseen «memoria», es decir, las máquinas secuenciales cuentan con registros (también llamados «flip-flops» (FF)) que almacenan información. El comportamiento de estos sistemas estará en función de las entradas y/o de lo que haya ocurrido previamente.

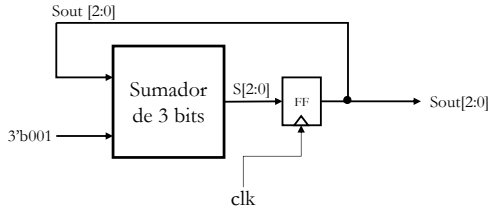


Figura 2-21. Contador secuencial.

Asimismo, este tipo de máquinas son controladas por una señal de reloj que los registros utilizan para «capturar» los valores que almacenan. Como ejemplo, en la Figura 2-21 se tiene un contador ascendente. Puede observarse que incluye al sumador combinacional de la Figura 2-20, aunque para poder hacer un conteo secuencial se debe «recordar» cuál es el valor actual de la cuenta e incrementarlo. El almacenamiento se realiza mediante los registros y la sincronización del conteo es controlada por una señal de reloj con nombre **clk**.

2.2.2 Análisis de tiempos en circuitos digitales

Debido a que todo componente electrónico no ideal tiene asociado cierto retraso temporal, los circuitos digitales tienen un *tiempo de ejecución* (también denominado como *retardo* o *latencia*). En este apartado se analizarán los tiempos de ejecución para ambos tipos de circuitos.

- Circuitos combinacionales:

En estos sistemas, la latencia del circuito se obtiene calculando el tiempo de propagación mayor, que resulta ser el camino que más componentes atraviesa entre las entradas y salidas (el cual se conoce como *camino crítico*). Por ejemplo, en la Figura 2-22 el circuito combinacional cuenta con distintos caminos que conectan los bits de entrada con los de salida, pero el que más componentes lógicos atraviesa es el que está resaltado de color azul, por lo que la suma de retardos de las compuertas que son parte de este camino ($3t_{and} + 2t_{or}$) será la latencia máxima de aquel sumador combinacional.

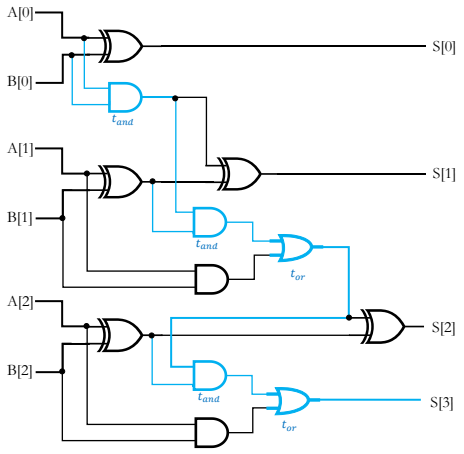


Figura 2-22. Camino crítico del sumador de 3 bits.

- Circuitos secuenciales:

El análisis de tiempos para estas máquinas es ligeramente más complejo. En un principio, la frecuencia de la señal de reloj será la que determine la rapidez del circuito. Por otro lado, esta frecuencia no puede ser incrementada de forma indiscriminada, pues debe respetarse la latencia de los bloques combinacionales que formen parte del circuito secuencial.

Retomando el ejemplo de la Figura 2-21, si se asume que el retardo que hay en el sumador combinacional es de 12 ns y el periodo de la señal de reloj de 10 ns, las señales se comportarán como se observa en el diagrama de tiempos de la Figura 2-23. De inmediato se observa que la salida (**Sout**) del circuito ya no se comporta como debería, lo que es producto de tener una señal de reloj cuya frecuencia es más rápida que la frecuencia máxima a la que puede funcionar el sumador (determinada por el inverso de su latencia).

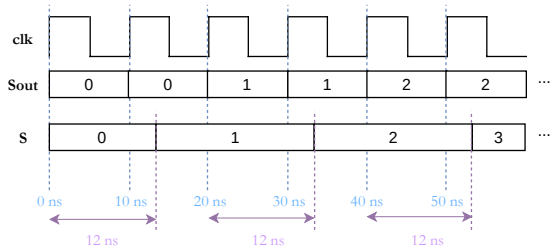


Figura 2-23. Diagrama de tiempos del contador secuencial.

Dicho lo anterior, se puede concluir que la frecuencia a la que puede operar una máquina secuencial está limitada por los retardos de sus bloques combinacionales. Expresado de forma matemática es que surge la restricción (2.g), donde f_{ms} es la frecuencia de reloj de la máquina secuencial y t_{lc} es la latencia de la lógica combinacional.

$$f_{ms} < \frac{1}{t_{lc}} \tag{2.g}$$

Ahora bien, hasta el momento se ha mencionado que las máquinas secuenciales sólo están limitadas por sus componentes combinacionales, lo que a su vez sugiere que los «flip-flops» son componentes ideales que no tienen restricción alguna; por supuesto, esto no es cierto. Los registros, al igual que cualquier otro componente real, tienen restricciones que también deben respetarse. Obsérvese la Figura 2-24 que incluye el diagrama del FF y a un lado el diagrama de tiempos.

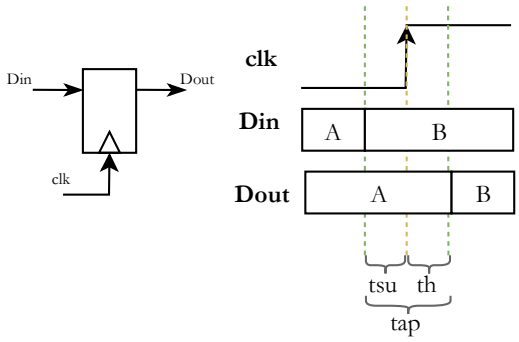


Figura 2-24. Diagrama de tiempos de un «flip-flop».

El diagrama de tiempos muestra el comportamiento de un «flip-flop» con respecto a una entrada cambiante y a la llegada de un flanco de reloj. El tiempo de «setup» (t_{su}) es aquel que indica cuánto debe permanecer estable la señal de entrada al registro antes de la llegada del flanco de reloj; por el contrario, el tiempo de «hold» (t_h) es el tiempo mínimo en que el valor de entrada deberá permanecer estable después de la llegada del flanco. El cumplimiento de ambos valores garantiza que el dato almacenado dentro del FF sea el deseado. Si se incumple alguno, es posible que el dato almacenado no sea el correcto, generando incertidumbre en el valor guardado (fenómeno también conocido como «metaestabilidad»).

Al valor conformado por la suma del tiempo de «hold» y del tiempo de «setup» se le conoce como tiempo de «apertura» (t_{ap}), haciendo alución a la captura de una imagen en una cámara fotográfica. Considerando las restricciones de los registros, entonces la frecuencia de operación de un sistema secuencial queda limitada por la restricción (2.h), que no es más que la condición anterior (2.g) añadiendo los valores asociados a los registros.

$$f_{ms} < \frac{1}{t_{lc} + t_{su} + t_h} \tag{2.h}$$

2.2.3 Segmentación encauzada

Dentro del diseño de sistemas digitales existen procesos combinacionales en los que varios bloques operan los datos de forma progresiva, es decir, la salida de un módulo se toma como entrada para un módulo posterior. Ejemplo de esto es la Figura 2-25, donde se tiene un proceso compuesto por 4 bloques.

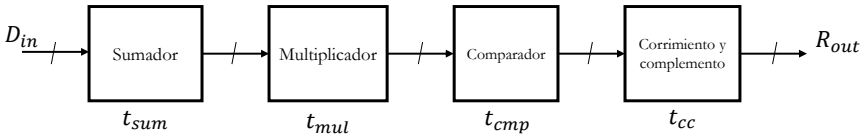


Figura 2-25. Proceso combinacional compuesto de cuatro etapas.

Si se hace el análisis de la latencia en este proceso, esta está definida por la suma de los retardos de cada módulo. En otras palabras, a la salida le tomará $t_t = t_{sum} + t_{mul} + t_{cmp} + t_{cc}$ en tener el resultado a partir de un cambio en las señales de entrada. Ahora se analizará qué sucede si, en vez de tener los módulos conectados de forma directa, se añaden registros a la salida de cada uno, como se ve en la Figura 2-26.

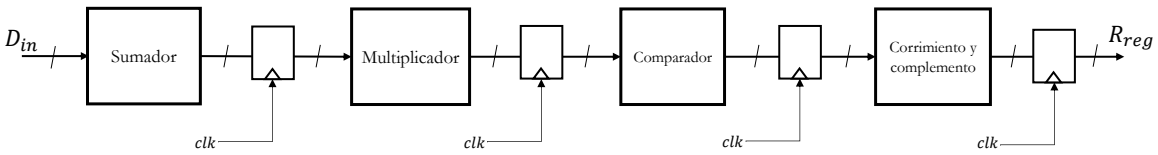


Figura 2-26. Proceso combinacional con registros a la salida de cada etapa.

Con la ilustración anterior, el proceso que anteriormente era sólo combinacional se convirtió en un proceso secuencial. Este nuevo proceso es controlado por la señal de reloj **clk**. Para comprender mejor el flujo de las señales a través de los

bloques, en la Figura 2-27 se tiene un diagrama de tiempos para un dato de entrada llamado «Ent1». Nótese cómo en este caso se necesitan 4 ciclos de **clk** para que la salida final esté lista ($t_t = 4T_{clk}$).

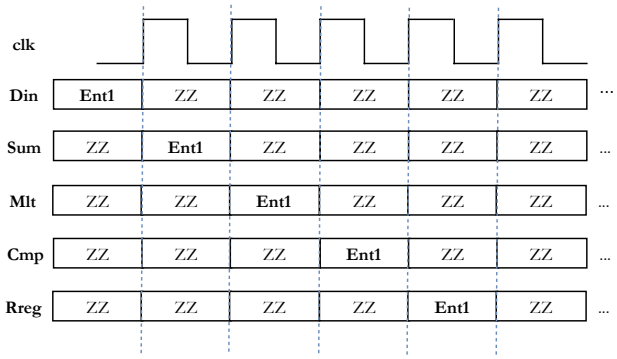


Figura 2-27. Diagrama de tiempos del proceso combinacional.

Dada esta nueva estructura y recordando la restricción (2.g), el periodo de la señal de reloj **clk** estará limitado por la latencia del módulo que tenga el mayor retardo. Asimismo, en el diagrama de tiempos se nota que mientras los datos de «Ent1» se están procesando en un algún módulo, el resto no está trabajando con ningún dato nuevo.

A partir de esta condición es que se implementa la técnica conocida como segmentación encauzada (encauzamiento o «pipeline»). Esta técnica tiene como objetivo añadir cierto paralelismo temporal, de manera que si se cambian las entradas al proceso en cada ciclo de reloj, cada módulo estará trabajando con datos distintos durante cada periodo. Obsérvese el diagrama de tiempos de la Figura 2-28, donde se añaden datos nuevos a la entrada del sistema en cada ciclo de reloj. Es posible notar que, ya que ha pasado una cantidad de ciclos de reloj igual al número de etapas, todos los bloques se encuentran trabajando al mismo tiempo con datos distintos; antes de este momento no todos estarán funcionando.

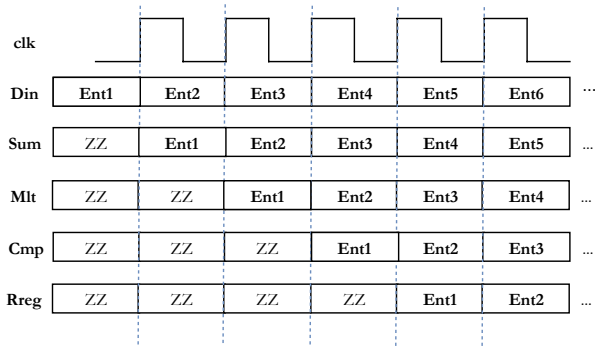


Figura 2-28. Diagrama de tiempos del proceso con segmentación encauzada.

Igualmente, el primer resultado final del proceso se obtiene una vez que han pasado 4 ciclos de reloj, pero de ese momento en adelante, un resultado final nuevo se tendrá listo en cada ciclo. Esta es la característica principal del «pipeline», que permite aumentar el desempeño (rendimiento o «throughput») del sistema. La única «desventaja» es que se tiene una latencia mayor en comparación a la estructura propuesta en la Figura 2-25, donde allí se presenta la *latencia mínima* posible desde que los datos se presentan a la entrada hasta que se obtiene el resultado final.

2.2.4 Formatos numéricos

Las primeras interrogantes que surgen al trabajar con sistemas digitales y con el sistema de numeración binario tiene que ver con la representación y magnitudes de los números: «¿Cómo se deberán representar los datos calculados?, ¿se debería usar formato de punto fijo o de punto flotante?; siendo el primer caso: «¿cuántos bits asignar a la parte fraccionaria?». La respuesta correcta depende directamente de la aplicación del sistema a diseñar.

En el presente apartado se describen los dos formatos numéricos principales para representar números reales en formato binario: formato de punto fijo y formato de punto flotante. Además de sus descripciones, también se mencionan las ventajas y desventajas que cada uno posee.

- Formato de punto fijo

En esta representación, la cantidad de bits que conforman a un número real puede ser dividida en dos partes: una cuyos bits pertenecen a la parte entera y otra en la que se almacena la parte fraccionaria. Por ejemplo, el número 11.25 en fomarto de 8 bits Q4[3] queda de la siguiente forma:

Magnitud en decimal	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}
Valor en binario	1	0	1	1	.	0	1	0	0

Como se puede observar, la posición de cada bit indica su magnitud. Para la parte izquierda, entre más alejado del punto binario más valor tendrá; mientras que para la parte derecha es el caso contrario, entre más alejado del punto binario más pequeño será su valor. La conversión de un número en este formato a decimal se realiza mediante la suma de las respectivas magnitudes de cada bit: $2^3(1) + 2^2(0) + 2^1(1) + 2^0(1) + 2^{-1}(0) + 2^{-2}(1) + 2^{-3}(0) + 2^4(0) = 11.25$.

Las ventajas de trabajar con formato de punto fijo es que las operaciones aritméticas se pueden implementar como si se tratara de números enteros. El diseñador es quien conoce qué parte del total del número representa fracciones y qué parte representa los enteros, y por ende es el encargado de interpretar los datos. Véase a continuación un breve ejemplo de suma aritmética con punto fijo:

$$\begin{array}{l} 001101_{(.)}101 + \\ 001001_{(.)}001 \\ \_\_\_\_\_\_\_ \\ 010110_{(.)}110 \end{array}$$

Ignorando el punto, el resultado se obtiene como si se sumaran en decimal 109 + 73 = 182. Considerando el punto fijo, la operación es 13.625 + 9.125 = 22.75. El resultado en binario es correcto para ambos casos.

Por otro lado, para la representación de números negativos normalmente se utiliza el complemento a 2 (C2). El C2 de un número en binario es una alternativa que permite simplificar las operaciones aritméticas elementales, aunque se debe reservar un bit del tamaño de palabra para poder utilizar esta representación. El complemento a dos de un número representado en binario se puede obtener de la siguiente manera:

$$X_{C2} = X_{C1} + 1; \quad donde\ X_{C1} = not(X)$$

Para regresar un número en complemento a dos a su valor equivalente positivo se tienen las siguientes expresiones:

$$X = not(X_{C2} - 1), \qquad o\ bien\ X = [X_{C2}]_{C2}$$

La desventaja que tiene el formato de punto fijo es que, si se desea una alta exactitud en la representación de los valores, se debe optar por incrementar el número de bits. A manera de ejemplo, en la Tabla 2-2 se encuentran algunos tamaños de palabra con los valores máximos y la resolución (el incremento más pequeño entre dos números). Se reserva un bit de la parte entera para la representación en C2.

Tabla 2-2. Parámetros de representación numérica en punto fijo.

Formato	Valor máximo	Resolución
16b Q8	127.99609375	0.00390625
32b Q16	32767.999984	0.00001525
64b Q32	2147483647.99	0.0000000002328

[3] Q*i* es una notación que indica que *i* son los bits correspondientes a la parte fraccionaria de un número real representado en binario.

- Formato de punto flotante

Las computadoras modernas digitales intentan representar a los números reales con una muy alta exactitud. El formato de punto flotante es la representación más utilizada por este motivo. Este permite tener una alta exactitud en la representación de los números y su estructura está basada en la representación exponencial (o científica). Si se tiene que x es un número en formato de punto flotante, su valor en decimal queda definido como:

$$x = \pm S \times 2^E$$

Donde E es un número entero denominado exponente, mientras que S es conocido como el significando o mantisa. Dada la expresión anterior, es posible imaginar que el punto decimal «flota» a la posición posterior de donde esté ubicado el primer dígito distinto de cero de la representación en decimal del número, y por ello es que se denomina de tal manera. Por ejemplo, el número 5.5 en punto flotante puede ser representado como:

$$5.5 = (+)1.011_b \times 2^2$$

Debido a la gran popularidad de esta representación, el «Institute of Electrical and Electronics Engineers» (IEEE) creó un estándar conocido como «IEEE 754» [32], en el que se establecen cinco formatos básicos para representar números en formato de punto flotante. Los tres más importantes se muestran en la Tabla 2-3 junto con sus especificaciones. Asimismo, el estándar establece números especiales como el infinito positivo y negativo, el cero signado y ciertos valores no numéricos (NaN). De estos casos especiales no se hablará más a detalle por simplicidad, aunque se considera importante mencionar su existencia.

Tabla 2-3. Parámetros del estándar IEEE 754 para representación en punto flotante. Fuente: [32].

Nombre	Bits del exponente	Bits de mantisa	Valor máximo
Binary 16	5	10	65471.7747
Binary 32	8	23	3.402e+38
Binary 64	11	52	$1.99\overline{9} \times 2^{-1023}$

Volviendo al ejemplo anteriormente mencionado, el valor de 5.5 en formato de punto flotante estándar «Binary 32» queda representado de la siguiente forma:

Signo(1 bit)	Exponente (8 bits)	Mantisa (23 bits)
0_b (+)	10000001_b (2^2)	$01100000000000000000000_b$ (1.375)

Cuya conversión a decimal se realiza tal como se explicó de manera previa:

$$x_d = (+)(1.375 \times 2^2) = +5.5$$

Como se observó en la Tabla 2-3, los valores máximos de cada formato del estándar 754 se incrementan de forma aún más exponencial que como ocurre en el caso del formato de punto fijo (presente en Tabla 2-2). El incrementar ligeramente el número de bits en esta representación permite tener un aumento mayor en exactitud y valores máximos.

Por otro lado, dado que los valores almacenados en binario no se pueden interpretar de forma directa sino que se tienen que «decodificar», las operaciones en formato de punto flotante son más complicadas en comparación con las de punto fijo. No obstante, hoy en día es el formato que más se utiliza y por ende se han diseñado maneras eficientes de ejecutar las operaciones aritméticas, que además están presentes en todos los procesadores de la actualidad [32].

2.2.5 Lenguajes y estilos de descripción de hardware

Los lenguajes de descripción de hardware (HDL, por sus siglas en inglés) surgieron como una manera de describir de forma fácil las especificaciones de diseño y funcionalidad de un circuito digital. El objetivo inicial era que los requerimientos plasmados en los archivos fuente fueran transformados a un diagrama esquemático (este proceso inicialmente se realizaba de forma manual). Con el paso de los años se dio lugar a programas computacionales que pudieran traducir automáticamente un archivo HDL a un circuito estructural. A estos programas se les nombró «sintetizadores» que, de forma simple, producen una descripción estructural de bajo nivel para un circuito cuyo funcionamiento está basado en una descripción hecha con un HDL.

Fue así como los lenguajes de descripción de hardware y los sintetizadores pasaron a formar parte de las herramientas de diseño asistido por computadora (CAD, en inglés), con dos lenguajes de descripción que predominaron: Verification Logic (*Verilog*) y VHSIC[4] Hardware Description Language (*VHDL*). A continuación se hablará de forma breve sobre la historia de Verilog y su evolución hacia SystemVerilog, ya que este es el lenguaje que se utilizará para el diseño del sistema.

Verilog surgió en el año de 1984 creado por Phil Moorby y Prabhu Goel. Inicialmente se creó con propósito de simulación y modelado de hardware como propiedad intelectual de Gateway Design Automation. En 1990 Cadence Design System adquirió la compañía Gateway y descubrieron que Verilog tenía potencial como lenguaje para convertirse en un estándar, tomando como motivación el planteamiento de que, si este permanecía como un lenguaje cerrado, el único que dominaría la industria como estándar sería VHDL. En 1991 se creó el proyecto Open Verilog International (posteriormente renombrado como Accellera), donde la documentación de Verilog pasó a ser de dominio público y en 1995 se convirtió en un estándar del IEEE [33].

Como todo estándar, Verilog ha sufrido actualizaciones y cambios a lo largo de su historia. Uno de los cambios más importantes ocurrió en 2005, cuando se presentó «System Verilog» (SV). Siendo este un lenguaje «nuevo» que incluía una gran variedad de características nuevas y funcionalidades adicionales a las ya preexistentes en Verilog. Además de corrección de errores, a SV también se le añadieron capacidades para funcionar como lenguaje de verificación de hardware (HVL). Dado que SV tenía las capacidades suficientes para modelar, diseñar, simular e implementar circuitos digitales, en 2009 Verilog y System Verilog fueron fusionados en un único estándar hoy conocido únicamente como *SystemVerilog*.

Con respecto a los estilos de descripción que son permitidos en los HDL, es posible categorizarlos en dos grandes grupos enlistados a continuación:

1. Estructural: este estilo describe la estructura del circuito con código referente a compuertas lógicas, registros y las interconexiones entre estos para formar el sistema deseado [34]. Este estilo también conocido como Register Transfer Level (RTL).
2. Comportamental: describe el comportamiento del diseño en términos de mayor abstracción. El código utilizado en este tipo de descripción puede llegar a asemejarse al presente en un lenguaje de programación de alto nivel. Mediante el código se detalla la manera en que trabaja el sistema, pero en ocasiones es posible ignorar cómo estará conformado en su interior [35]. De estilos comportamentales se encuentran al menos los dos siguientes:
   Flujo de datos: este estilo permite definir el flujo que tendrán las señales entre los módulos que conforman el sistema a través de expresiones tipo «when, case, if-else», así como ecuaciones booleanas.
   Funcional: en este se expone la manera en que funciona el sistema con expresiones de tipo «for», así como operadores aritméticos directos «+, -, *».

Para el diseño del sistema en este trabajo se planea utilizar principalmente un estilo de tipo «flujo de datos». Aunque no es el estilo de descripción de más bajo nivel, permite tener un buen control sobre la generación de bloques lógicos dentro de los módulos. Sin embargo, se piensa recurrir a la utilización de un estilo completamente funcional en módulos cuya operación sea simple y de poca relevancia.

[4] VHSIC proviene del término en inglés 'Very High Speed Integrated Circuit'.

CAPÍTULO 3
DISEÑO DEL SISTEMA

En el presente capítulo se habla inicialmente sobre los requerimientos que fueron establecidos para el sistema de control, además de un primer acercamiento a las propuestas de arquitectura factibles de implementar, para posteriormente relatar a profundidad el diseño del sistema.

La estructura del capítulo es la siguiente: se comienza con una sección dedicada a los requerimientos, posteriormente al concepto de diseño y propuestas de arquitectura, y en seguida se habla un poco más a detalle sobre la arquitectura elegida; después se inicia el diseño con la elección del formato numérico a utilizar y se continúa con todo el desarrollo del sistema.

3.1 DETERMINACIÓN DE REQUERIMIENTOS

Puesto que la creación de este sistema es una necesidad concreta de los titulares del grupo de electrónica de alta frecuencia y microondas de la Facultad de Ingeniería de la UNAM, el Dr. Oleksandr Martynyuk es quien ha establecido los requerimientos deseados para el sistema de control a diseñar. Estos son enlistados y comentados a continuación:

- El número de elementos del arreglo será un máximo de 1500 ($e_{a_{máx}}$).
- El sistema deberá proporcionar las señales de control a todos los elementos en un tiempo de respuesta de un microsegundo.
- La configuración por elemento mínima será de 2 bits ($b_{a_{mín}}$) con 4 diodos PIN por elemento ($d_{e_{mín}}$).
- La configuración por elemento máxima será de 5 bits ($b_{a_{máx}}$) con 32 diodos PIN por elemento ($d_{e_{máx}}$).

Los requerimientos establecidos pueden ser desglosados para traducirse a términos más particulares asociados al sistema de control, es decir, al número de señales de control necesarias para cada uno de los casos. La cantidad de señales de control (b_c) se puede calcular mediante la siguiente expresión: $b_c = e_a d_e$, donde e_a es el número de elementos del arreglo, y d_e es el número de diodos por elemento.

- Caso 1: 1500 elementos de 4 diodos PIN.

$$b_{c_1} = (1500)(4) = 6{,}000\ bits\ de\ control$$

- Caso 2: 1500 elementos de 32 diodos PIN.

$$b_{c_2} = (1500)(32) = 48{,}000\ bits\ de\ control$$

3.2 PROPUESTAS PRELIMINARES

En la sección de requerimientos se observó que la cantidad de bits de control necesaria se encuentra en el orden de los miles hasta las decenas de miles para el caso más grande. La cantidad de bits, que está directamente asociada a la cantidad de pines, hace imposible el implementar el controlador en un único dispositivo, ya que no existe alguno que se acerque a la cantidad de terminales necesaria para cubrir la demanda total de señales de control. Por tal motivo, se debe pensar en una alternativa para conseguir esa cantidad de señales, además de que se cumpla con el tiempo de respuesta solicitado.

Después de un análisis de distintas alternativas y considerando los trabajos de la investigación del estado del arte (secc. 1.2), se planeó la utilización de registros de corrimiento para distribuir las señales de control hacia todos los elementos del arreglo. Un dispositivo lógico reconfigurable sería el encargado de mandar las señales hacia los registros de forma serial y estos tendrían las salidas paralelas, por lo que cada registro de corrimiento se encargaría de controlar tantos elementos como sus pines de salida le permitieran.

Aunque en un principio suena como una alternativa factible de implementar, se harán algunos cálculos para verificar si es posible realizarlo de tal manera. Para los cálculos se pensará en utilizar registros de 32 bits y se comienza con establecer qué parámetros son los que definen a los registros en cuanto a tamaño y frecuencia.

$$Señales\ (bits)\ de\ control\text{: } \boldsymbol{b_C} = e_a d_e$$

$$Tiempo\ de\ respuesta\text{: } \boldsymbol{t_r}$$

$$Frecuencia\ de\ respuesta\ \boldsymbol{f_r} = \frac{1}{t_r}$$

$$Pines\ FPGA: \boldsymbol{P}$$

$$N\acute{u}mero\ de\ registros\ de\ corrimiento: \boldsymbol{n_{SR}}$$

$$Tama\tilde{n}o\ (bits)\ del\ registro\ de\ corrimiento: \boldsymbol{b_{SR}}$$

$$Frecuencia\ del\ registro\ de\ corrimiento: \boldsymbol{f_{SR}}$$

La cantidad total de registros debe ser menor o igual a los pines disponibles de la(s) FPGA, con el fin de que cada terminal pueda servir como salida hacia un registro de corrimiento. Este parámetro además define qué tanto es posible paralelizar la tarea:

$$n_{SR} \leq P \quad (1)$$

Dado el caso en el que se utilicen todos los pines: $n_{SR} = P \quad (2)$. Entonces, el tamaño que de los registros de corrimiento será:

$$b_{SR} = \frac{b_c}{n_{SR}} \ (3)$$

La frecuencia a la que deben trabajar estará en función del «tiempo de distribución», definido por la resta del tiempo de respuesta total menos el tiempo de cálculo: $t_d = t_r - t_c$.

$$f_{SR} = \frac{b_{SR}}{t_d} \rightarrow sust.(3) \rightarrow f_{SR} = \frac{\frac{b_C}{n_{SR}}}{t_d} = \frac{b_C}{t_d n_{SR}} \quad (4)$$

Despejando n_{SR} de (3) y (4):

$n_{SR} = \frac{b_C}{b_{SR}} \ (5)$ Número de registros en función de los bits de control y el tamaño de los registros.

$n_{SR} = \frac{b_C}{t_d f_{SR}} \quad (6)$ Número de registros en función de los bits de control, tiempo de respuesta y frecuencia de los registros.

Regresando a la desigualdad (1), se establecen las siguientes condiciones con (5) y (6):

$$\frac{b_C}{b_{SR}} \leq P \quad (7)$$

$$\frac{b_C}{t_d f_{SR}} \leq P \quad (8)$$

$$Despejando\ (8) \rightarrow f_{SR} \geq \frac{b_C}{t_d P} \ (9)$$

$$Despejando\ (7) \rightarrow b_{SR} \geq \frac{b_C}{P} \ (10)$$

Recordando las siguientes condiciones mínimas que deberán respetarse (secc. 3.1): el arreglo constará de 1500 elementos con 4 diodos PIN cada uno, y el sistema debe ser capaz de mandar las señales de control a todos ellos 1 vez cada microsegundo. Se elige una FPGA XC7A100T [36] de ejemplo, con disponibilidad de hasta 300 pines. Asumiendo también que el tiempo de cálculo $t_c = 0.3\ \mu s$.

$$b_C = (1500)(4) = 6000\ bits\ de\ control$$

$$P = 300$$

La expresión (9) indica cuál deberá ser la frecuencia mínima de los registros de corrimiento para satisfacer las condiciones anteriores, si se utilizan solo 190 registros de corrimiento:

$$f_{SR} \geq \frac{b_C}{t_d n_{sr}} = \frac{6000}{(0.7x10^{-6})(190)} = 45.12\ MHz$$

$$f_{SR} \geq 45.12\ MHz$$

Mientras que la expresión (10) indica de qué tamaño deberán ser los registros de corrimiento:

$$b_{SR} \geq \frac{b_C}{n_{sr}} = \frac{6000}{190} = 31.57\ bits$$

Después de búsqueda exhaustiva de la disponibilidad de registros de corrimiento comerciales, se encontró que para el caso de los registros de 32 bits, aquellos que se encuentran a la venta se sitúan en una frecuencia máxima de operación de 16 MHz (p. ej. el modelo HV5623 [37]), lo que resulta en menos de la mitad de rapidez requerida. Por tanto, es posible asegurar que no es viable implementar la arquitectura del sistema con registros comerciales.

Concluyendo lo anterior, se debe encontrar una alternativa de crear los registros de corrimiento, o bien, una alternativa de arquitectura. Las siguientes dos secciones hablan sobre el par de propuestas de arquitecturas que fueron planteadas a partir del análisis de este apartado.

3.2.1 Arquitectura 1. «Maestro-esclavos»

La Figura 3-1 ilustra el diagrama de la arquitectura propuesta. Está compuesta de una FPGA maestra encargada de realizar todos los cálculos matemáticos para cada uno de los elementos, posteriormente los envía codificados y de forma paralela-serial hacia todos los registros de corrimiento, los cuales también estarán hechos a partir de dispositivos de lógica reconfigurable; las salidas de los dispositivos esclavos son las que controlan la etapa de potencia que entrega los voltajes requeridos para la polarización directa o inversa de los diodos PIN.

Sacando provecho de que los registros estarán hechos de dispositivos de lógica reconfigurable y que ya no serán simples registros de corrimiento (como se había establecido originalmente en la sección 3.2), se contará con lógica combinacional de decodificación interna para cada elemento, de tal manera que la maestra les envíe las fases codificadas de forma serial a los esclavos, y estos la decodifiquen según la matriz de funcionamiento (explicada al final de sección 2.1.4). Al hacer esto se reduce la cantidad de datos que tienen que ser serializados y, en consiguiente, el tiempo de respuesta, así como el tamaño de los registros. La Figura 3-2 muestra la estructura interna de uno de los dispositivos que estará funcionando como esclavo. La interfaz de usuario servirá para ingresar los valores de parámetros a la tarjeta maestra para realizar la configuración, así como enviar los ángulos de orientación deseados.

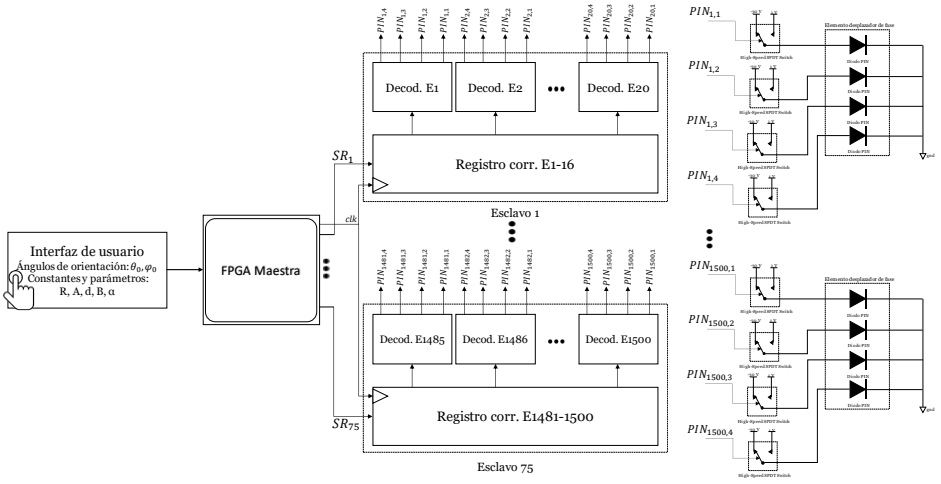


Figura 3-1. Diagrama de arquitectura tipo maestro-esclavos.

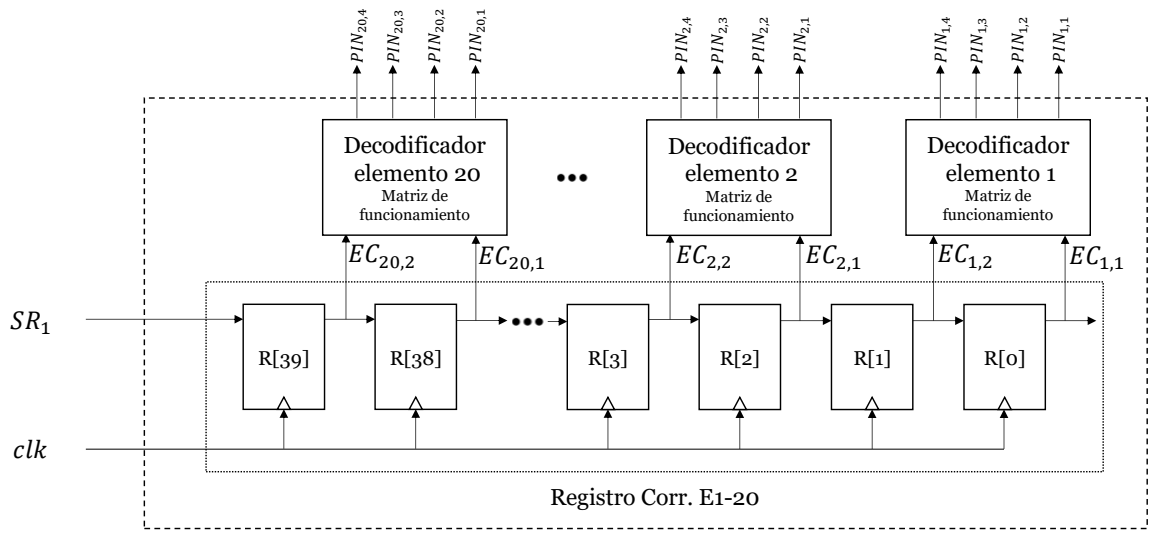


Figura 3-2. Diagrama de la estructura interna de un dispositivo esclavo.

Respecto al cómo se realizarán los cálculos matemáticos para los desplazadores de fase, existe al menos un par de maneras de ejecutarlas, y se tendrá que realizar un análisis sobre cuál se ejecuta en menos tiempo y se puede paralelizar mejor, ya que por ahora no se ha detallado sobre el tiempo que tarde la FPGA maestra en realizar los cálculos.

- Tabla de búsqueda: la FPGA contendrá guardados los resultados de las operaciones matemáticas de forma previa, así, para cada combinación de ángulos solamente se buscará cuál es el resultado que le corresponde, y posteriormente se realizará la codificación según la matriz de funcionamiento del arreglo. A priori se estima que este método será el más rápido, pero el cuánto se podrá paralelizar dependerá de la cantidad de recursos que utilice y la cantidad de combinaciones que se piensen considerar.
- Funciones lógicas: consistirá en implementar las operaciones matemáticas a través de funciones lógicas, de esta manera se tendrá que realizar al menos una operación por elemento. Se considera que sería más lenta que la tabla de búsqueda, pero la utilización de recursos debería ser menor y eso conllevaría a una mejor implementación paralela, posiblemente utilizando algún tipo de encauzamiento (explicado en secc. 2.2.3).

Para finalizar la primera propuesta, se mencionan algunas de las características que poseería el sistema, agrupadas en las que son consideradas como posibles ventajas y como posibles desventajas. Posteriormente se hace un breve análisis de costo de implementación de la arquitectura.

Potenciales ventajas:

- La sincronización de todos los registros estará dada por la FPGA maestra sin una mayor complicación.
- Los dispositivos lógicos donde se implementen los registros serán optimizados y únicamente se necesita diseñar uno, ya que los demás serán réplicas exactas.
- Solamente se requieren ingresar los valores de configuración y ejecución en la FPGA maestra, además de que solamente ella los almacena y procesa.

Potenciales desventajas:

- La cantidad de esclavos máxima está limitada por la cantidad de pines disponibles de la FPGA maestra. Si se convirtiera en un problema, una posible solución sería implementar más de una maestra trabajando en conjunto para incrementar los pines, o cambiarla por una tarjeta con una mayor cantidad de pines.
- Que todos los cálculos radiquen en una sola tarjeta podría suponer insuficiencia de recursos lógicos, afectando el paralelismo o la velocidad de respuesta.

Un aspecto muy importante a tener en cuenta respecto a cada arquitectura es el costo de implementación. Cabe mencionar que para las estimaciones que se harán se asumirá que los dispositivos elegidos cuentan con los recursos lógicos suficientes para ejecutar las tareas mencionadas en ambas arquitecturas, que de momento se desconocen.

Hablando de implementar los esclavos en FPGA de bajo costo, se proponen FPGA de Xilinx modelo XC7A12T, cuyo precio por unidad ronda los 13 dólares [36]. Según sus especificaciones, cuentan con 150 pines de entrada/salida. Por la cantidad de salidas, un esclavo puede dar señal de control a 30 elementos de 4 diodos cada uno, y los pines extra se reservan para las señales de reloj y de entrada.

Con esta implementación, se necesitarían un total de 50 FPGA para cubrir la demanda de 6000 señales de control, es decir, la FPGA maestra debe tener al menos 55 pines disponibles (los de los esclavos y cinco extras reservados para señales de control), además de la cantidad de recursos suficiente para la realización de todos los cálculos y el almacenamiento de datos. La tarjeta XC5VLX155T XMF5 con FPGA de Xilinx cuenta con un máximo de 135 pines de entrada/salida. Su precio unitario es de aproximadamente 280 dólares [38].

Se concluye finalmente que, para implementar la primera arquitectura, en el caso del arreglo más simple se necesitarían 650 dólares para los esclavos y 280 dólares para la FPGA maestra, siendo un total aproximado de 930 USD.

3.2.2 Arquitectura 2. «Módulos paralelos independientes»

La Figura 3-3 ilustra el diagrama de la segunda propuesta de arquitectura. Esta consistiría en un conjunto de dispositivos lógicos (FPGA) que trabajarán de forma paralela e independiente, aunque sincronizada. Cada uno atenderá a un conjunto de elementos del arreglo y dentro de cada uno de ellos se realizarían los cálculos necesarios para la etapa de configuración y la etapa de ejecución.

Se tendrían tantos módulos como fueran necesarios para cubrir el número de elementos del arreglo. Al igual que en la propuesta anterior, se buscaría encontrar cuál es la manera más conveniente de realizar las operaciones matemáticas para consumir el menor tiempo posible y paralelizarlo al máximo. Dado que cada módulo constará de un dispositivo dedicado, se tendrán que optimizar los recursos para que cubra/controle los más elementos que se puedan, sin dejar de satisfacer las condiciones puestas. En la Figura 3-4 se encuentra un esquema con la estructura genérica interna de uno de los bloques independientes.

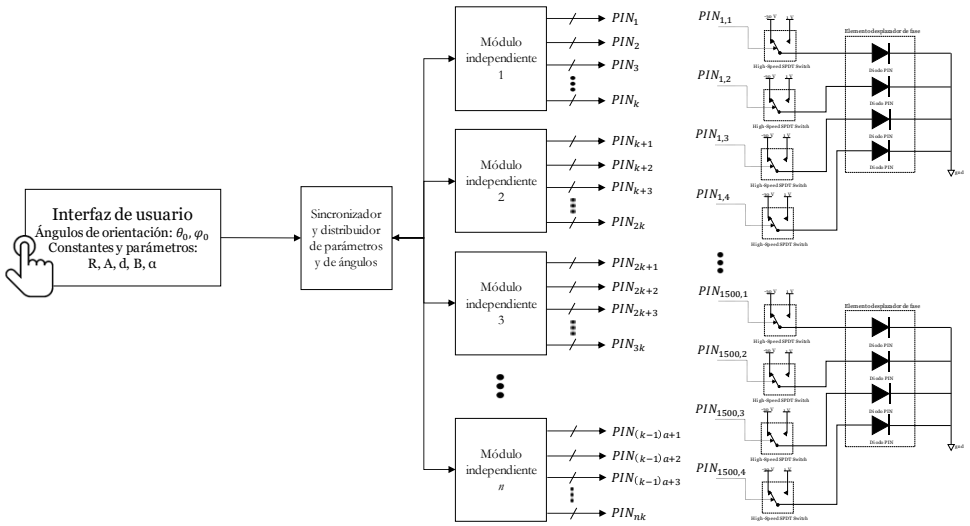


Figura 3-3. Diagrama de la arquitectura con módulos independientes paralelos.

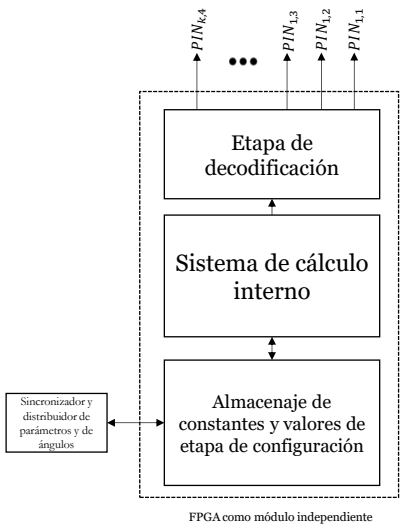


Figura 3-4. Diagrama de la estructura interna de uno de los módulos independientes.

A continuación se mencionan algunas de las características que presentaría la segunda propuesta, agrupadas como ventajas y desventajas:

Potenciales ventajas:

- Se elimina la limitación de módulos por la cantidad de pines de una FPGA maestra. En este caso se pueden tener tantos como sean necesarios para cubrir el total.
- Los datos ya no tendrían que ser enviados de forma serial, por lo que ya no se necesitan registros de corrimiento y así se incrementa la frecuencia de operación, logrando cumplir más fácilmente con las restricciones temporales.
- La escalabilidad se vuelve más sencilla de realizar por la inexistencia de las limitaciones anteriores.
- Si el conjunto de elementos que cubre un módulo es cercano físicamente, es probable que haya manera de simplificar los cálculos que realiza.

Potenciales desventajas:

- Se tiene que buscar una manera de sincronizar los módulos para que sus tareas las realicen de tal forma.
- Los módulos deben conocer de alguna manera qué elementos del arreglo son aquellos que les corresponde.
- Cada módulo debe recibir la información de las etapas de configuración y de ejecución, procesarla y hacer los cálculos pertinentes. Quizás, para la etapa de configuración se podrían precargar los datos que serían calculados, de esta manera se simplificarían las máquinas y los recursos ahorrados se podrían utilizar para la etapa de ejecución.
- Ya que ahora no una sola tarjeta almacena la información, cada módulo debe guardar los datos asociados al conjunto de elementos que se encuentre controlando.
- Cada módulo tiene una configuración distinta, puesto que las regiones que controlarán serán únicas para cada uno. Esto puede ser propenso a errores de configuración y conexión.
- La complejidad y utilización de recursos se incrementa en esta arquitectura, por lo que seguramente cada módulo cubra un número reducido de elementos en comparación a los módulos esclavos de la arquitectura 1.

Respecto a la estimación de presupuesto, para esta propuesta se necesitan dispositivos que tengan un equilibrio entre los pines disponibles y la cantidad de recursos lógicos, aunque se recuerda que para esta estimación se dará por hecho que estos serán suficientes para lo que se vaya a implementar, por lo que la cantidad de pines sigue siendo el factor más limitante.

Después de una investigación, se decidió escoger la FPGA XC7S100 que cuenta con 76,800 celdas lógicas y un máximo de 400 pines. El costo del circuito integrado es alrededor de 82 dólares [39]. En este caso, si se ocuparan 380 pines de la FPGA como salidas y los demás se reservaran para las señales entrantes, se necesitarían 16 dispositivos trabajando en paralelo para cubrir 6000 señales de control. El costo de implementación de esta arquitectura sería de aproximadamente 1300 dólares.

3.3 ELECCIÓN Y PROFUNDIZACIÓN SOBRE LA ARQUITECTURA ELEGIDA

Analizando las potenciales ventajas, desventajas y costos de implementación de las dos arquitecturas propuestas anteriormente, se decidió optar por la Arquitectura 1. «Maestro-esclavos» ya que, con las estimaciones hechas, de principio se considera que esta presenta un mejor equilibrio entre las características positivas y negativas, además de un costo menor a la segunda. Algunos de los motivos se mencionan a continuación: primero, se evita el problema de sincronización entre todos los módulos independientes, pues en la arquitectura «maestro-esclavos» la sincronización está a cargo de la única tarjeta maestra; la interfaz de usuario solamente se comunicará con una tarjeta en vez de con todos los módulos; y finalmente, la mayor carga de trabajo recaerá sobre un solo dispositivo, ya que los dispositivos esclavos servirán únicamente como decodificadores y distribuidores de las señales de control, mas no se piensa realizar cálculos en ellos.

Previamente, en la Figura 3-1 se mostró un diagrama de bloques de la arquitectura propuesta, enfocada al caso de 1500 elementos de 4 diodos PIN. En la ilustración se observan las distintas etapas por las que estará conformada de forma general. En la presente sección se darán más detalles y especificaciones de los módulos internos basados en aquella figura.

- Módulos esclavos

Los dispositivos esclavos estarán conformados por un registro de corrimiento y lógica de decodificación, la cual decodificará el valor de fase correspondiente a las respectivas señales de polarización para los diodos, según como lo indique la matriz de polarización del arreglo. La cantidad de esclavos se expresará con la variable n_{es}, mientras que la cantidad de elementos que controlará cada esclavo será expresado como e_{es}. Pensando en trabajar con $n_{es} = 75$, se necesitan 80 pines para las salidas que controlarán la polarización de los diodos, y al menos 5 extra para las señales de control y comunicación restantes. Tal disposición requiere que cada esclavo posea un $e_{es} = 20$ en el caso de 4 diodos. Considerando una comunicación serial a una frecuencia de 150 MHz, se realizan los siguientes cálculos:

  - Para el caso de 4 fases discretas por elemento, se pueden codificar mediante 2 bits, por lo que, para controlar los 20 elementos, el tamaño del registro de corrimiento debe ser de 40 bits.
  - Si se coloca una señal de reloj de 150 MHz, se tiene un tiempo de distribución para la parte del registro de $\frac{40}{150x10^6\ Hz}$, dando un total de **0.266 µs.**
  - La lógica combinacional de decodificación deberá constar, respectivamente, de 20 módulos que conviertan las señales de 2 bits a las 4 señales de control que les corresponden a los diodos de cada desplazador. Es una lógica muy simple de implementar que debería consumir no más de unos pocos ns.

Añadiendo ambos valores, los módulos esclavos consumirán un tiempo total aproximado de **0.35 µs**.

- FPGA maestra

Esta será el módulo principal y el cerebro del sistema, pues será quien se encargue de realizar todos los cálculos y almacenar toda la información referente al arreglo. Para un mejor entendimiento, el funcionamiento se ha dividido en dos etapas nombradas como «*etapa de configuración*» y «*etapa de ejecución*».

La *etapa de configuración* consiste en obtener y procesar la información referente al arreglo, es decir, utilizando los parámetros: radio del arreglo (R), distancia vertical entre elementos (d), distancia horizontal entre elementos (B), inclinación del eje Y' (α), posición del alimentador (A) y longitud de onda (λ). A partir de estos valores se calcularán las posiciones de cada uno de los elementos (x_i, y_i), así como la distancia que hay entre ellos y el alimentador (r_i), ya que son los valores necesarios para la posterior etapa de ejecución, en donde se hará el cálculo de la operación (2.b). Esta etapa se ve como previa al funcionamiento activo de la antena, por lo que el tiempo de respuesta no es crucial y solamente se necesitará ejecutar una sola vez por arreglo.

Para llevar a cabo dichas tareas será necesario un sistema de cálculo y uno de almacenamiento de datos, además de una interfaz de usuario por donde se recibirán los parámetros antes mencionados. La Figura 3-5 muestra un diagrama de bloques que ilustra los módulos que serán necesarios para la etapa de configuración.

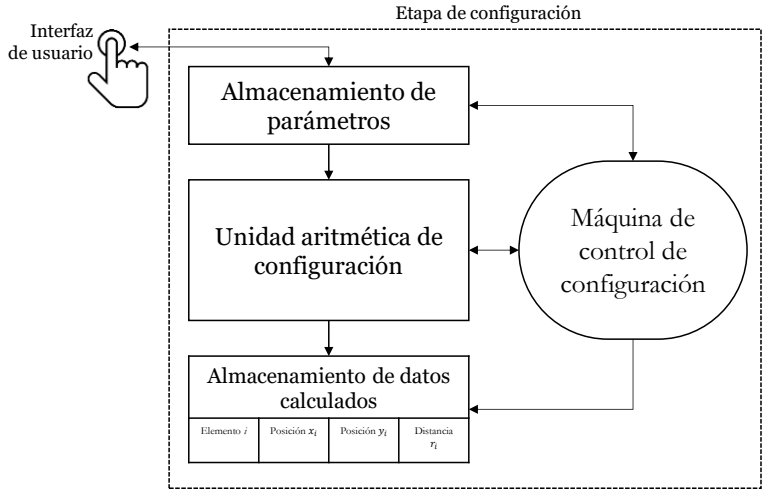


Figura 3-5. Diagrama preliminar de la etapa de configuración del sistema.

La segunda etapa es la *etapa de ejecución*, en esta es cuando se deben hacer los cálculos para mandar las posiciones de forma serial hacia los registros; aunque claro está que antes de cualquier operación, se deben recibir los ángulos a los que se desea reorientar el haz (θ_0, φ_0). Puesto que el propósito de este sistema es trabajar en el orden de MHz, no tendría sentido que los ángulos fueran ingresados de forma manual mediante la interfaz de usuario. Se propone que estos sean enviados desde algún otro sistema a través de algún protocolo con una velocidad de transferencia acorde, que se calcula en el siguiente párrafo.

Dando por hecho que el sistema cumple con el tiempo de respuesta solicitado, es necesario recibir un nuevo par de ángulos de orientación cada microsegundo (según los reqs. de secc. 3.1). Si cada ángulo se envía en formato de punto fijo, con 3 bits para la parte entera y 16 para la fraccionaria (lo que permite representar de 0 a 2π rad), se necesita una tasa de transferencia de $(19 \times 2)\ bits\ /\ 1\ \mu s$, resultando en una velocidad de 38 Mbps. El protocolo Serial Peripheral Interface (SPI) con tasas de hasta 60 Mbps [40] resulta más que suficiente como propuesta de protocolo de comunicación para recibir los ángulos de orientación.

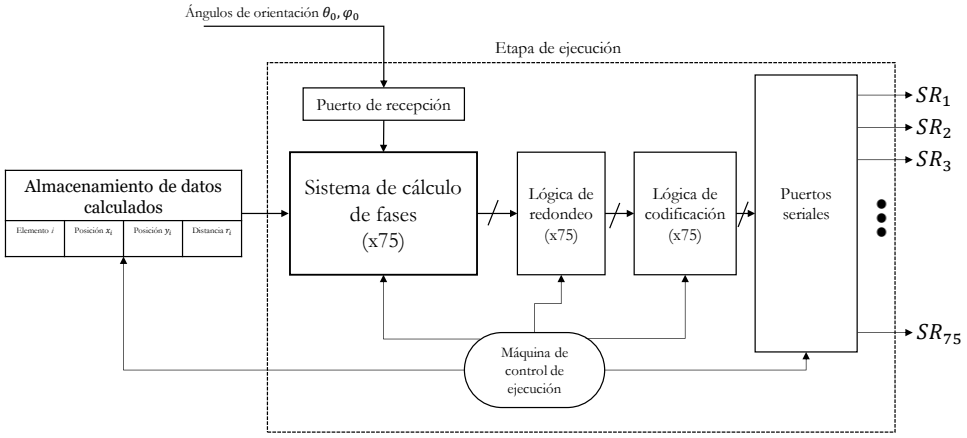


Figura 3-6. Diagrama preliminar de la etapa de ejecución del sistema.

El proceso de ejecución se puede observar en la Figura 3-6, que básicamente consta de tres partes: para el cálculo de la fase se realiza la operación matemática (2.b). Posteriormente a la obtención de cada fase, seguirá la etapa de redondeo y luego la codificación, donde se tomará el resultado con magnitud real y se redondeará a la fase más cercana alcanzable por el arreglo, para después codificarse y enviarse por un puerto serial hacia el dispositivo esclavo correspondiente. La etapa final consiste en los puertos seriales, quienes se encargarán de enviar de forma organizada la información codificada de los desfasamientos calculados, para el total de los elementos de cada dispositivo esclavo.

Para esta etapa el tiempo es crucial, y se esperaría que la resolución de la operación matemática sea la parte más compleja, y por ende la que más requiera tiempo. Se podría buscar un tiempo máximo de 300 ns para las operaciones matemáticas, dejando un margen de 200 ns para los redondeos, las codificaciones y el envío de datos. Esto daría un tiempo total de **0.5 µs** para la FPGA maestra, sumado a los 0.35 µs de los esclavos, quedaría un margen de 0.15 µs que podría ser considerado como el tiempo de conmutación reservado para los conmutadores tipo Single Pole Double Throw (SPDT), consiguiendo así el tiempo límite de 1 µs. En la Figura 3-7 se observa la estructura completa de la FPGA maestra.

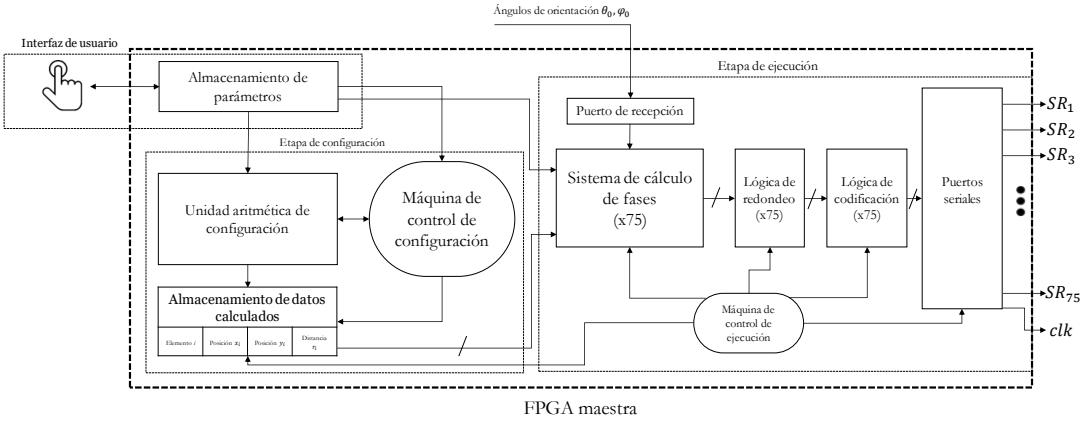


Figura 3-7. Estructura interna preliminar de la FPGA maestra.

3.4 ELECCIÓN DEL FORMATO NUMÉRICO

Para poder elegir el formato adecuado se necesita tener claro cuál es el objetivo principal del sistema: en este caso, a grandes rasgos se requiere que, a partir de los datos asociados a cada elemento del arreglo y los ángulos de orientación, realizar la operación matemática (2.b), cuyo resultado después será redondeado a la fase discreta más cercana al valor real. Dicho de otra manera, se entiende que es posible tener un margen de error, gracias principalmente al redondeo que se realiza después de la operación matemática. El error presente en el resultado de la fase calculada con el formato numérico elegido para el sistema deberá ser menor a 0.098 rad. Este valor proviene a partir del siguiente análisis:

De acuerdo a los requerimientos (secc. 3.1) el arreglo más complejo será de 5 bits, lo que, según la expresión (2.c), se traduce a $(2^5 = 32)$ fases discretas. La resolución se calcula dividiendo 2π radianes entre el número de fases, obteniendo para tal caso una fase cada 0.19635 radianes o cada 11.25°. Con base en esto, se puede proponer que el error en el resultado de la expresión (2.b) sea menor a la mitad de la resolución máxima, es decir, menor a $0.19635/2 = 0.098175$ rad. Un error más pequeño que tal valor permite que la fase obtenida por el formato numérico elegido, cuyo resultado no difiera en más de ± 0.098175 rad con respecto al resultado calculado en formato flotante, sea redondeada al mismo valor discreto al que sería redondeado con la fase obtenida en punto flotante.

Volviendo al formato numérico, trabajar con valores de punto fijo permite simplificar el diseño y ejecución de las operaciones aritméticas y del procesamiento de datos en general, aún más si se utiliza el complemento a 2 (C2) como representación de números negativos. Por consiguiente, se propone que los números estén representados en formato de punto fijo de 32 bits: 16 bits para la parte entera y 16 bits para la fraccionaria. Esta configuración daría unos valores máximo y mínimo de 32,767.99998 y -32,768, con una resolución de ± 0.0000152.

Se realizará una simulación con el fin de verificar que este formato numérico cumple con el error máximo establecido. Mediante el uso de Matlab se hará la siguiente prueba: se obtendrán los datos correspondientes a los elementos de un arreglo de prueba de 1499 elementos; posteriormente se calcularán y redondearán las fases de cada uno de ellos para un par de ángulos de orientación. Se considerará como un arreglo del número de bits más grande: $b_a = 5$.

Primero se realizará con el formato numérico que maneja Matlab, el cual es de punto flotante de precisión doble (64 bits); y posteriormente se realizará la misma prueba haciendo todos los cálculos con el formato elegido de punto fijo. El objetivo es comparar ambos resultados para ver si en los dos casos se obtienen las mismas fases codificadas, o si bien el error acumulado de las operaciones con formato de punto fijo resulta en una diferencia considerable de las fases finales.

Tabla 3-1. Algunas fases codificadas en punto flotante y punto fijo para un arreglo de ejemplo.

Vector fases cód. p. flot.	…	20	17	14	21	16	12	8	5	…
Vector fases códg. p. fijo	…	20	**16**	14	21	16	12	**7**	**4**	…

La Tabla 3-1 muestra algunos de las fases redondeadas obtenidas en ambos experimentos. Puede observarse que efectivamente existen algunas diferencias en los valores obtenidos con ambos formatos. De las 1499 fases obtenidas, 417 de ellas no coincidieron entre ambos experimentos, lo que representa un 28 % del total.

No obstante, el vector de diferencias absolutas entre ambos vectores de fases mostró tener un valor máximo de 1, implicando que las fases que no coinciden nunca se diferencian por más de una unidad (es decir, si la fase codificada obtenida en p. flotante es 25, la fase obtenida con p. fijo estará entre 24, 25 o 26, pero nunca mayor o menor).

Se observó las diferencias se presentaron en casos en los que el resultado de la fase real se encontraba cercano a los límites entre dos fases discretas, lo que conlleva a que el error acumulado de hacer las operaciones en punto fijo resulte en redondear hacia una fase distinta. Es importante tener presente que en la gran mayoría de casos forzosamente existe un error entre la fase real y la fase discreta debido al redondeo, independientemente del formato numérico utilizado. Si la fase real se encuentra cercana al punto medio entre dos discretas, el error entre elegir la fase superior o la inferior no será tan distinto.

Para comprender mejor lo estipulado en el párrafo anterior, en la Figura 3-8 se ilustra un ejemplo hipotético: supóngase un arreglo cualquiera de 2 bits (4 fases: 0, 90, 180 y 270 [°]). Se muestran las etapas de cálculo, redondeo y codificación tanto en punto flotante como en punto fijo. Las fases redondeadas terminan siendo distintas, pero lo importante de la ilustración es notar que en ambos casos existe un error entre la fase original y la fase discreta. Si bien el de los cálculos en punto fijo es ligeramente más grande, los errores solo difieren en 0.66° o 0.012 rad.

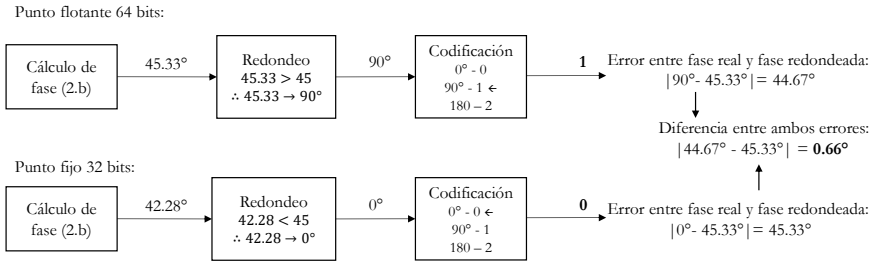


Figura 3-8. Ejemplo de cálculo y redondeo de fases para punto flotante y punto fijo.

Con el análisis anterior se concluye que los resultados comprueban que las diferencias de fases no son lo suficientemente grandes para causar un error considerable en la orientación de la antena, por lo que el formato de 32 bits Q16[5] será el utilizado para representar y procesar los datos dentro del sistema de control.

3.5 DISEÑO DE LA ETAPA DE CONFIGURACIÓN

De acuerdo con la arquitectura presentada en el apartado 3.3, la primera etapa del funcionamiento del sistema se refiere al reconocimiento de los elementos y sus respectivos datos (x_i, y_i, r_i; secc. 2.1.4), en función de los parámetros de la geometría del arreglo. En esta sección se tratará el diseño de dicha etapa, tal como está resaltada en la Figura 3-9.

[5] Q*n* es una notación que indica que *n* son los bits correspondientes a la parte fraccionaria de un número real representado en binario.

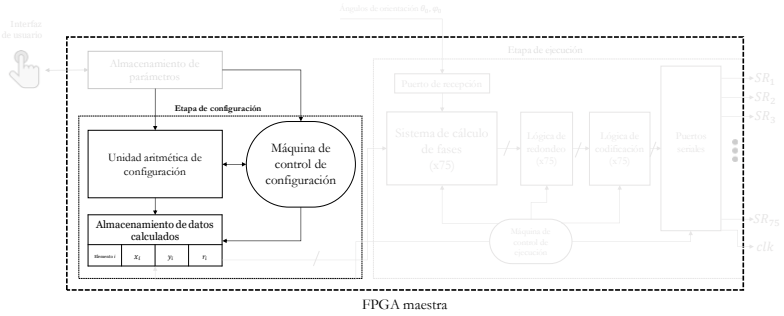


Figura 3-9. Etapa de configuración resaltada sobre estructura de FPGA maestra.

Es posible visualizar el flujo de la etapa de configuración como un algoritmo cuyo objetivo es guardar en memoria los datos de los elementos pertenecientes al arreglo. Según las expresiones matemáticas obtenidas en la sección 2.1.5, se puede ejecutar mediante un barrido por los índices de columnas y de filas a partir de un par de índices iniciales (**VIF**, **VIC**), obteniendo las posiciones de cada elemento y verificando si pertenece al arreglo, para entonces almacenar sus datos correspondientes. El pseudocódigo del algoritmo de configuración se muestra a continuación:

```
Constantes R, B, d, alfa, xa, ya, za; //Parámetros del arreglo y coordenadas del alimentador
Constantes VIF, VIC; // índices iniciales negativos que queden fuera del arreglo
Constantes VFF = -VIF; VFC= -VIC; // Finales positivos
Num_elementos = 0; // Contador de número de elementos del arreglo, inicializado en cero

for (c = columna_inicial; c <= columna_final: c++) { // Barrido de columnas
        for (f = fila_inicial; f <= fila_final; f++) { // Un barrido completo de filas para cada columna
                offsetX = c*B; // Número de columna por separación en eje X
                PosX = f*d*cos(alfa) + offsetx; //Posición en X del elemento n
                PosY = f*d*sen(alfa); //Posición en Y del elemento n
                modulo = raíz(PosX^2 + PosY^2);
                Si el módulo es menor o igual al radio del arreglo { // Elemento pertenece al arreglo
                        Guarda su PosX dentro del VectorPosX;
                        Guarda su PosY dentro del VectorPosY;
                        DistRn = raíz((xa – PosX)^2 + (ya – PosY)^2 + (za)^2); // Calcula dist. a alimentador
                        Guarda su DistRn dentro del VectorDistRn;
                        Incrementa en 1 el Num_elementos; // Puesto que se encontró un elemento
                }
        }
}
```

El pseudocódigo anterior fue programado en Matlab para verificar que funcionase correctamente, realizando simulaciones de arreglos con distintos parámetros; también para graficar los elementos encontrados, así como la malla generada por todos aquellos que son «visitados» por el barrido. Una de las gráficas obtenidas por la simulación se encuentra en la Figura 3-10.

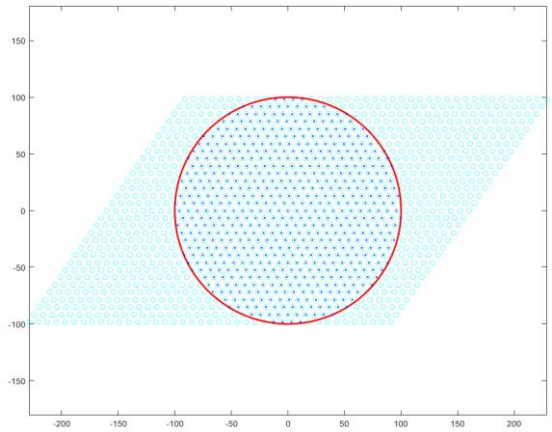


Figura 3-10. Gráfica de la simulación de cálculo de posiciones de los elementos. Los puntos azules representan los elementos que entran dentro de la circunferencia del arreglo, y los círculos cian son todos aquellos que fueron considerados por el barrido.

3.5.1 Unidad aritmética de configuración

En la Figura 3-11 se encuentra ubicada la unidad aritmética dentro de la estructura interna de la FPGA. Para implementar las operaciones requeridas por el algoritmo de configuración (establecido en la sección 3.5) se necesita el diseño de tres operaciones aritméticas básicas: suma, resta, y multiplicación; aunque igualmente se requiere del cálculo de raíz cuadrada y de funciones trigonométricas. En esta sección y sus respectivos apartados se explica cómo fueron implementados los bloques aritméticos, y se finaliza con la interconexión de estos para conformar a la unidad aritmética.

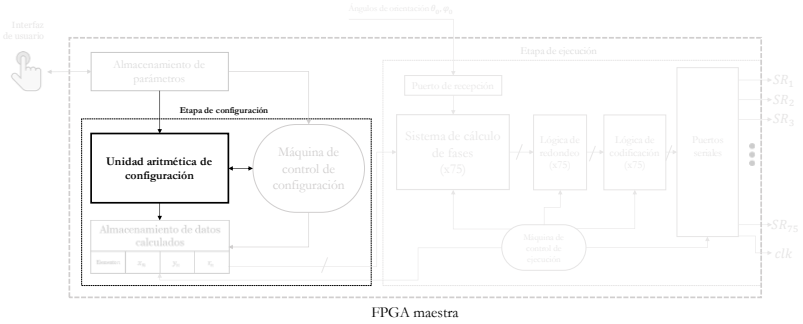


Figura 3-11. Unidad aritmética de configuración resaltada sobre la estructura interna.

3.5.1.1 Módulos de suma, resta y multiplicación

Inicialmente se diseñaron los bloques de suma, resta y multiplicación de forma estructural, pese a que el lenguaje de descripción permite la utilización directa de estos operadores mediante el estilo funcional, haciendo que se sinteticen de manera automática. Es necesario verificar si el hardware generado con los operadores directos funciona como se necesita y si no supera en recursos en comparación con una implementación estructural, en caso contrario, se mantendrá el uso de los módulos descritos de forma estructural.

Un módulo de suma y uno de resta generados por el uso de operadores directos demostraron funcionar de forma correcta mediante simulaciones, además de que ambos utilizaron una menor cantidad de recursos en comparación a los bloques estructurales, como se puede ver en la Figura 3-12, que muestra las tablas de utilización de recursos para el mismo sumador de 32 bits en estilo funcional y en RTL. Por ende, estas operaciones se mantendrán en estilo funcional.

Name	Slice LUTs (20800)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)
N SumRTL	51	23	51	96

Name	Slice LUTs (20800)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)
N SumFunc	32	8	32	96

Figura 3-12. Comparación de la utilización de recursos para un sumador de 32 bits en estilo RTL y estilo funcional.

Por otro lado, las implementaciones de la multiplicación en forma funcional y estructural sí tuvieron diferencias; utilizando el operador directo se obtuvieron resultados que difirieron con los obtenidos por la implementación estructural.

Por tal motivo, para el caso de la multiplicación se utilizará el multiplicador descrito estructuralmente, el cual está hecho a partir de sumadores de 32 bits y algunas compuertas lógicas. La estructura base se muestra en la Figura 3-13, misma que fue extrapolada para trabajar a 32 bits. La multiplicación de 2 números de 32 bits en Q16 da como resultado un número de 64 bits en Q32; pero siempre que se esté seguro de que el resultado sea un número que no exceda 32 bits, se puede truncar el resultado tomando los 16 bits menos significativos de la parte entera y los 16 bits más significativos de la parte fraccionaria del resultado original.

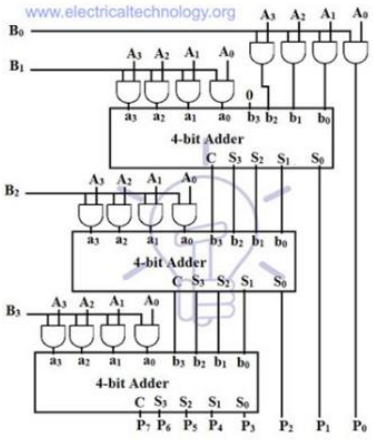


Figura 3-13. Multiplicador de 4 bits a partir de sumadores en cascada. Fuente: [41].

Adicionalmente a la estructura mostrada en la figura anterior, también se implementó lógica adicional para que el multiplicador siempre opere con números positivos; al final, si el resultado de la multiplicación debería ser negativo, es decir, si sólo uno de los dos números es negativo, el producto originalmente positivo se convierte a complemento a 2 para tener el valor correcto a la salida. Esta lógica adicional fue implementada debido a que la multiplicación tiene que ser operada de forma distinta si alguno de los números es negativo, aun estando representado en C2; por ende, la solución más sencilla que se halló fue manejar los signos de forma externa. La Figura 3-14 muestra entonces el diagrama de bloques de cómo está conformado el multiplicador completo de 32 bits. (Nota: este diseño posteriormente es modificado en la sección 3.7.7)

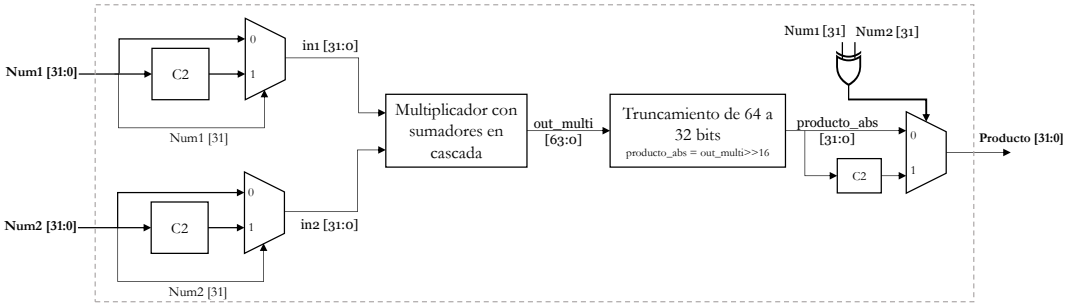


Figura 3-14. Implementación del multiplicador de 32 bits en Q16.

3.5.1.2 Módulos para funciones trigonométricas seno y coseno

Las funciones trigonométricas son ciertamente difíciles de implementar a nivel de hardware por la complejidad en cuanto a sus equivalentes en operaciones matemáticas elementales. Es posible implementarlas a manera de «algoritmo», como el algoritmo CORDIC que realiza la rotación de un vector unitario iterativamente hasta dar con el ángulo deseado, resultando en que las componentes del vector rotado son los valores de seno y coseno del ángulo. Dependiendo de lo exactos que se deseen los resultados son los ciclos de reloj necesarios para el cálculo, pero se recomiendan al menos 20 iteraciones [42].

La otra opción es con tabla de búsqueda, es decir, que se alojen dentro de la memoria de la FPGA los valores del seno y coseno. Esto tendría un tiempo de respuesta reducido debido a la eliminación de todo cálculo. Por las propiedades de simetría del seno y el coseno, se puede almacenar únicamente ¼ de longitud de onda (de 0 rad a $\pi/2$ rad). Distribuyendo esos 90° en 512 muestras se obtiene una resolución de una muestra por cada 0.1757 grados o cada 0.003067 radianes. La simetría de las señales seno y coseno nos permiten utilizar este único cuarto de señal para tener los resultados de las señales completas, mediante las relaciones matemáticas que se muestran en la Figura 3-15.

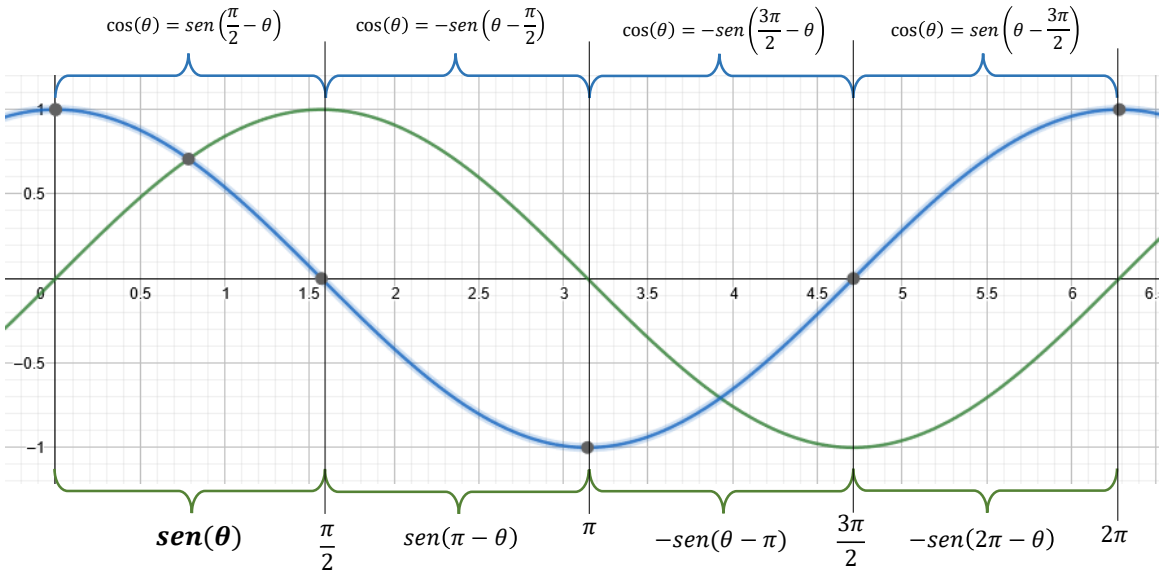


Figura 3-15. Extrapolaciones de las formas de onda seno y coseno.

Para implementar el sistema completo que realiza la obtención del seno y el coseno a partir de un ángulo dado en radianes, se diseñaron los bloques mostrados en la Figura 3-16. De ahora en adelante, a la unidad trigonométrica se le representará mediante un bloque con símbolo «∀».

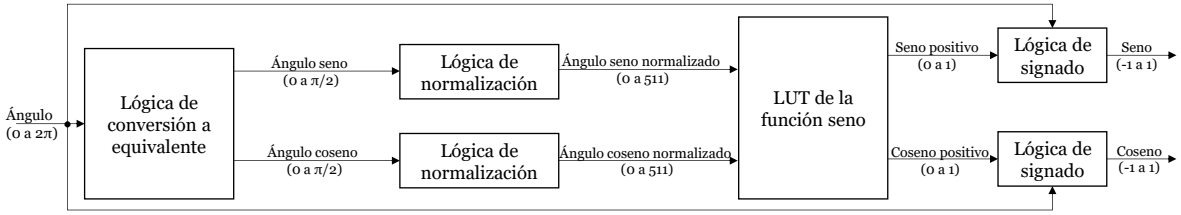


Figura 3-16. Diagrama de bloques de las funciones trigonométricas seno y coseno.

3.5.1.3 Módulo para obtención de raíz cuadrada

Para esta operación también se han encontrado distintas alternativas de diseño. Después de una investigación y estudiar diversas opciones, se decidió elegir el método «non-restoring modified», que fue obtenido del artículo [43] y consiste en los siguientes pasos. Sea A el número del que desea obtener raíz cuadrada:

1. Dividir A en pares de bits a partir del punto decimal
2. Restar "01" a los 2 bits más significativos de A.
3. Si la resta es positiva (mayor o igual a 0), el bit de la raíz correspondiente a ese par de bits es 1, si es negativa, 0.
4. Si la resta es positiva, los siguientes dos bits de A se bajan al resultado de la resta, y a este se le resta la raíz actual con un "01" a la derecha y volver al paso 3.
5. Si la resta va a ser negativa, en vez de restar se retoma el resultado de la resta anterior y se le añaden los siguientes dos bits de A. También se le resta la raíz actual con un "01" añadido a la derecha y volver al paso 3.
6. Seguir hasta terminar con todos los pares de bits.

A continuación se calculará la raíz de 45.59375 (101101.100110_b), cuyo valor es 6.7523 (110.110000001_b), a manera de ejemplo para ilustrar el funcionamiento del método de interés.

$q_5\ q_4\ q_3\ \ q_2\ q_1\ q_0$

$1\ \ 1\ \ 0\ .\ 1\ \ 1\ \ 0$ 6.75 aprox. 6.75231

$\sqrt{10\ 11\ 01\ .\ 10\ 01\ 10}$ 45.59375

-01

$01\ 11$: $resta\ pos. \rightarrow q_5 = 1$

$-01\ 01$: 0q5 01

$00\ 10\ 01$: $resta\ pos. \rightarrow q_4 = 1$

$-00\ 11\ 01$: 00 q5 q4 01

$00\ 10\ 01\ 10$: $resta\ neg. \rightarrow q_3 = 0$

$-00\ 01\ 10\ 01$: 00 0q5 q4q3 01

$00\ 00\ 11\ 01\ 01$: $resta\ pos. \rightarrow q_2 = 1$

$-00\ 00\ 11\ 01\ 01$: 00 00 q5q4 q3q2 01

$00\ 00\ 00\ 00\ 00\ 10$: $resta\ pos. \rightarrow q_1 = 1$

$-\ 00\ 00\ 01\ 10\ 11\ 01$: 00 00 0q5 q4q3 q2q1 01

: $resta\ neg. \rightarrow q_0 = 0$

Aunque en un principio pareciese que la raíz cuadrada se obtiene mediante un proceso secuencial, resulta posible implementar este algoritmo de manera puramente combinacional. Para tal propósito se ha diseñado la siguiente estructura lógica: por cada resta se cuenta con un comparador que compara el minuendo con el sustraendo, si el minuendo es mayor o igual al sustraendo, el valor de la raíz en ese bit será 1, además de que el minuendo del siguiente restador será el resultado de la resta actual con los dos siguientes bits del radical concatenados a la derecha; en caso contrario, la raíz será 0 y el minuendo de la siguiente resta será el minuendo de la resta actual con los dos siguientes bits del radical concatenados a la derecha; para ambos casos, si ya no quedan bits del radicando, se concatenarán «00» a la derecha. El sustraendo de la siguiente resta será la raíz acumulada, con un «01» añadido a la derecha y los ceros necesarios a la izquierda para igualar el tamaño del minuendo.

Debido a que los números en este sistema están representados en 32 bits Q16, la raíz requeriría únicamente 8 bits para la parte entera y 8 para la fraccionaria; sin embargo, se pueden mantener los 16 bits de la fraccionaria para una mayor exactitud, resultando en un total de 24 bits. Para cumplir tal propósito son necesarios 23 restadores y 24 comparadores. La Figura 3-17 ilustra el diagrama de bloques de la lógica implementada para el cálculo de raíz cuadrada. A este bloque se le representará de ahora en adelante con el símbolo «√».

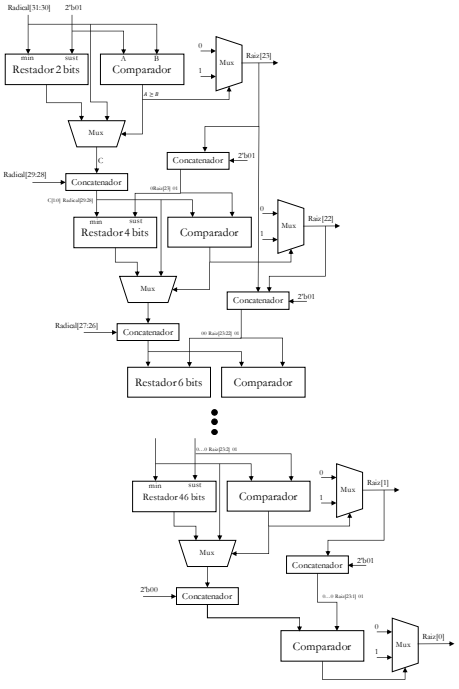


Figura 3-17. Diagrama de bloques del módulo para raíz cuadrada.

Posteriormente a la descripción de este bloque, se realizó el «testbench»[6] de 500 números reales del 33.3333 hasta el 16666.666 que fueron ingresados al módulo de raíz cuadrada. Sus valores de raíz obtenidos por el bloque diseñado se exportaron a un archivo de texto, para posteriormente compararlos con las raíces calculadas con el formato de punto flotante estándar de Matlab. Los resultados de error obtenidos se pueden observar en la Tabla 3-2.

Tabla 3-2. Errores obtenidos en el módulo de raíz cuadrada.

Error promedio e_p	Error máximo $e_{máx}$	Error cuadrático medio e_c
0.000007636	0.000015101	0.000008719

La tabla anterior muestra que los errores encontrados entre las raíces cuadradas de Matlab y las obtenidas por el módulo diseñado rondan las centenas de milésimas, y el error más grande apenas llega a ser un poco más de una decena y media de milésima; por lo tanto, también se comprueba que el bloque implementado funciona correctamente en relación a los resultados esperados, y con este bloque terminado se completan los módulos aritméticos necesarios para implementar la etapa de configuración.

3.5.1.4 Interconexión de módulos

Para completar el bloque aritmético de configuración se deben unir todos los módulos diseñados y así conseguir las operaciones necesarias y los resultados de interés. Los módulos fueron interconectados como lo muestra la Figura 3-18. Las entradas del sistema son los parámetros del arreglo, así como el número de columna y número de fila, pues con estas se realiza el barrido y a las salidas se entregan los valores de coordenadas, módulo, y distancia al alimentador del elemento en cuestión.

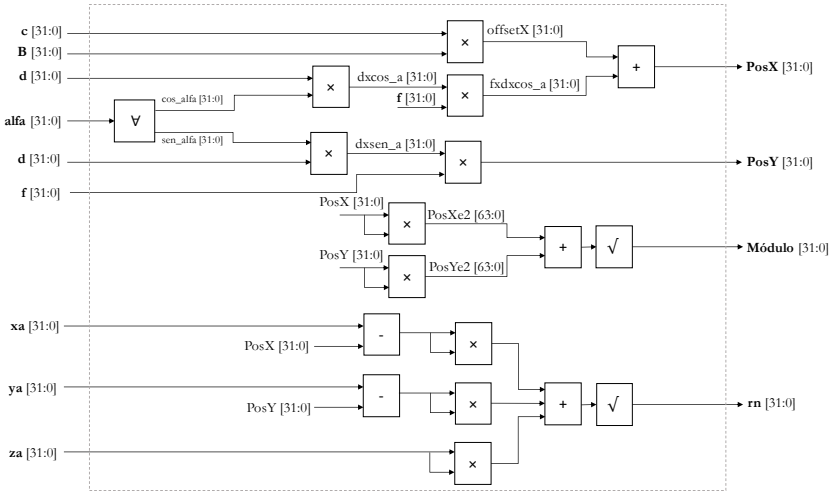


Figura 3-18. Estructura interna de la unidad aritmética de configuración.

Finalmente, con el fin de comprobar el error acumulado que hay en los resultados finales de posiciones y distancias calculadas mediante el diseño y las obtenidas por software, se colocaron los parámetros de un arreglo de ejemplo, de los cuales se obtuvieron los datos de los elementos mediante otro «testbench»; también se exportaron los valores a archivos de texto para compararse con los obtenidos por Matlab. Los parámetros establecidos para esta prueba y los resultados obtenidos se muestran a continuación. R = 100, B = 10, d = 10, α = 55°, xa = 50.465, ya = 33.3333, za = 110.8 [mm].

Tabla 3-3. Errores obtenidos en los cálculos de configuración de un arreglo de ejemplo.

Parámetro	Error promedio e_p	Error máximo $e_{máx}$	Error cuadrático medio e_c
Posición en X	0.035631	0.082571	0.04209
Posición en Y	0.026034	0.06033	0.030753

[6] En un «testbench» es posible corroborar el funcionamiento de uno o más archivos HDL. Mediante una simulación del comportamiento de las señales de entrada, estas son inyectadas de forma virtual hacia el sistema descrito, siendo posible durante la simulación monitorear el comportamiento de las señales internas y de salida.

Módulo	0.020302	0.079856	0.033731
Distancia al alimentador	0.010856	0.064127	0.020238

3.5.2 Memoria de almacenamiento de datos

En la Figura 3-19 se puede observar la etapa de almacenamiento de datos resaltada sobre la estructura de la FPGA. De acuerdo con el algoritmo de configuración presente en la sección 3.5, mediante en análisis de los ciclos «for» es posible ver el orden en que se realiza el barrido a través de las filas y columnas. Este sigue una secuencia de abajo hacia arriba y de izquierda a derecha, como lo ilustra la Figura 3-20. De esta ilustración también hay que notar en qué orden son «encontrados» los elementos del arreglo, pues en la enumeración de aquella figura se sigue el orden del algoritmo.

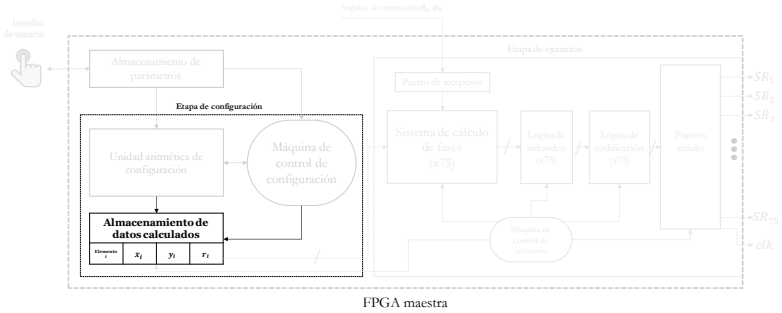


Figura 3-19. Memoria de almacenamiento de datos resaltada sobre la estructura interna.

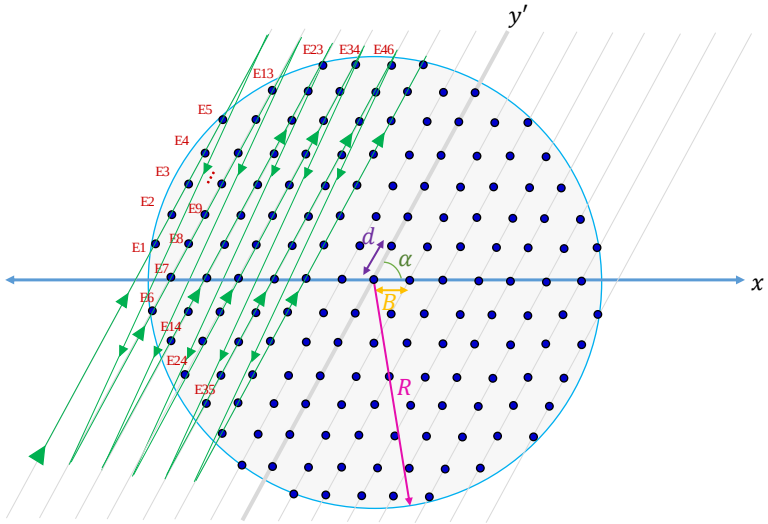


Figura 3-20. Secuencia de barrido del algoritmo.

A continuación se realizarán algunos cálculos necesarios para obtener el tamaño de memoria requerido, según los datos que se necesitan almacenar con base en el número de elementos establecido en los requerimientos (secc. 3.1), y el número de bits reservado para cada dato.

Datos requeridos por elemento: $D_i = x_i + y_i + r_i = 32\ b + 32\ b + 32\ b = 96\ bits$

Número de elementos máximo: $e_a = 1500$

Tamaño de memoria: $B_m = e_a D_i = (1500)(96\ b) = 144\ kb = 18\ kB$

Con respecto al ancho de palabra, es decir, el tamaño de cada registro de memoria, este estará en función de la cantidad de dispositivos esclavos (n_{es}) y la cantidad de elementos asignada a cada esclavo (e_{es}). Se propone un e_{es} de 20 elementos por unidad, lo que requiere un total de 75 dispositivos en tal caso (e_a/e_{es}). Como se busca controlar a los 75 esclavos de

forma paralela, y la operación matemática (2.b) realizada en la etapa de ejecución requiere los 3 datos asociados a cada elemento, se necesita entonces que el ancho de palabra sea:

Ancho de localidad de memoria: $W_m = n_{es} D_i = (75)(96\ b) = 7200\ b$

Finalmente, la cantidad de localidades de memoria será directamente igual a la capacidad de control de los esclavos ($len_m = e_{es}$). Por otro lado, la longitud de la localidad de memoria que puede ser creada por la herramienta Xilinx Block Memory Generator [44] permite máximo 4096 bits, y el valor de Wm resultó ser de 7200 bits.

Si se divide el total de Wm en dos localidades, cada localidad podría tener los datos de 38 elementos ($n_{es}/2$), necesitando una longitud de palabra de 3648 bits ($\frac{n_{es}}{2} D_i$). Claro está que como se utilizan los datos de 75 elementos a la vez, una localidad contendría los datos de los primeros 38 elementos, y la siguiente contendría los de los siguientes 37, esto para mantener una sincronía con el número de dispositivos esclavos (n_{es}) que se manejarán en paralelo.

A primera vista, esta nueva distribución provocaría que el acceso a los datos necesitara el doble de tiempo, por la necesidad de tener que obtener la información de dos registros de memoria en vez de uno. No obstante, ya dentro de esta etapa se puede duplicar el reloj de la memoria, para que de tal manera la obtención de los datos se haga de forma más rápida, consiguiendo así compensar el tiempo de respuesta. Las memorias creadas por la herramienta pueden trabajar hasta 450 MHz [44].

La otra posible opción es que, en vez de duplicar su tamaño, se creen dos memorias que se encuentren trabajando en paralelo, eliminando el problema de la división en dos localidades. Por tal razón la segunda opción resulta más viable. Las memorias tendrán las características siguientes:

Memoria 1: $W_{m1} = \frac{n_{es}}{2} D_i = (38)(96\ b) = 3648\ b$ Memoria 2: $W_{m2} = \frac{n_{es}}{2} D_i = (37)(96\ b) = 3552\ b$

Si la operación $\frac{n_{es}}{2}$ resulta en un número no entero, como es el caso, una memoria tiene que ser ligeramente mayor que la otra. Ambas comparten un único bus de direcciones y de reloj puesto que se estarán manipulando al mismo tiempo en todo momento. Recuérdese que la cantidad de localidades de memoria sería igual a la cantidad de elementos que controle cada esclavo ($len_m = e_{es}$).

Resulta importante recordar que no solo la etapa de configuración controlará la memoria, sino que la interfaz de usuario y la etapa de ejecución también deben tener control sobre la lectura de datos. Para tal propósito, se diseñó un bloque llamado «Mux memorias», que permite ceder el control de la memoria a cada máquina según las señales **FFCfg**, **reconfig** y **flagsendmem** (explicadas más adelante). Hechas las estimaciones anteriores, la estructura de las memorias de elementos y de la lógica de lectura se muestran en la Figura 3-21; los datos de las dos memorias se pueden ver como una localidad conjunta.

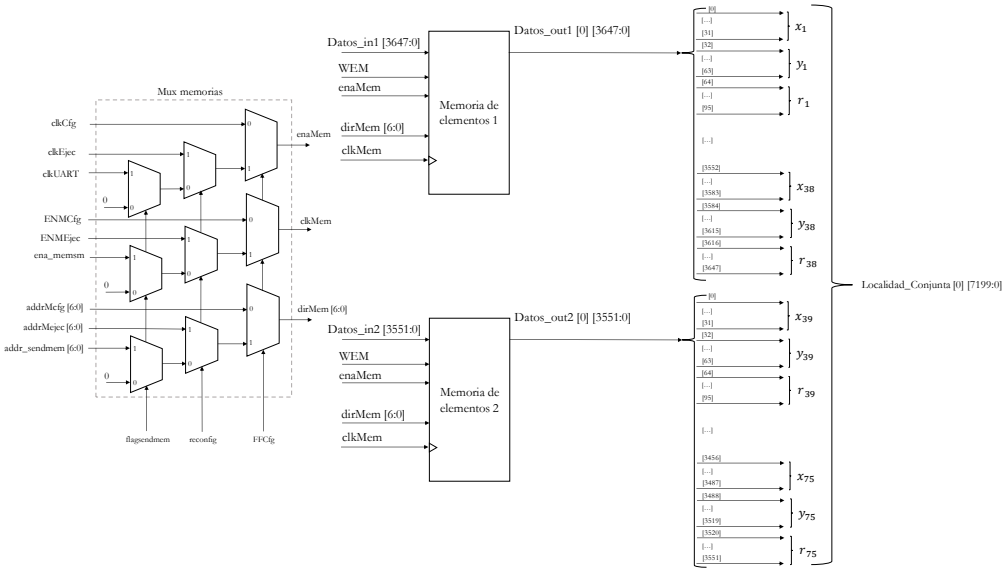


Figura 3-21. Estructura de datos en las memorias de elementos.

La estructura de las memorias propuesta en la Figura 3-21 también sirve para identificar cuáles son los elementos que cada dispositivo esclavo estará controlando. Para entender mejor la asignación está la Tabla 3-4. Es posible notar cómo una localidad contiene los datos de n_{es} elementos consecutivos (siguiendo el orden mostrado en la Figura 3-20). Cada uno de los datos que se ubican en una misma localidad irá posteriormente hacia un bloque de ejecución distinto, siendo que cada bloque de ejecución está encargado de un esclavo.

Tabla 3-4. Almacenamiento de datos en memorias.

Localidad conjunta 1	Localidad conjunta 2	Localidad conjunta 3	…	Localidad conjunta e_{es}
Datos elem. 1	Datos elem. $n_{es}+1$	Datos elem. $2n_{es}+1$	…	Datos elem. $(e_{es}-1)n_{es}+1$
Datos elem. 2	Datos elem. $n_{es}+2$	Datos elem. $2n_{es}+2$	…	Datos elem. $(e_{es}-1)n_{es}+2$
Datos elem. 3	Datos elem. $n_{es}+3$	Datos elem. $2n_{es}+3$	…	Datos elem. $(e_{es}-1)n_{es}+3$
…	…	…	…	…
Datos elem. n_{es}	Datos elem. $2n_{es}$	Datos elem. $3n_{es}$	…	Datos elem. $(e_{es})n_{es}$

En otras palabras, los esclavos controlarán a elementos no consecutivos debido a la estructura de la memoria. Para terminar de ejemplificar, si se tiene un número de esclavos $n_{es} = 75$ y un arreglo de 1500 elementos, entonces la asignación de elementos a cada esclavo es la presente en la Tabla 3-5. El mismo análisis se puede realizar para cualquier arreglo, solo basta conocer el número de esclavos y el número de elementos.

Tabla 3-5. Asignación de elementos a esclavos para un arreglo de 1500 elementos y 75 esclavos.

Esclavo	Localidad conjunta 1	Localidad conjunta 2	Localidad conjunta 3	…	Localidad conjunta 20
1	Datos elem. 1	Datos elem. 76	Datos elem. 151	…	Datos elem. 1426
2	Datos elem. 2	Datos elem. 77	Datos elem. 152	…	Datos elem. 1427
3	Datos elem. 3	Datos elem. 78	Datos elem. 153	…	Datos elem. 1428
4	Datos elem. 4	Datos elem. 79	Datos elem. 154	…	Datos elem. 1429
…	…	…	…	…	…
74	Datos elem. 74	Datos elem. 149	Datos elem. 224	…	Datos elem. 1499
75	Datos elem. 75	Datos elem. 150	Datos elem. 225	…	Datos elem. 1500

Ahora bien, se hablará sobre la manera en que se ingresarán los datos de los elementos hacia las memorias. Se plantea el uso de registros temporales, cuyas salidas concatenadas se convertirán en una «localidad conjunta» para los datos de entrada de las memorias. Por cada elemento del arreglo se tendrán 96 bits que contienen sus datos, y estos serían ingresados a un registro de 96 bits, de los que en total se tendrían 75 para cubrir el máximo de dispositivos esclavos. Obsérvese la Figura 3-22.

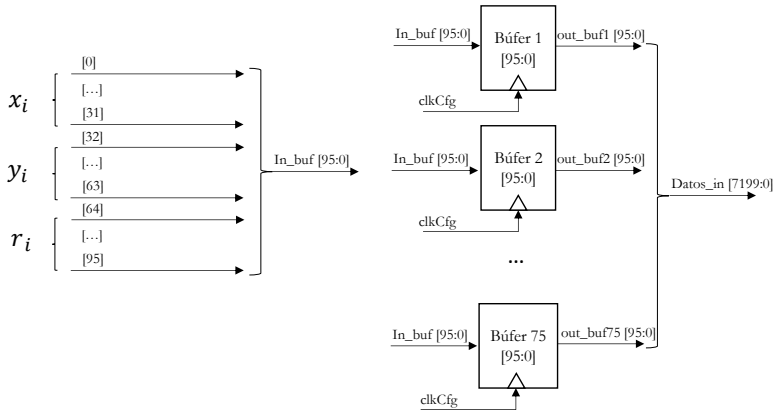


Figura 3-22. Distribución de señales en los búfers temporales para el llenado de memoria.

Todos los búfers comparten un único bus de datos llamdo **In_buf [95:0]**, la elección de cuál búfer es el que se carga en determinado momento será dado por la máquina de control de configuración, a través de un demultiplexor, un respectivo selector nombrado **sel_DMBuf**, y el control de sus señales de reloj. Para ilustrar esta lógica se muestra la Figura 3-23.

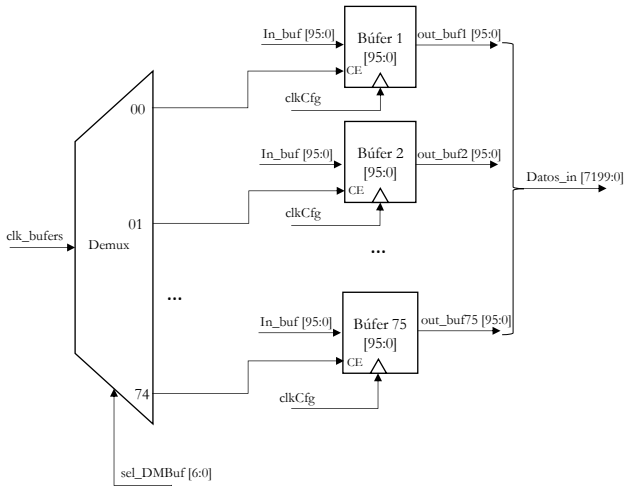


Figura 3-23. Control de carga de datos hacia los búfers mediante demultiplexor y activador de reloj.

3.5.3 Máquina de control de configuración

La máquina de control de configuración se encuentra resaltada en la Figura 3-24. Hasta el momento se han explicado los bloques lógicos que formarán parte del algoritmo de configuración, pero aún sigue pendiente el sistema secuencial que se encargará de coordinar y sincronizar todos los módulos para la tarea principal de esta etapa: almacenar los datos referentes a los elementos del arreglo dentro de las «memorias de elementos». Para tal propósito, en el presente apartado se hablará y se diseñará esta máquina, que llevará por nombre «máquina de control de configuración», misma que tiene como propósito «ejecutar» el algoritmo de la sección 3.5. Para comenzar, se describen el par de tareas principales que debe realizar.

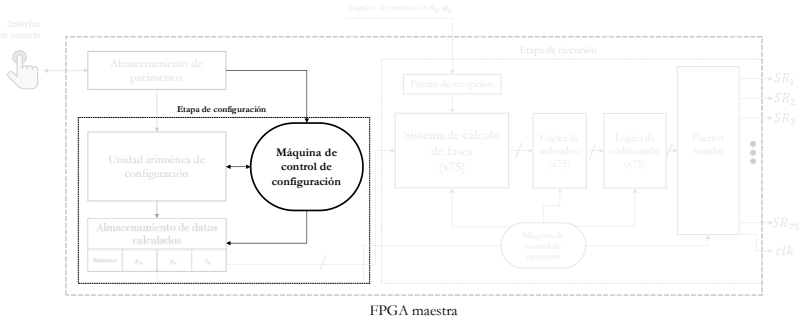


Figura 3-24. Máquina de control de configuración resaltada sobre la estructura interna.

1. Realizar el barrido de filas y columnas del algoritmo: teniendo un par de contadores secuenciales cuyas salidas sean los índices de fila y de columna, la máquina deberá controlar el incremento de estos índices para cumplir con el algormito de configuración.
2. Controlar la lógica del llenado de memoria: la máquina deberá detectar cuando se haya encontrado un elemento perteneciente al arreglo, almacenar sus datos en los registros temporales y repetir hasta completar los datos para el total de bits correspondientes a una localidad conjunta de las dos memorias, después ingresar los valores hacia las memorias, y repetir el proceso las veces que sean necesarias hasta completar el barrido secuencial.

3.5.3.1 Contadores para índices de fila y de columna

Tal como se observó en la Figura 3-20, el barrido sigue una dirección de abajo hacia arriba y de izquierda a derecha, o bien, de la fila negativa progresivamente hacia la positiva y de la columna negativa hacia la positiva.

Para realizar el barrido se necesitan un par de máquinas secuenciales, una que realice el incremento del índice de filas «**F**», y otra que incremente el índice de columnas «**C**». Ambos índices comienzan en un valor inicial (VIF y VIC), y se incrementan unitariamente hasta llegar a los valores finales (VFF y VFC). Cuando se llega al valor final de fila (VFF), se incrementa el valor de columna y se reinicia el valor de fila con su respecitvo VIF, siguiendo el mismo ciclo sucesivamente hasta llegar a los valores finales de columna (VFC) y de fila (VFF).

Siguiendo la secuencia anterior, un diagrama de bloques de ambas máquinas se muestra en la Figura 3-25, en la cual se puede ver que el incremento de la máquina de columnas es controlado por la máquina de filas.

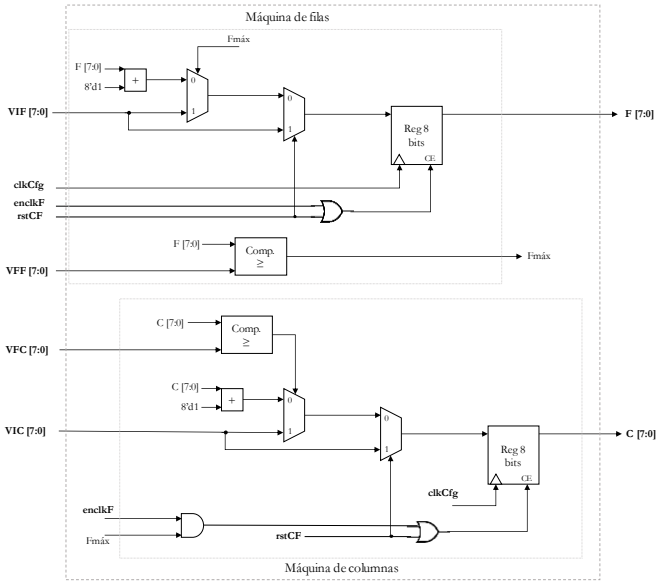


Figura 3-25. Diagrama de bloques de la máquina de barrido de filas y columnas.

3.5.3.2 Máquina secuencial de configuración

Habiendo establecido la estructura que tendrá la lógica para el llenado de las memorias de elementos y las máquinas de barrido, lo consiguiente es diseñar la máquina secuencial de control. Para esto se ha optado por ilustrar con un diagrama de estados simplificado. Se tienen varias señales nuevas que son salidas o entradas a la máquina, mismas que son nombradas y descritas en la Tabla 3-6 posterior al diagrama de estados de la Figura 3-26. En aquella tabla se explican los estados y las transiciones de la máquina. Las transiciones del diagrama que no tienen ninguna señal asociada son incondicionales; las que las tienen alguna(s), son estas las únicas que afectan a cada estado en concreto. Finalmente, un diagrama de bloques con la estructura de la máquina de control se encuentra en la Figura 3-27.

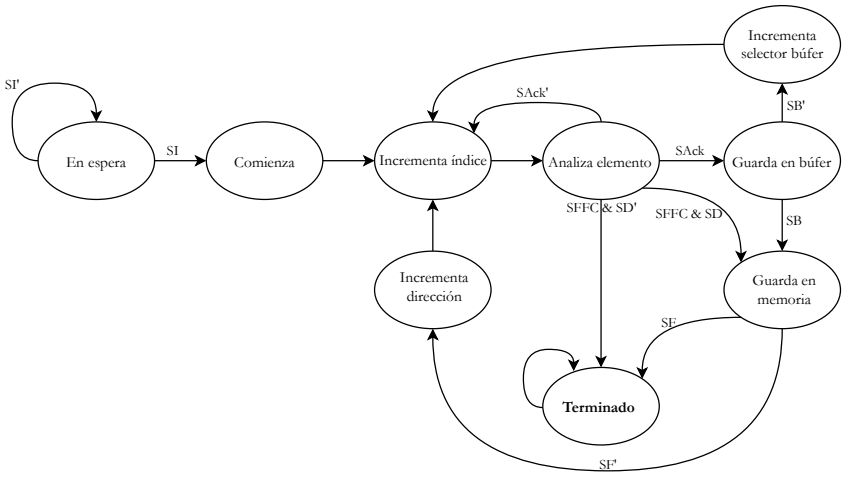


Figura 3-26. Diagrama de estados de la máquina de control de configuración.

Tabla 3-6. Descripción de señales y estados de la máquina de control de configuración.

Señales		
Tipo	**Nombre**	**Descripción**
Entrada	**SI**: señal de inicio	Es una señal lógica cuyo valor positivo sirve para dar inicio a la máquina maestra. **SI = (StartCfg & flagParam)** **StartCfg** es una señal que en el diseño final provenirá de un botón físico. **flagParam** es la señal que indica que ya se tienen almacenados los parámetros del arreglo.
	SAck: señal de reconocimiento	Esta señal se activa cuando se ha detectado que el elemento en cuestión pertenece al arreglo. Es la salida de un bloque que compara el módulo del elemento para saber si es menor o igual al radio del arreglo. **SAck = (modulo ≤ R)**
	SB: señal de búfers	Se activa cuando se han llenado todos los búfers temporales, esto es cuando el **sel_DMBuf** ha alcanzado el valor del número de dispositivos esclavos n_{es}. **SB = (sel_DMBuf ≥ n_{es})**
	SD: señal de datos pendientes	Se coloca en uno lógico siempre que el selector **sel_DMBuf** es mayor a cero, es decir, cuando al menos un búfer tiene guardado algún dato en el momento. **SD = (sel_DMBuf > 0)**
	SFFC: señal de finalización de filas y columnas	Se activa cuando los índices de fila y columna han llegado a sus valores de VFF y VFC. **SFFC = (F ≥ VFF) & (C ≥ VFC)**
	SADDR: señal de fin de contador de dirección	Se activa cuando el contador de dirección de memoria **addrMCfg** ha alcanzado el número máximo de localidades de las memorias **lenM**. **SADDR = (addrMCfg ≥ lenM – 1)**
	SF: señal de finalización	Esta señal se activa cuando se han terminado de llenar las memorias, ya sea porque el bus de direcciones alcanza el valor máximo, o bien porque la señal **SFFC** se ha puesto en 1. **SF = SADDR | SFFC**
Salida	**rstFC**: reinicio de índices	Señal que carga los registros de índices **F** y **C** con sus valores iniciales **VIF** y **VIC**.
	enclkF: activador reloj filas	Señal que activa el reloj de la máquina de filas.
	enclkBF: reloj búfers	Activador de reloj para los registros búfers temporales.
	rstBuf: señal reinicio de búfers	Señal para limpiar los búfers temporales.
	inc_DMBuf: reloj selector multiplexor	Activador de señal de reloj para el contador llamado **sel_DMBuf**.
	inc_addrM: reloj dirección	Activador de reloj para el contador de direcciones de las memorias de elementos.
	FFCfg: bandera de finalización de configuración	Señal tipo bandera que se activa cuando se ha llegado al final del algoritmo de configuración.
	WEM: habilitación de escritura de memorias	Señal que habilita la escritura a la memoria de elementos.
	ENMCfg: habilitación de memorias	Señal que habilita las memorias de elementos.
Estados		
Nombre (abreviatura)	**Descripción y transiciones**	**Valor de señales de salida (las señales no explícitas en cada estado tienen valor cero).**
En espera (idle)	El estado en que inicia la máquina de estados, esperando a la señal SI para pasar al estado de «start».	**rstFC = 1** **rstBuf = 1**
Comienza (start)	Estado transitorio que permite analizar el elemento en los índices iniciales. Transiciones: **SAck = 1**, se ha detectado un elemento los índices de columna y fila iniciales, se procede a saltar a «saveBF». **SAck = 0**, no se ha detectado ningún elemento, se pasa a «incF».	Todas las señales de salida son cero.

Incrementa índice (incF)	En este estado se activa la señal de reloj de la máquina de filas para incrementar su valor. Transición incondicional hacia «anaElem».	**enclkF = 1**
Análisis elemento actual (anaElem)	Analiza el elemento con los índices actuales. Transiciones: **SAck**, se encontró un elemento perteneciente al arreglo, salta hacia «saveBF». **SAck'**, no se encontró ningún elemento, se regresa a «incF». **SFFC & SD**, finalizó el barrido y quedan algunos datos en los búfers, por lo que se pasa a «saveMem». **SFFC & SD'**, finalizó el barrido y los búfers están vacíos, por lo que se finaliza el algoritmo en «finished».	Todas las señales de salida son cero.
Guarda en búfer (saveBF)	En este estado se envía el activador de reloj hacia los búfers temporales, señal que llegará hacia el búfer indicado por el selector, para guardar los datos del elemento encontrado. Transiciones: **SB**, los búfers están llenos, por lo que se pasa a «saveMem». **SB'**, aún quedan búfers vacíos, se pasa a «incSel».	**enclkBF = 1**
Incrementa selector de búfer (incSel)	En este estado se activa la señal de reloj para la máquina del **sel_DMBuf**, con el objetivo de incrementar su valor. Transición incondicional hacia «incF».	**inc_DMBuf = 1**
Guarda en memorias (saveMem)	Se proporciona la activación de la memoria para escribir los datos de los búfers hacia las memorias de elementos. Transiciones: **SF = 1**, significa que se han llenado las memorias o se ha terminado el barrido, por lo que se salta al estado final «finished». **SF = 0**, significa que el algoritmo aún debe continuar, se salta hacia «incDir».	**WEN = 1** **ENM = 1**
Incrementa dirección memorias (incDir)	El estado sirve para activar incrementar el contador de dirección de memorias, limpiar los búfers y el selector de búfer. Transición incondicional hacia «incF».	**inc_addrM = 1** **rstBuf = 1**
Terminado (finished)	Es el estado final al que se desea llegar, donde se han terminado de almacenar los datos de todos los elementos pertenecientes al arreglo Transición incondicional hacia sí mismo. La bandera de finalización de configuración se activa únicamente en este estado.	**FFCfg = 1**

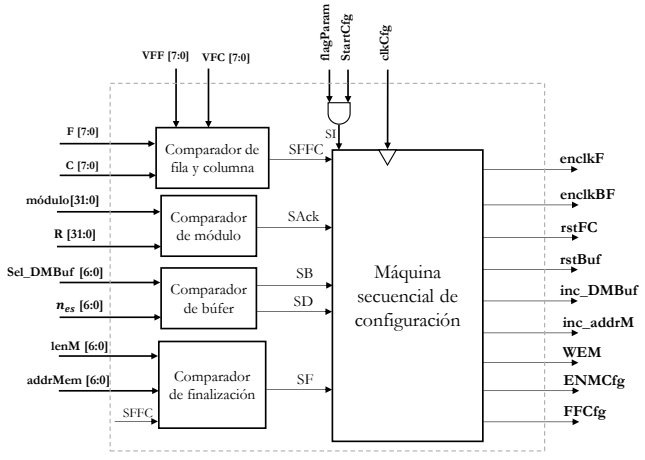


Figura 3-27. Estructura interna de la máquina de control de configuración.

3.5.4 Interconexión de módulos

Habiendo completado el diseño de las máquinas secuenciales y combinacionales, el último paso en el diseño es interconectar todos los módulos, completando así el subsistema correspondiente a la etapa de configuración como está estipulado previamente en la Figura 3-9. El diagrama de bloques de esta etapa se puede consultar en la Figura 3-28. (Nota: la estructura de la etapa de configuración es modificada en la sección 3.7.13)

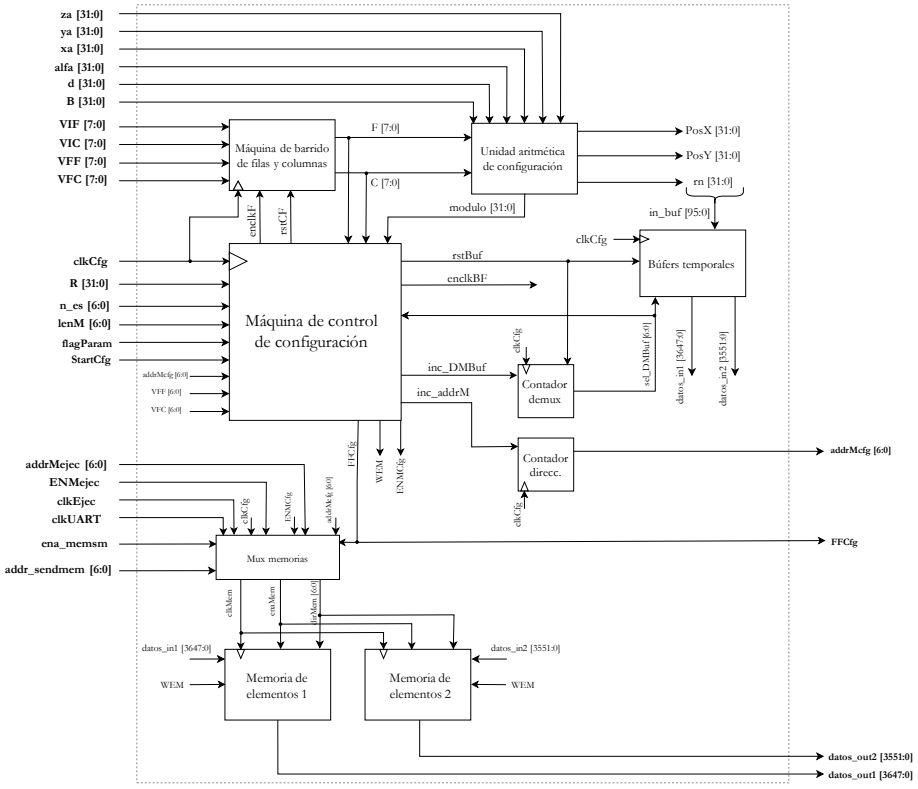


Figura 3-28. Interconexión de módulos para la etapa de configuración.

3.6 DISEÑO DE LA INTERFAZ DE USUARIO Y ALMACENAMIENTO DE PARÁMETROS

En el presente apartado y las subsecuentes secciones se habla sobre el concepto y diseño de la interfaz de usuario (IU), además de la forma en que serán almacenados y procesados los parámetros dentro de la tarjeta maestra. A través de la interfaz se ingresarán los datos y parámetros necesarios para el funcionamiento del sistema y se enviarán hacia la FPGA. Adicionalmente, el usuario podrá verificar los datos recibidos y los calculados por la FPGA, como manera de corroborar la correcta ejecución de la etapa de configuración diseñada en la sección 3.5. En la Figura 3-29 se presentan los dos componentes a diseñar resaltados sobre la arquitectura general.

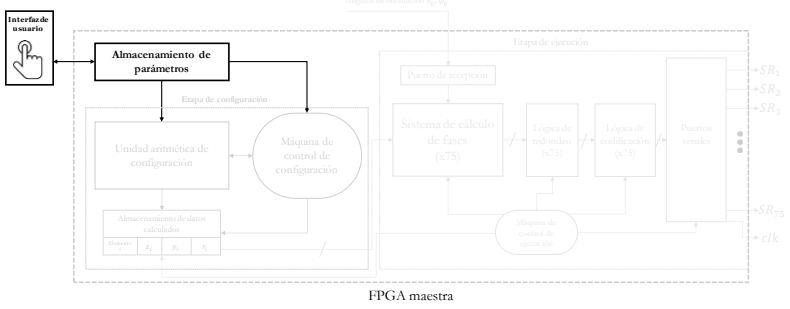


Figura 3-29. Almacenamiento de parámetros e interfaz de usuario resaltados sobre estructura interna.

La idea principal de la IU es contar con un programa en otro sistema, como una computadora personal, en donde el usuario tenga manera de ingresar los parámetros del arreglo, y una vez ingresados se envíen por un protocolo de comunicación hacia la FPGA maestra, quien confirmará la recepción de los mismos y comenzará la respectiva etapa de configuración.

Si hablamos de una computadora de uso general, el programa de carga de datos puede ser implementado en Python o Matlab, diseñando así un software que cuente con una interfaz de usuario sencilla y fácil de manejar, donde se puedan teclear los datos del arreglo, el programa haga el resto de cálculos necesarios y envíe la información hacia la FPGA maestra. También se contará con indicadores visuales para saber cuando la tarjeta ha recibido los datos y cuando ha terminado la etapa de configuración. La Figura 3-30 siguiente ilustra un esquema preliminar de cómo se visualiza la interfaz de usuario.

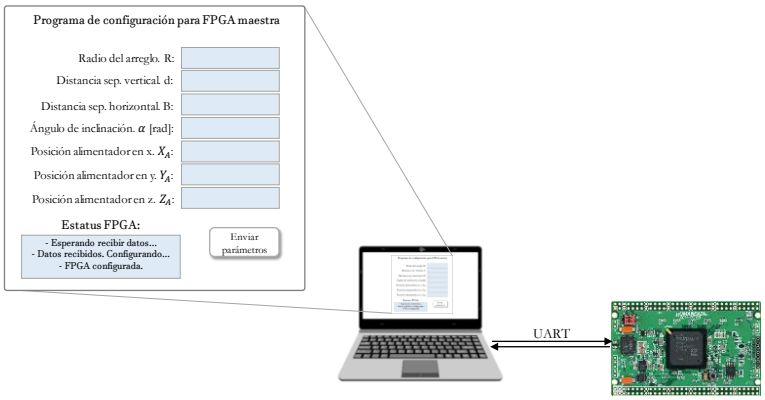


Figura 3-30. Ilustración de la interfaz de usuario.

Se optó por programar la IU en lenguaje Python, utilizando la biblioteca Tkinter [45] para los elementos visuales, mientras que para la comunicación con la tarjeta se utilizó la biblioteca Pyserial [46]. En el listado siguiente se indican y describen las funciones accesibles al usuario.

Envío de parámetros: se envían los parámetros R, d, B, α, Ax, Ay, Az, $1/\lambda$ y b_a (secc. 2.1.4) hacia la tarjeta maestra, tecleados previamente por el usuario.

Solicitud de estatus: se envía un comando a la FPGA para que devuelva el estatus en el que se encuentra. Se cuenta con tres estados:

- Parámetros recibidos: se han recibido los parámetros, pero aún no se finaliza o inicia el proceso de configuración.
- Configuración terminada: se ha terminado el algoritmo de configuración y el llenado de la memoria de elementos.
- Error de comando/datos: no se ha recibido el comando correctamente o se ha recibido un comando no válido.

Verificar parámetros: se solicita a la FPGA que envíe los parámetros recibidos, el programa los recibe y los compara con los previamente enviados. Indica visualmente si estos coincidieron o no, además de desplegar los valores recibidos en la terminal de Python.

Verificación de memoria de elementos: se envía un comando para que la FPGA envíe los datos almacenados en la memoria de elementos. El programa compara estos con los calculados internamente y, bajo un margen de error, verifica que sean iguales. Los datos de la tarjeta también son exportados a un archivo de texto para poder ser utilizados o analizados en algún otro programa.

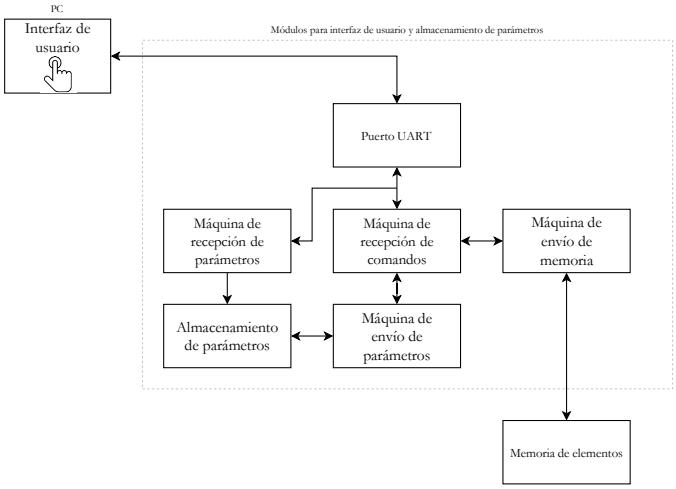


Figura 3-31. Diagrama de bloques de la interfaz de usuario y almacenamiento de parámetros.

Para hacer funcionar todos estos comandos, la tarjeta maestra debe contar con circuitos secuenciales que le permitan almacenar la información recibida, interpretar comandos y realizar las acciones necesarias según cada comando. Asimismo, se debe contar con registros que conserven los valores de los parámetros y constantes del arreglo. Un diagrama de bloques de la interfaz de usuario y almacenamiento de parámetros se encuentra en la Figura 3-31. Las explicaciones de los módulos allí incluidos se encuentran en los siguientes apartados. Se finaliza esta sección con la Figura 3-32, que muestra la ventana de la IU ya programada.

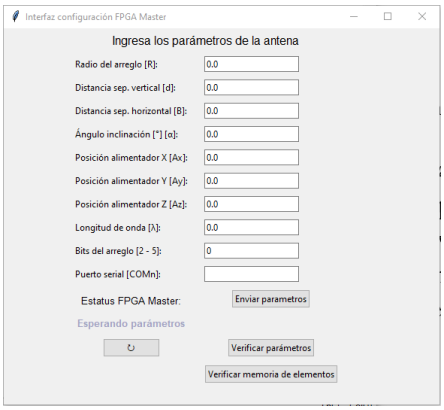


Figura 3-32. Ventana principal de la interfaz de usuario.

3.6.1 Almacenamiento de parámetros

En esta sección se explica el diseño de los registros que almacenan las constantes y parámetros asociados al arreglo. Como se explicó anteriormente, la interfaz de usuario trabaja a través del protocolo UART, lo que implica a su vez que los datos se reciben en conjuntos de 8 bits. Así bien, la mayoría de parámetros están conformados por 32 bits, es decir, 4 conjuntos de 8 bits.

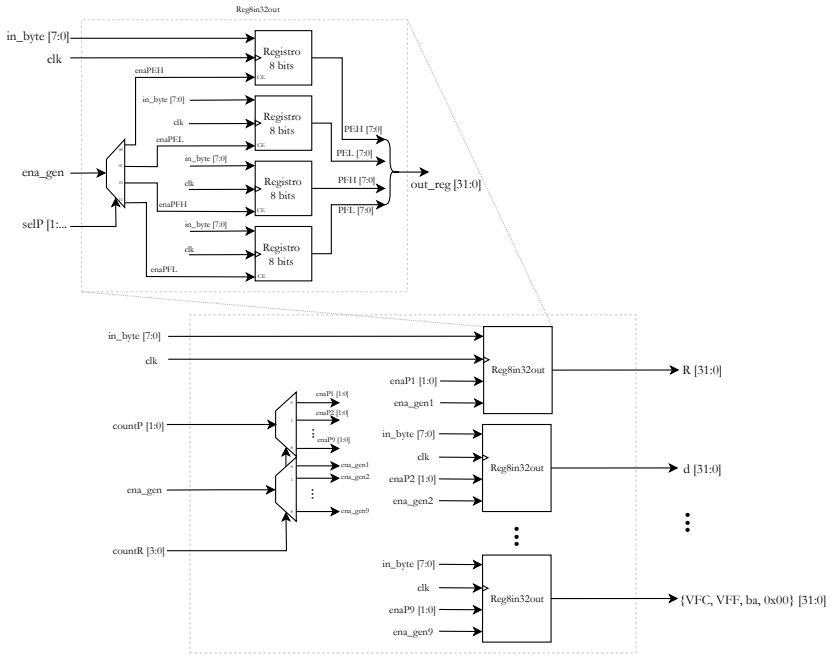


Figura 3-33. Estructura interna del módulo de almacenamiento de parámetros

Los registros de 32 bits deben ser llenados progresivamente, conforme se van recibiendo los datos a través del puerto. Por ende, el *módulo de almacenamiento de parámetros* (ilustrado en la Figura 3-33) cuenta con forma de elegir a qué registro se desea escribir mediante la señal **countR**, y a qué parte (byte) del registro mediante la señal **countP**.

Los 9 registros, tal como fueron asignados, deberán recibirse en un orden específico: cada registro de 32 bits se deberá almacenar en el orden PE_H, PE_L, PF_H, PF_L (PE siendo Parte Entera y PF siendo Parte Fraccionaria; H indica los 8 bits más significativos y L los 8 menos significativos). El orden de los registros deberá ser el siguiente: R, d, B, α, Ax, Ay, Az, $1/\lambda$, y los valores concatenados de {VFC, VFF, b_a, 0x00}[7] (parámetros explicados en secciones 2.1.4 y 2.1.5).

3.6.2 Transceptor UART

Esta máquina secuencial es la encargada del envío y recepción de datos a través del protocolo UART. Su funcionamiento, de forma sencilla, se explica en el siguiente párrafo. Posterior a la explicación se encuentra la Figura 3-34 con los diagramas de bloques de ambas máquinas secuenciales que conforman este puerto.

Para la recepción de datos se utiliza una técnica llamada «sobremuestreo», en la cual una máquina secuencial monitorea la señal de datos a una frecuencia 4 veces mayor que la tasa de transferencia, con el fin de detectar a tiempo el flanco de bajada que indica el inicio de una recepción. Está descrita para trabajar con datos de 8 bits, sin bit de paridad y con 1 bit de detención, a una tasa de 115,200 bauds. A la salida de la máquina se encuentra el byte recibido (**rx_byte[7:0]**), así como una bandera para indicar cuando se está recibiendo un dato (**is_receiving**), cuando se ha terminado de recibir (**received**), y cuando ha habido un error en la recepción (**recv_error**).

Para la transmisión se utiliza otra máquina secuencial que controla la señal de salida **tx**, la mantiene en un nivel alto cuando no se está transmitiendo, y cuando se detecta la señal de inicio de transmisión comienza el envío del byte ingresado **tx_byte[7:0],** bajo los mismos parámetros que la máquina de recepción. La señal de inicio **transmit** sirve para comenzar la transmisión y la bandera **is_transmitting** permanece en nivel alto mientras se esté transmitiendo.

[7] La notación {valor1, valor2, …} indica la concatenación de varios valores para formar un solo bus.

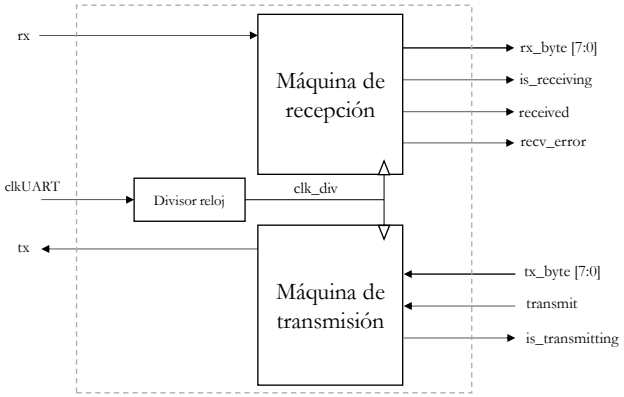


Figura 3-34. Diagrama de bloques del transceptor UART.

3.6.3 Máquina de recepción de parámetros

La FPGA está pensada para recibir los parámetros del arreglo de manera inicial. Por ende, la *máquina de recepción de parámetros* es la primera en activarse en cuanto se detecta la primera recepción por el puerto UART. Se cuenta con 9 registros de 32 bits que corresponden a los parámetros del arreglo, como se vio en la sección 3.6.1.

Debido a que los datos se reciben en paquetes de un byte, los registros de 32 bits se llenan paulatinamente. Para esto se diseñó la máquina secuencial de recepción de parámetros, que se encarga de distribuir los bytes recibidos hacia los registros del módulo de almacenamiento de parámetros. Estos datos están pensados para recibirse y distribuirse en un orden específico, mismo que es mencionado en el apartado 3.6.1.

Por lo tanto, la máquina de recepción de parámetros cuenta con dos contadores: uno que se encarga de activar el registro en cuestión **countR**, y otro que se encarga de indicar a qué parte del registro **countP** es a la que se va a escribir en cada momento. Cuando se ha finalizado la recepción de los parámetros, una señal de salida llamada **flagParam** se activa permanentemente.

El diagrama de estados simplificado de la máquina de recepción de parámetros se adjunta en la Figura 3-35, posteriormente se muestra la Tabla 3-7 con la descripción de los estados y las señales. Finalmente, en la Figura 3-36 está el diagrama de bloques.

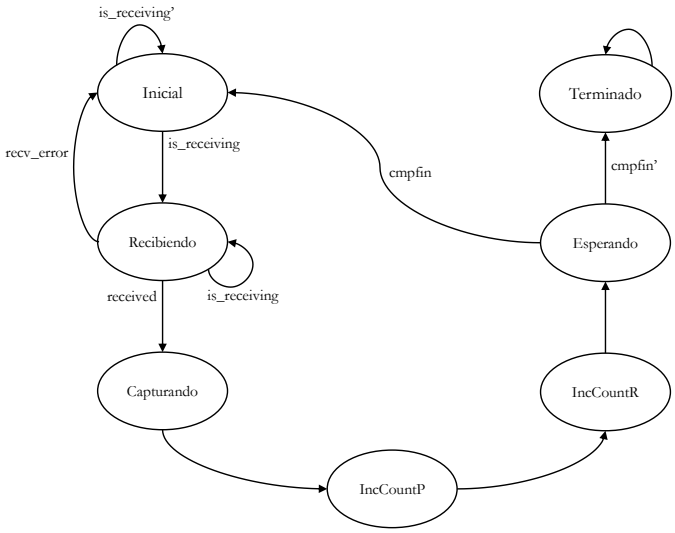


Figura 3-35. Diagrama de estados de la máquina secuencial de recepción de parámetros.

Tabla 3-7. Descripción de señales y estados de la máquina secuencial de recepción de parámetros.

Señales		
Tipo	**Nombre**	**Descripción**
Entrada	**is_receiving**: señal de recepción del puerto UART	Esta señal proviene del transceptor UART, indica que se está recibiendo un dato.
	recv_error: señal de error del puerto UART	Hubo un error en la recepción del dato por el puerto UART.
	received: señal de dato recibido del puerto UART	Se activa cuando se ha terminado de recibir el dato por el puerto UART.
	ZselecP: selector de parte es cero	Se activa cuando el selector de parte **countP** es igual a cero.
Salida	**cmpfin**: selector de registro es valor máximo	Se activa cuando el selector de registro **countR** es mayor o igual a 9.
	ena_gen: activador de registro	Señal que activa el reloj del registro de 8 bits en cuestión.
	clk_selecP: activador de reloj para registro **countP**	Permite el incremento del selector de parte.
	clk_selecR: activador de reloj para registro **countR**	Permite el incremento del selector de registro.
	flagParam: señal de finalización de recep. de parámetros	Se activa permanentemente cuando se llega al estado final de la máquina.
Estados		
Nombre (abreviatura)	**Descripción y transiciones**	**Valor de señales de salida (las señales no explícitas en cada estado tienen valor cero).**
Inicial	El estado de reposo en el que inicia la máquina de estados. Transiciones: **is_receiving = 0**, se mantiene en «inicial». **is_receiving = 1**, se está recibiendo el primer parámetro, por lo que se pasa a «recibiendo».	Todas las señales de salida son cero.
Recibiendo	Estado de espera hasta que se complete la recepción del dato. Transiciones: **recv_error = 1**, no se recibió el dato de forma correcta, por lo que se regresa a «inicial». **recv_error = 0 & received = 0**, aún no se completa la recepción, se mantiene en «recibiendo». **recv_error = 0 & received = 1**, se recibió un dato sin errores, por lo que se pasa a «capturando».	Todas las señales de salida son cero.
Capturando	El estado sirve para activar el reloj del registro en cuestión. Transición incondicional hacia «incCountP».	**ena_gen = 1** **clk_selecP = 0** **clk_selecR = 0** **flagParam = 0**
IncCountP	Activa el reloj del registro «countP» para incrementar su valor. Transición incondicional hacia «incCountR».	**ena_gen = 0** **clk_selecP = 1** **clk_selecR = 0** **flagParam = 0**
IncCountR	Se incrementa el selector de registro siempre y cuando el contador de parte sea cero. Transición incondicional hacia «esperando».	**ena_gen = 0** **clk_selecP = 0** si **ZselecP = 1**, **clk_selecR = 1** si **ZselecP = 0**, **clk_selecR = 0** **flagParam = 0**
Esperando	Se analiza la cantidad de datos recibidos hasta el momento. Transiciones: **cmpfin = 0**, aún no se acaban de llenar los registros, por lo que se pasa a «inicial». **cmpfin = 1**, se han terminado de llenar los registros, se pasa al estado final «terminado».	**ena_gen = 0** **clk_selecP = 0** **clk_selecR =0** **flagParam = 0**
Terminado	Estado final en al que se llega cuando se recibieron todos los parámetros. Transición incondicional hacia «terminado».	**ena_gen = 0** **clk_selecP = 0** **clk_selecR = 0** **flagParam = 1**

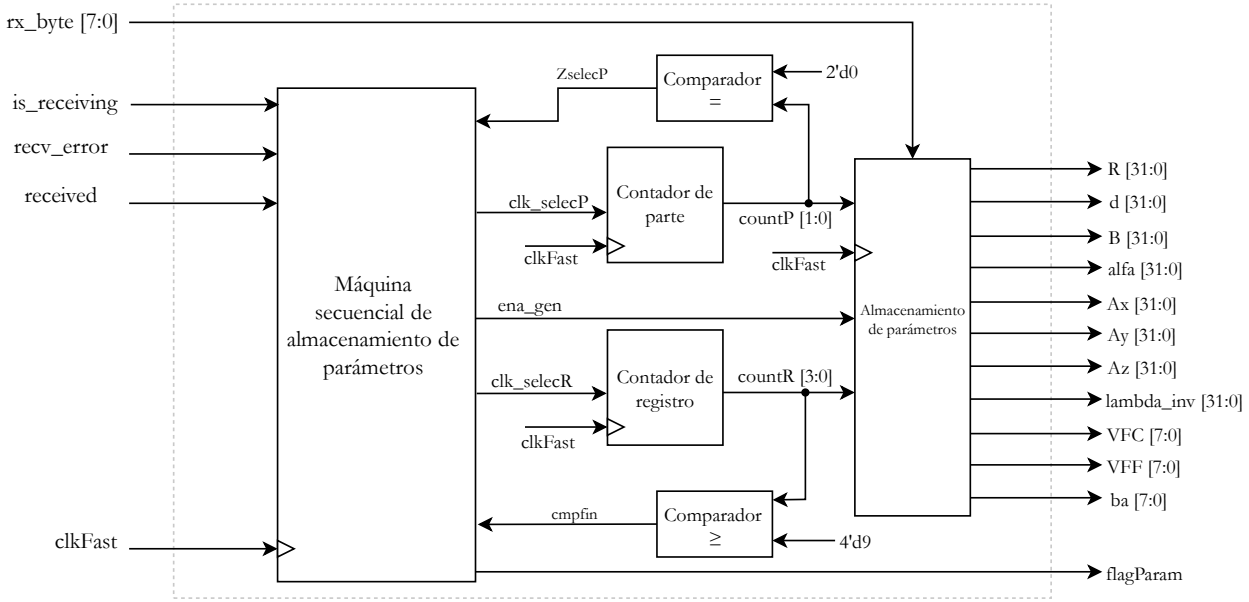


Figura 3-36. Diagrama de bloques de la máquina de recepción de parámetros.

3.6.4 Máquina de recepción de comandos

Esta máquina secuencial (MRC) comienza a funcionar cuando la de recepción de parámetros ha terminado su proceso de llenado. Mediante la señal **flagParam** es como esta máquina decide si ignorar o si tomar el dato recibido por el puerto UART como un comando. El propósito de este circuito secuencial es interpretar los comandos recibidos posteriores a la recepción de parámetros, y detonar las acciones correspondientes según la instrucción recibida.

Se cuenta con tres comandos, el primero es la *solicitud de estatus* (código 0x66). Cuando se detecta este código, la máquina envía un byte de respuesta dependiendo del estado en el que se encuentre, el cual es interpretado y desplegado al usuario por el programa de la interfaz.

El segundo es la *verificación de parámetros* (código 0x44). Cuando se detecta este código, la máquina de recepción activa otra máquina secuencial llamada «Máquina de envío de parámetros», que es explicada en la sección 3.6.5, cede el control de transmisión a esta, y espera a que termine de enviar los parámetros almacenados.

El tercer comando es la *verificación de memoria* (código 0x99). Al igual que en el caso de los parámetros, la máquina activa otra máquina secuencial llamada «Máquina de envío de memoria» que se explica en la sección 3.6.6, e igualmente cede el control de transmisión y esperará a que esta termine el envío de los datos contenidos en la memoria de elementos.

El diagrama de estados simplificado de la máquina de recepción de comandos (MRC) se encuentra en la Figura 3-37. Posterior a esta puede hallarse la Tabla 3-8 donde se muestran las entradas, salidas y transiciones del diagrama de estados. Finalmente, en la Figura 3-38 se encuentra el diagrama de bloques.

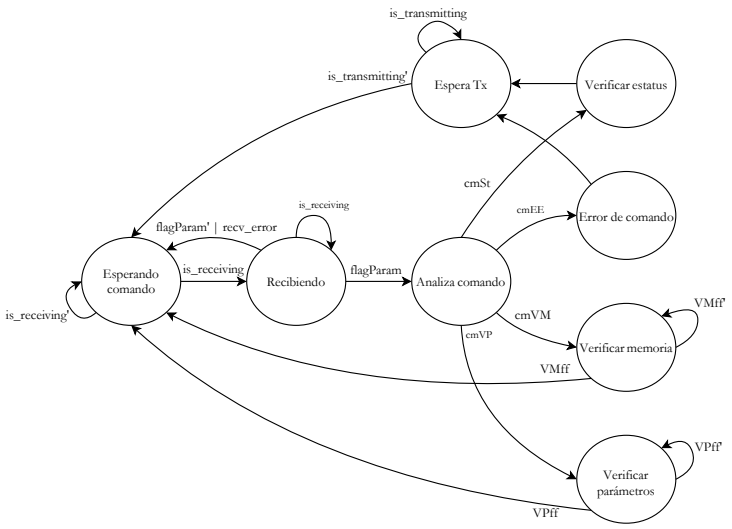


Figura 3-37. Diagrama de estados de la máquina secuencial de recepción de comandos.

Tabla 3-8. Descripción de señales y estados de la máquina secuencial de recepción de comandos.

Señales		
Tipo	**Nombre**	**Descripción**
Entrada	**is_receiving:** señal de recepción del puerto UART	Esta señal proviene del transceptor UART, indica que se está recibiendo un dato.
	recv_error: señal de error del puerto UART	Hubo un error en la recepción del dato por el puerto UART.
	received: señal de dato recibido del puerto UART	Se activa cuando se ha terminado de recibir el dato por el puerto UART.
	flagParam: bandera de finalización de recepción de parámetros	Proviene de la máquina de recepción de parámetros. Indica cuando ya se terminó la recepción.
	FFCfg: bandera de finalización de configuración	Proviene de la máquina de control de configuración, se activa cuando esta ha terminado su funcionamiento.
	is_transmitting: señal de transmisión del puerto UART	Se activa cuando el puerto UART se encuentra transmitiendo un dato.
	cmVS: comando de estatus	Se activa cuando el byte recibido por UART es igual al comando de solicitud de estatus (0x66).
	cmVP: comando de verificación de parámetros	Se activa cuando el byte recibido por UART es igual al comando de verificación de parámetros (0x44).
	cmVM: comando de verificación de memoria	Se activa cuando el byte recibido por UART es igual al comando de verificación de memoria (0x99).
	cmEE: comando no válido	Se activa cuando **cmVP, cmVM y cmVS** son cero, lo que indica que el byte recibido no corresponde a ningún comando válido. **cmEE = (~cmVS) & (~cmVP) & (~cmVM)**
	VPff: bandera de finalización de envío de parámetros	Se activa cuando la máquina de envío de parámetros llega a su estado final.
	VMff: bandera de finalización de envío de memoria	Se activa cuando la máquina de envío de memoria llega a su estado final.
	transmit_mem: inicio de transmisión proveniente de la máquina de envío de memoria	Indica el inicio de tranmisión para el puerto UART.
	transmit_param: inicio de transmisión proveniente de la máquina de envío de parámetros	Indica el inicio de tranmisión para el puerto UART.
	tx_bytemem [7:0]: byte a transmitir proveniente de la máquina de envío de memoria	Señal que contiene el byte que se transmitirá por el puerto UART.

	tx_byteparam [7:0]: byte a transmitir proveniente de la máquina de envío de parámetros	Señal que contiene el byte que se transmitirá por el puerto UART.
Salida	**flagsendparam**: activación de máquina de envío de parámetros	Se utiliza para activar a la máquina de envío de parámetros.
	flagsendmem: activación de la máquina de envío de memoria	Se utiliza para activar a la máquina de envío de memoria.
	transmit: inicio de transmisión de puerto UART	Señal de inicio de transmisión para el puerto UART.
	tx_byte [7:0]: byte a transmitir por el puerto UART	Byte que será tomado por el puerto UART para transmitir.
Estados		
Nombre (abreviatura)	**Descripción y transiciones**	**Valor de señales de salida (las señales no explícitas en cada estado tienen valor cero).**
Esperando comando (wait_cmd)	El estado de reposo en el que inicia la máquina de estados, esperando recibir un comando. Transiciones: **is_receiving = 0**, se mantiene en «wait_cmd». **is_receiving = 1**, se está recibiendo algo por el puerto UART, por lo que se pasa a «recibiendo».	Todas las señales de salida son cero.
Recibiendo (recibiendo)	Estado de espera hasta que se complete la recepción del dato, pero también se verifica si este será un comando. Transiciones: **flagParam = 0 | recv_error = 1**, el dato que se está recibiendo es un parámetro o no se recibió correctamente, por lo que se regresa a «wait_cmd». **flagParam = 1 & received = 1**, el dato que se recibió es un comando, se pasa a «ana_cmd». **flagParam = 1 & is_receiving = 1**, el dato que se recibirá es un comando, pero aún no se termina de recibir, se mantiene en «recibiendo».	Todas las señales de salida son cero.
Analiza comando (ana_cmd)	El estado sirve para analizar el comando recibido por UART. Transiciones: **cmVS = 1**, el comando corresponde a solicitud de estatus, por lo que se pasa a «status». **cmVP = 1**, el comando corresponde a verificación de parámetros, por lo que se pasa a «verifyparam». **cmVM = 1**, el comando corresponde a verificación de memoria, por lo que se pasa a «verifymem». **cmEE = 1**, el comando no corresponde a ninguno válido; se pasa a «cm_incorr».	Todas las señales de salida son cero.
Verificar estatus (status)	Envía el código de estatus dependiendo de la etapa en que esté el sistema. Transición incondicional hacia «wait_tx».	**transmit = 1** **si (~flagparam), tx_byte = 8'h11** **si (flagParam & ~FFCfg), tx_byte = 8'h22** **si(flagParam & FFCfg), tx_byte = 8'h44**
Espera Tx (wait_tx)	Estado de espera en lo que se finaliza la transmisión por el puerto UART. Transiciones: **is_transmitting = 1**, se mantiene esperando en «wait_tx». **is_transmitting = 0**, se pasa al estado «wait_cmd».	Todas las señales de salida son cero.
Verificar memoria (verifymem)	Se da inicio y se cede el control de transmisión a la máquina de envío de memoria. Transiciones: **VMff = 0**, se mantiene en «verifymem». **VMff = 1**, se terminó el envío de memoria, por lo que se pasa a «wait_cmd».	**flagsendparam = 0** **flagsendmem = 1** **transmit = transmit_mem** **tx_byte = tx_bytemem**
Verificar parámetros (verifyparam)	Se da inicio y se cede el control de transmisión a la máquina de envío de parámetros. Transiciones: **VPff = 0**, se mantiene en «verifyparam».	**flagsendparam = 1** **flagsendmem = 0** **transmit = transmit_param** **tx_byte = tx_byteparam**

	VPff = 1, se terminó el envío de parámetros, por lo que se pasa a «wait_cmd».	
Error de comando (cmd_incorr)	Se envía un byte 0xEE por el puerto UART para avisar que no se ha recibido ningún comando válido. Transición incondicional hacia «wait_tx».	**transmit = 1** **tx_byte = 8'hEE**

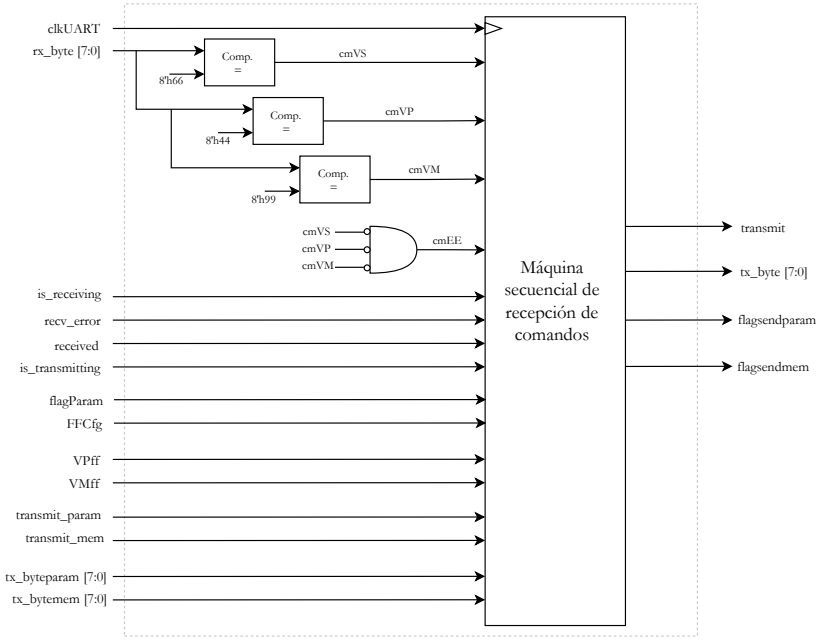


Figura 3-38. Diagrama de bloques de la máquina de recepción de comandos.

3.6.5 Máquina de envío de parámetros

Este circuito secuencial se encarga de realizar el envío de los parámetros recibidos por la interfaz de usuario, en el mismo orden en que fueron recibidos. Su activación depende exclusivamente de la máquina de recepción de comandos diseñada en la sección 3.6.4. El funcionamiento se describe en el diagrama de estados de la Figura 3-39. Las señales de salida son el byte de transmisión y la señal de inicio de transmisión, pero estas sólo pasan al puerto UART si la máquina de recepción de comandos lo permite; además, tiene una señal de finalización llamada **VPff** que se activa cuando termina el envío (más detalles en la Tabla 3-9). Al final del apartado, dentro de la Figura 3-40 se encuentra el respectivo diagrama de bloques.

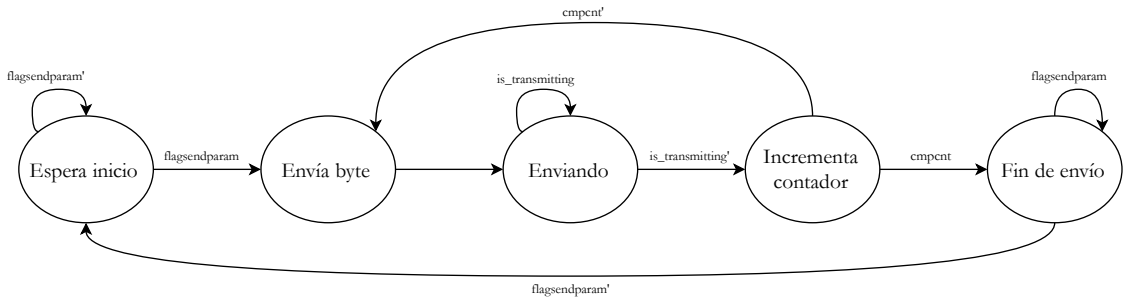


Figura 3-39. Diagrama de estados de la máquina secuencial de envío de parámetros.

Tabla 3-9. Descripción de señales y estados de la máquina secuencial de envío de parámetros.

Señales		
Tipo	**Nombre**	**Descripción**
Entrada	**flagsendparam:** señal de activación de la máquina	Esta señal le indica a la máquina cuándo comenzar el envío de parámetros.
	is_transmitting: señal de transmisión del puerto UART	Se activa cuando el puerto UART se encuentra transmitiendo un dato.
	cmpcnt: señal de comparación de contador	Se activa cuando el contador de bytes ha llegado a su valor máximo (36).
Salida	**VPff**: finalización de envío de parámetros	Se activa cuando esta máquina llega a su estado final.
	transmit_param: inicio de transmisión de puerto UART	Señal de inicio de transmisión para el puerto UART.
	tx_byteparam [7:0]: byte a transmitir por el puerto UART	Byte que será tomado por el puerto UART para transmitir.
	inc_cparam: incrementa el registro **countparam**	Señal que activa el reloj para el registro contador **countparam**.
	countparam [7:0]: contador de byte de parámetro	Permite elegir qué byte de los 36 que conforman los parámetros se enviará en determinado momento.
	byte_param [7:0]: byte del parámetro	Es el byte seleccionado por el selector **countparam** en terminado momento.
Estados		
Nombre (abreviatura)	**Descripción y transiciones**	**Valor de señales de salida (las señales no explícitas en cada estado tienen valor cero).**
Espera inicio (wait_start)	El estado de reposo en el que inicia la máquina de estados, esperando la señal de activación. Transiciones: **flagsendparam = 0**, se mantiene en «wait_start». **flagsendparam = 1**, se comienza el envío, se pasa a «send».	Todas las señales de salida son cero.
Envía byte (send)	Se envía el byte en cuestión seleccionado por el selector **countparam**.	**transmit_param = 1** **tx_byteparam = byte_param**
Enviando (sending)	Se espera a la finalización de la transmisión por UART. Transiciones: **is_transmitting = 1**, se mantiene en «sending». **is_transmitting = 0**, se terminó la recepción, por lo que se pasa a «inc_cntp».	**tx_byteparam = byte_param**
Incrementa contador (inc_cntp)	Incrementa el selector de byte para enviar el siguiente, o bien finalizar el envío. Transiciones: cmpcnt = 1, se finalizó el envío de todos los bytes, por lo que se pasa a «end_send». cmpcnt = 0, aún quedan bytes por enviar, por lo que se regresa a «send».	**inc_cparam = 1**
Fin de envío (end_send)	Estado final en el que se indica la finalización del envío de datos. Transiciones: **flagsendparam = 1**, se mantiene en «end_send» en lo que espera a que la MRC detecte que esta ya terminó. **flagsendparam = 0**, se pasa al estado inicial «wait_start».	**VPff = 1**

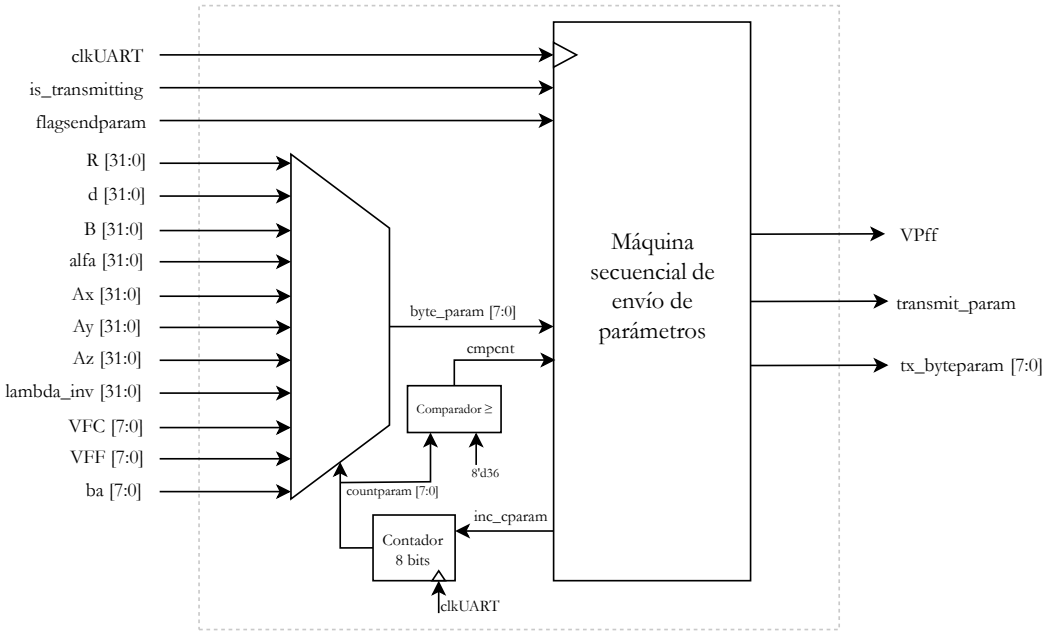


Figura 3-40. Diagrama de bloques de la máquina de envío de parámetros.

3.6.6 Máquina de envío de memoria

Bastante similar a la máquina de envío de parámetros, la máquina de envío de memoria (MEM) se encarga de controlar el envío de los datos contenidos en la memoria de elementos hacia el puerto UART. Está bajo control de la máquina de recepción de comandos, por lo que esta solo se activa si la MRC se lo indica. Asimismo, la MRC debe cederle el control del puerto a esta máquina.

Viéndolo desde el lado de la memoria, la MEM debe tener control sobre la lectura de la memoria en cuanto se active para poder realizar el envío de los datos; sin embargo, la etapa de ejecución tiene prioridad, por lo que si se está haciendo el proceso de ejecución no se enviarán los datos de la memoria correctamente. El diagrama de estados de la máquina secuencial se encuentra en la Figura 3-41. Posteriormente se encuentra la Tabla 3-10 con los estados y las señales de entrada y salida, y finalmente en la Figura 3-42 se muestra el diagrama de bloques de la MEM.

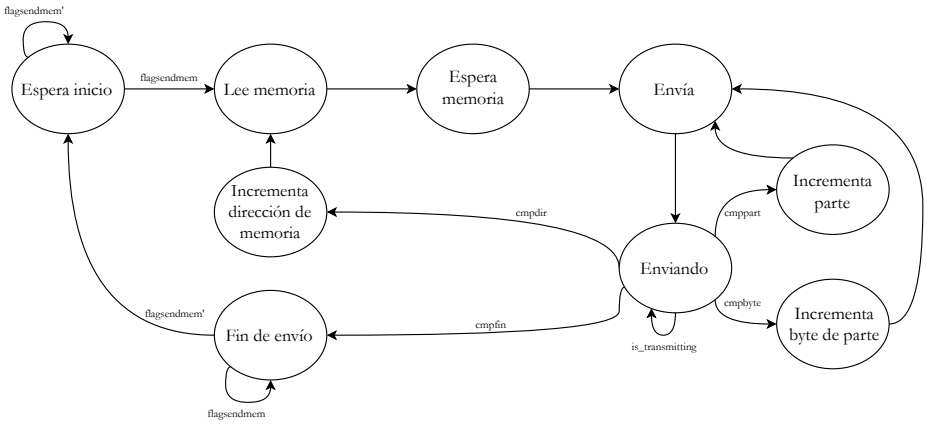


Figura 3-41. Diagrama de estados de la máquina secuencial de envío de memoria.

Tabla 3-10. Descripción de señales y estados de la máquina secuencial de envío de memoria.

Señales		
Tipo	**Nombre**	**Descripción**
Entrada	**flagsendmem:** señal de activación de la máquina	Esta señal le indica a la máquina cuándo comenzar el envío de parámetros.
	is_transmitting: señal de transmisión del puerto UART	Se activa cuando el puerto UART se encuentra transmitiendo un dato.
	cmpbyte: señal de comparación de contador	Se mantiene en nivel alto cuando el contador **part_bytepart** es menor a 3.
	cmppart: señal de comparación de contador	Se mantiene en nivel alto cuando el contador **part_datomem** es menor a $n_{nuc} - 1$.
	cmpdir: señal de comparación de contador	Se mantiene en nivel alto cuando el contador **addr_sendmem** es menor a **addrMcfg** (Figura 3-28).
	cmpfin: señal de comparación de contador	Se activa cuando ninguno de los otros comparadores está en nivel alto. **cmpfin = (~cmpbyte) & (~cmppart) & (~cmpdir)**
Salida	**VMff**: finalización de envío de memoria	Se activa cuando esta máquina llega a su estado final.
	transmit_mem: inicio de transmisión de puerto UART	Señal de inicio de transmisión para el puerto UART.
	tx_bytemem [7:0]: byte a transmitir por el puerto UART	Byte que será tomado por el puerto UART para transmitir.
	byte_mem [7:0]: byte de parte	Es el byte que es elegido por el selector **part_bytepart.**
	ena_memsm: señal de activación para la memoria de elementos	Sirve para activar la lectura de la memoria por esta máquina.
	inc_part1: incrementa el registro **part_datomem**	Señal que activa el reloj para el registro contador **part_datomem**.
	inc_part2: incrementa el registro **part_bytepart**	Señal que activa el reloj para el registro contador **part_bytepart**.
	inc_addr: incrementa el registro **addr_sendmem**	Señal que activa el reloj para el registro contador **addr_sendmem**.
	part_datomem [7:0]: registro de parte de memoria	Permite elegir 32 bits dentro de la localidad, correspondientes a la info. de un elemento.
	part_bytepart [1:0] : registro de byte de parte	Permite elegir un byte de los 32 bits de info. del elemento.
	addr_sendmem [6:0]: registro de dirección de memoria	Contador de direcciones de memoria.
Estados		
Nombre (abreviatura)	**Descripción y transiciones**	**Valor de señales de salida (las señales no explícitas en cada estado tienen valor cero).**
Espera inicio (wait_start)	El estado de reposo en el que inicia la máquina de estados, esperando la señal de activación. Transiciones: **flagsendmem = 0**, se mantiene en «wait_start». **flagsendmem = 1**, se ordena la activación, se pasa a «fetch».	Todas las señales de salida son cero.
Lee memoria (fetch)	Se activa la memoria de elementos para que saque el valor en la localidad «addr_sendmem». Transición incondicional hacia «wait_mem».	**ena_memsm = 1**
Espera memoria (wait_mem)	Se espera un ciclo de reloj extra para que la memoria tenga los datos listos, ya que tarda 2 ciclos de reloj. Transición incondicional hacia «send».	**ena_memsm = 1**
Envía (send)	Comienza el envío del byte «byte_mem» por el puerto UART. Transición incondicional hacia «sending».	**transmit_mem = 1** **tx_bytemem = byte_mem**
Enviando (sending)	Espera a que finalice la transmisión del byte enviado. Transiciones: **is_transmitting = 1**, se mantiene en «sending». **is_transmitting = 0 & cmpbyte = 1,** se pasa a «inc_pbp».	Todas las señales de salida son cero.

	is_transmitting = 0 & cmppart = 1, se pasa a «inc_part». **is_transmitting = 0 & cmpdir = 1,** se pasa a «inc_dir». **is_transmitting = 0 & cmpfin = 1,** se pasa a «end_send».	
Incrementa byte de parte (inc_pbp)	En este estado se incrementa el contador **part_bytepart**. Transición incondicional hacia «send».	**inc_part2 = 1**
Incrementa parte (inc_part)	En este estado se incrementa el contador **part_datomem**. Transición incondicional hacia «send».	**inc_part1 = 1**
Incrementa dirección de memoria (inc_dir)	En este estado se incrementa el contador **addr_sendmem**. Transición incondicional hacia «fetch».	**inc_addr = 1**
Fin de envío (end_send)	Estado final en el que se indica la finalización del envío de datos. Transiciones: **flagsendparam = 1**, se mantiene en «end_send» en lo que espera a que la MRC detecte que esta ya terminó. **flagsendparam = 0**, se pasa al estado inicial «wait_start».	**VMff = 1**

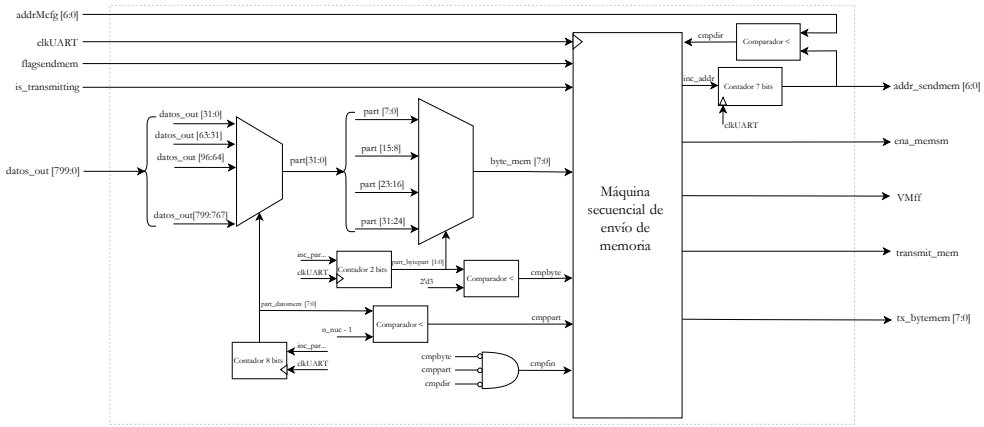


Figura 3-42. Diagrama de bloques de la máquina de envío de memoria.

3.6.7 Interconexión de módulos

Habiendo completado los módulos de los que está compuesta la interfaz de usuario y almacenamiento de datos, para terminar con el diseño se interconectaron como se muestra en la Figura 3-43, completando la estructura interna genérica que fue establecida previamente en la Figura 3-31.

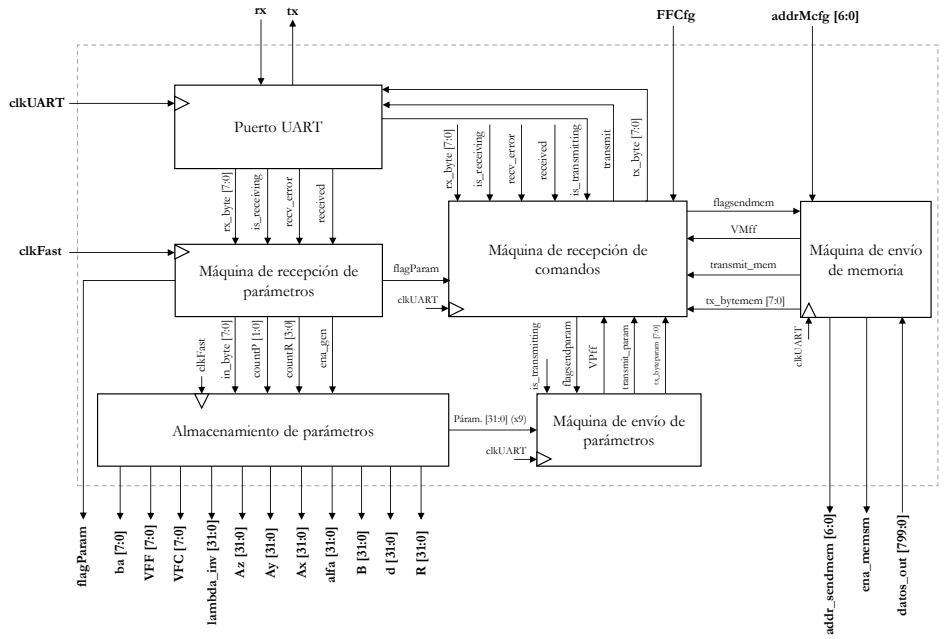


Figura 3-43. Interconexión de módulos para la interfaz de usuario y almacenamiento de parámetros.

3.7 DISEÑO DE LA ETAPA DE EJECUCIÓN

En la sección 3.5 se concluyó con el diseño de la etapa de configuración, donde al final del proceso se tendrán almacenados los datos referentes a los elementos del arreglo reflectivo (x_i, y_i, r_i; secc. 2.1.4) en las «memorias de elementos», de tal manera que se tienen disponibles los datos de n_{es} elementos en cada localidad conjunta de memoria.

La etapa de ejecución, tal como se presenta en la arquitectura de la sección 3.3, consiste en que, a partir de un par de ángulos de orientación θ_0 y φ_0, se obtengan las fases requeridas de n_{es} elementos simultáneamente mediante la operación matemática (2.b); posteriormente las fases serán redondeadas hacia la fase discreta más cercana y finalmente se codificarán, para ser enviadas a través de los puertos seriales hacia los dispositivos esclavos.

Se comenzará el diseño de la etapa de ejecución (resaltada sobre la estructura interna en la Figura 3-44) con un análisis de tiempos para discutir las posibles alternativas de implementación.

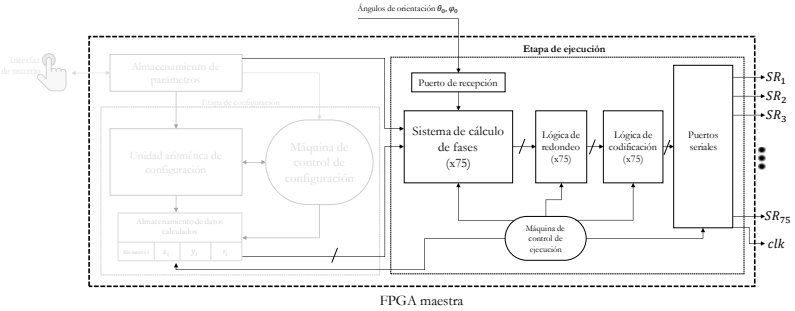


Figura 3-44. Etapa de ejecución resaltada sobre estructura de FPGA maestra.

3.7.1 Análisis de tiempos para alternativas de implementación

El número de bits que se debe enviar por los puertos seriales por cada fase está directamente relacionado al valor de b_a, es decir, se necesitan solo b_a bits por cada fase de cada elemento. Por ende, el caso más simple se enviarán solo un par de bits por cada fase, mientras que en el caso más complejo se necesitarán enviar cinco (véanse reqs. en secc. 3.1).

Con respecto al tiempo de respuesta, el funcionamiento más rápido debería darse en el caso en que $b_a = 2$, puesto que solo se requerirá de dos ciclos de reloj para enviar cada fase; y el caso más tardado será cuando $b_a = 5$, dado que serán necesarios cinco ciclos de reloj para cada fase. Las dos formas posibles de la etapa de ejecución serán mencionadas a continuación.

1. Mediante «pipelining»[8]: teniendo registros a la salida de cada etapa, se tendría entonces un encauzamiento de tres etapas (ilustrado en Figura 3-45), siendo la primera la unidad de cálculo de fases, la segunda el módulo de redondeo, y la tercera el de codificación. De tal forma se conseguiría tener un desempeño que, una vez que la segmentación estuviese llena, tuviera lista una fase codificada en cada ciclo de reloj.

2. Eliminando los registros (latencia mínima): si se eliminan los registros presentes en la Figura 3-45, los datos del elemento y de los ángulos de orientación se propagan libremente por la unidad de cálculo de fases y el resto de módulos, teniendo así el retardo mínimo posible desde que los datos salen de las memorias hasta que se obtienen sus fases respectivas codificadas.

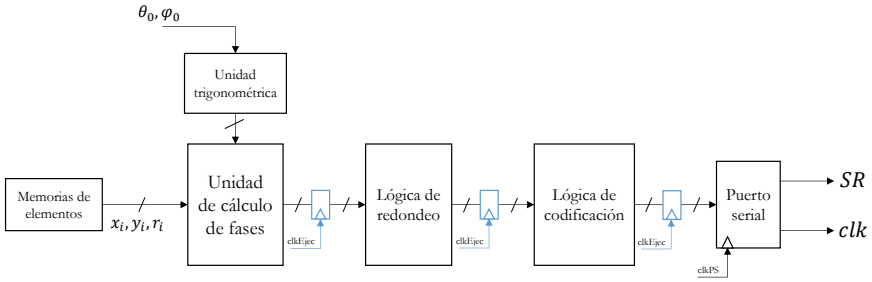


Figura 3-45. Etapa de ejecución segmentada por registros.

Con base en ambas alternativas se hacen las siguientes consideraciones: la primera implementación («pipeline») sería de gran utilidad para el caso de funcionamiento más rápido, es decir, cuando $b_a = 2$, ya que esto permitiría tener lista la fase siguiente inmediatamente después de los dos ciclos de reloj de **clkPS** necesarios para enviar la anterior. No obstante, para el caso en que $b_a > 2$, el flujo del encauzamiento tendría que detenerse hasta que los puertos terminen de enviar las fases, para entonces enviar las siguientes. Aunque estas se podrían almacenar en registros temporales, no resulta de mayor utilidad si de cualquier forma se debe esperar a la finalización del envío por el puerto serial.

La segunda alternativa (latencia mínima) podría ser más útil para casos en los que $b_a > 2$, es decir, donde se necesiten más de dos ciclos de reloj para completar el envío de cada fase. Esto es debido a que así es más probable que se tuviese el tiempo suficiente de propagación, para que la siguiente fase fuera calculada, redondeada y codificada mientras se envía la anterior. En el caso de funcionamiento más rápido, cuando $b_a = 2$, es posible que fuera necesario limitar la frecuencia de los puertos seriales para respetar la latencia total de la lógica combinacional (detalles en secc. 2.2.2).

Para analizar a mayor detalle las dos alternativas y poder tomar una decisión, hace falta hacer también un análisis temporal con respecto a la distribución de las señales en los dispositivos esclavos, para ambos casos establecidos en los requerimientos (secc. 3.1). Se comenzará en el caso más grande, donde se necesitan distribuir las 48,000 señales de control para un arreglo de 1500 elementos de 5 bits.

Primero, se asumirá que el máximo de pines libres disponibles en la FPGA maestra es de 80, reservando 5 para señales que no sean de datos hacia los puertos seriales. Esto da un total de 75 dispositivos esclavos, cada uno controla a 20 elementos. Recordando que se propuso una frecuencia $f_{sr} = 150\ MHz$ para la comunicación serial, se realizan los siguientes cálculos con respecto al dispositivo esclavo:

1. Tiempo necesario para llenar el registro de corrimiento de cada esclavo: $t_{sr} = \frac{(e_{es}\ b_a) + 1}{f_{sr}} = \frac{(20)(5) + 1}{150\ MHz} = 0.673\ \mu s$
2. Latencia de la lógica de decodificación: $t_{dec} < 10\ ns$
3. Tiempo total del dispositivo esclavo: $t_{es} = t_{sr} + t_{dec} = 673\ ns + 10\ ns \approx 0.7\ \mu s$

[8] En la segmentación encauzada o «pipeline», un proceso se divide en varias etapas con la finalidad de que cada una opere con distintos datos en cada momento, consiguiendo así un paralelismo temporal dentro de un sistema de ejecución único (más detalles en secc. 2.2.3).

Con respecto a la FPGA maestra, cada fase tardará 5 ciclos de reloj en ser enviada, por ende la siguiente fase debe estar lista en menos de 5 ciclos de reloj de los dispositivos esclavos. Entonces, el tiempo requerido para obtener la siguiente fase (t_{te}) está sujeto a la siguiente condición:

$$t_{te} < 5/f_{sr} \rightarrow t_{te} < 33.333\ ns$$

El tiempo de respuesta total sería entonces definido por la siguiente expresión:

$$t_{rt} = t_{te} + t_{es}$$

Pasando ahora al caso más sencillo, donde se trabajará con un arreglo de 1500 elementos de 2 bits y 4 diodos PIN, lo que implica un total de 6000 líneas de control. Se conserva el mismo total de 75 dispositivos esclavos controlando 20 elementos cada uno. Ahora se propone una frecuencia de comunicación serial a 60 MHz y se realiza el mismo análisis:

1. Tiempo necesario para llenar el registro de corrimiento de cada esclavo: $t_{sr} = \frac{(e_{es}\ b_a) + 1}{f_{sr}} = \frac{(20)(2) + 1}{60\ MHz} = 0.683\ \mu s$
2. Latencia de la lógica de decodificación: $t_{dec} < 10\ ns$
3. Tiempo total del dispositivo esclavo: $t_{es} = t_{sr} + t_{dec} = 683.3\ ns + 10\ ns \approx 0.7\ \mu s$

Con respecto a la FPGA maestra, recordando que cada fase codificada necesita de dos ciclos de reloj para ser enviada, la siguiente fase debe estar lista en un tiempo (t_{te}) menor a 2 ciclos de reloj de los dispositivos esclavos, es decir:

$$t_{te} < 2/f_{sr} \rightarrow t_{te} < 33.333\ ns \qquad (3.a)$$

Se puede observar que aun reduciendo la frecuencia de comunicación serial (en comparación al caso anterior), se tiene un tiempo de respuesta menor a un microsegundo si se consigue respetar t_{te}, cuyo valor límite es el mismo en ambos análisis. Habiendo hecho estos cálculos, lo siguiente es definir los requisitos de la lógica de la FPGA maestra, para que en ambas alternativas se pueda cumplir con la restricción (3.a).

3.7.1.1 Alternativa 1: encauzamiento

Para la primera alternativa que sería tipo «pipeline» se tiene que, tal como está propuesta en la Figura 3-45, después de tres de ciclos de reloj de **clkEjec** estará lista la primera fase codificada, y de ese momento en adelante se tendrá lista una nueva en cada ciclo de reloj. Esto significa que, para poder cumplir con la restricción (3.a), la latencia de cada etapa de la segmentación deberá ser menor a 33 ns. Asimismo, también se deben considerar los tiempos asociados a los registros que están entre cada etapa (detallados en secc. 2.2.2).

Consultando la hoja de datos de la familia Spartan 7 de FPGA de Xilinx [47], el tiempo de «setup» para los registros en estos dispositivos es $t_{su} = 0.84\ ns$. Por el contrario, el tiempo de «hold» tiene un valor negativo, por lo que puede ser descartado de los análisis. Obsérvese a continuación la Figura 3-46, en ella se encuentran las variables para las latencias de cada bloque y de cada etapa de la segmentación. t_{lc1} es la latencia de la primera etapa, considerando que está compuesta de la unidad trigonométrica y la unidad de cálculo. t_{lc2} consiste únicamente en el módulo de redondeo, y t_{lc3} se refiere a la latencia del módulo de codificación. Por otro lado, los tiempos t_{bn} ya consideran la suma de cada t_{lcn} con el t_{su} de la FPGA mencionada.

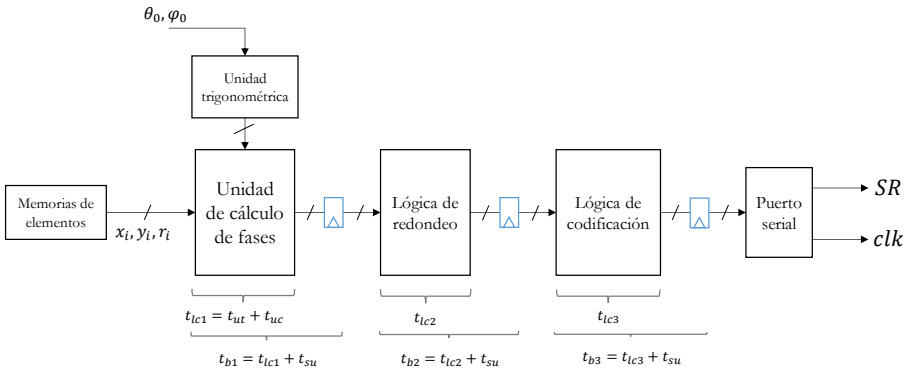


Figura 3-46. Retardos en las etapas del encauzamiento.

Tal como está estructurada la Figura 3-46, para que la primera alternativa de implementación cumpla con la condición (3.a), los tres valores de latencia $t_{b1}, t_{b2}\ y\ t_{b3}$ deberán tener un valor menor a 33 ns. Si esto se consigue, entonces es viable de implementar.

3.7.1.2 Alternativa 2: latencia mínima

Para el caso de la latencia mínima, se tendrían las tres etapas de la Figura 3-45 sin ningún registro de por medio. Esto conlleva a que tan pronto los datos del elemento en cuestión salgan de las memorias, junto con los ángulos de orientación, se propaguen las señales por los tres bloques hasta llegar a la salida de la lógica de codificación, donde el resultado será tomado por el puerto serial para enviarlo hacia el esclavo.

Para que esta alternativa de implementación sea factible de implementar, es decir, cumpla con la restricción (3.a), la suma de las latencias de los tres bloques deberá ser menor a 33 ns. Expresado formalmente según como fueron nombrados los tiempos en la Figura 3-46, la condición se expresa como $t_{lc1} + t_{lc2} + t_{lc3} < 33.333\ ns$.

Si se es realista, se ve complicado el hecho de satisfacer la condición anterior. El problema recae principalmente en el bloque de cálculo de fases, pues la operación (2.b) a realizar es un cálculo computacionalmente complejo que seguramente por sí solo requiera de al menos 33 ns para ejecutarse, si no es que incluso más tiempo. Por esa razón es que se diseñarán los módulos pensados para trabajar con la alternativa 1 discutida en la sección anterior 3.7.1.1. Se comenzará por la unidad de cálculo de fases.

3.7.2 Unidad de cálculo de fases

Del diagrama de bloques presentado en la Figura 3-44 y omitiendo de momento el puerto de recepción (por donde se reciben el par de ángulos de orientación), la primera etapa es la unidad que calcula los ángulos de desfase, es decir, la unidad aritmética que realiza la operación (2.b) y que se destaca en la Figura 3-47. Este módulo puede ser implementado mediante intercionexiones de los bloques aritméticos y trigonométricos previamente diseñados (seccs. 3.5.1.1 y 3.5.1.2), ya que no se requiere de ninguna operación nueva.

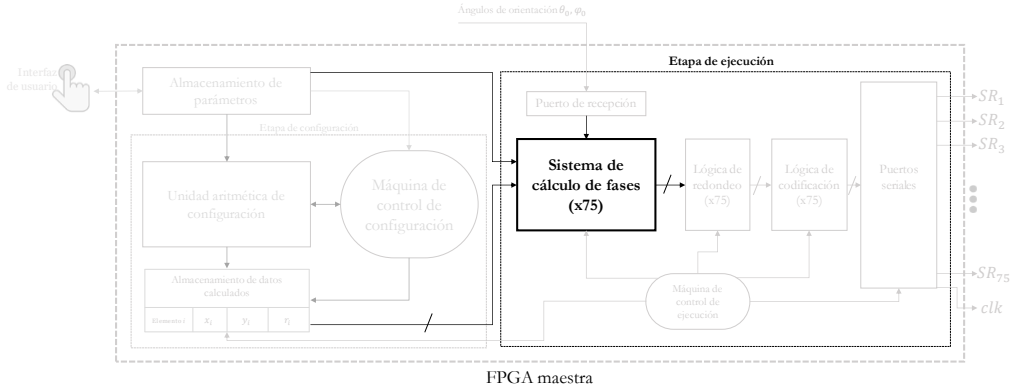


Figura 3-47. Sistema de cálculo de fases resaltado sobre la estructura interna.

En consecuencia, en la Figura 3-48 se muestra el diagrama de bloques con las interconexiones respectivas. En la ilustración se observa que las entradas al módulo son, además de los ángulos de orientación (θ_0, φ_0), los datos del elemento en cuestión $(x_i, y_i, r_i$; secc. 2.1.4) y el valor del número de onda «k» (definido en la expresión (2.a)). Igualmente es posible notar que a la salida de la unidad se encuentra un registro cuyo reloj es nombrado **clkUC**, siguiendo con el diseño acorde a lo establecido en el apartado 3.7.1.1. (Nota: este diseño es modificado posteriormente en la sección 3.7.8)

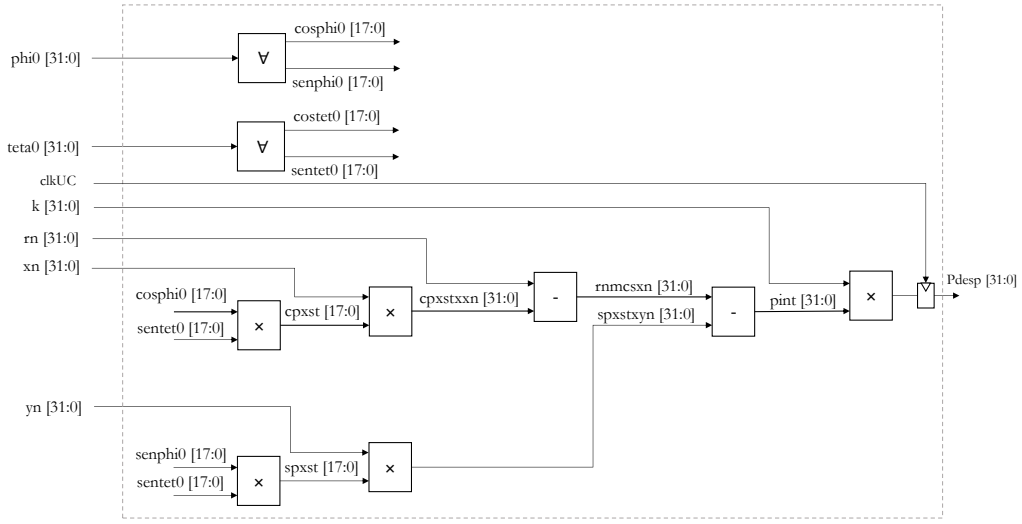


Figura 3-48. Unidad de cálculo de fases para la etapa de ejecución.

3.7.3 Unidades de redondeo y codificación

La segunda etapa presente en el diagrama de la Figura 3-44 es la lógica de redondeo. Este bloque es necesario debido a la cantidad finita de fases que son alcanzables por los elementos del arreglo. La salida de la unidad de cálculo (diseñada en secc. 3.7.2) es una fase dada en radianes, que además de tener un valor real, su magnitud puede ser mayor a 2π. Esto significa que la unidad inicialmente debe convertir la fase real a un ángulo equivalente que se encuentre entre 0 y 2π rad, para posteriormente redondear este nuevo valor al más discreto más cercano. La etapa final es la de codificación, donde se convertirá el valor discreto en radianes a su respectivo código como lo indique la «matriz de funcionamiento» (explicada al final de secc. 2.1.4). Ambas etapas se encuentran resaltadas en la Figura 3-49.

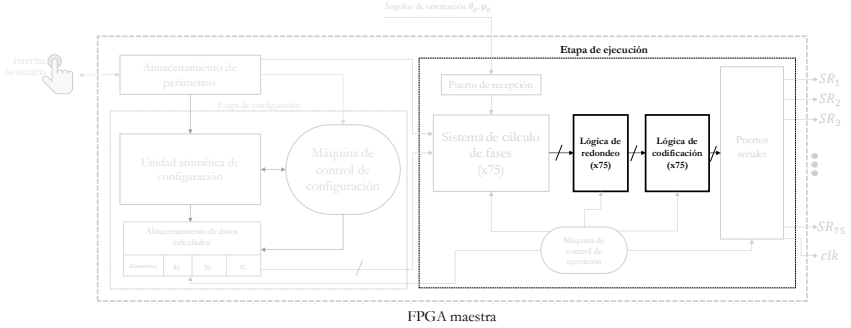


Figura 3-49. Etapas de redondeo y codificación resaltadas sobre la estructura interna.

A continuación se describe un ejemplo hipotético del procedimiento que realizará la unidad de redondeo, asumiendo que cada elemento del arreglo puede desfasar en 4 ángulos: 0, 90, 180 y 270 [°].

$$\left.\begin{array}{c} P_{desp} = 48.65\ rad = 2787.4397\ ° \\ |P_{desp}|\ mayor\ a\ 2\pi\ rad \\ Fase\ equivalente: 48.65\ rad - (7)(\ 2\pi\ rad) = 4.6677\ rad = 267.4395° \end{array}\right\} \text{Equivalencia}$$

$$\left.\begin{array}{c} Fase\ más\ cercana\ a\ 267.45° \rightarrow 270° \\ \therefore\ Fase\ redondeada\ a\ 270° \end{array}\right\} \text{Redondeo}$$

Tal como se realiza en el ejemplo anterior, la primera parte consiste en convertir el valor de P_{desp} a un equivalente (P_{eq}) en caso de que su magnitud sea mayor a 2π rad, o conservarlo si es que es menor. Por ello se comenzará con este submódulo

que llevará por nombre «*submódulo de equivalencia*». El algoritmo descrito de forma simplificada se encuentra en la Figura 3-50, junto con un ejemplo resaltado en color rojo para mejor comprensión. La notación $p_f[x]$ indica que se conserva solo la parte fraccionaria de un valor denominado x.

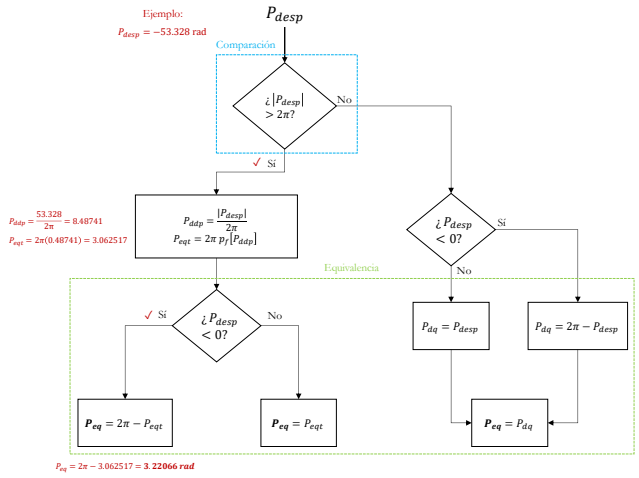


Figura 3-50. Funcionamiento del submódulo de equivalencia de la unidad de redondeo.

A partir del diagrama anterior surge uno de los «retos» de este submódulo, la división necesaria de la magnitud de la fase P_{desp} entre 2π para el cálculo de P_{ddp}, ya que hasta ahora se ha evitado el uso de división por ser una operación no tan sencilla de implementar. Después de una investigación sobre distintas alternativas para división aritmética a nivel de hardware, con la esperanza de encontrar algún algoritmo como el de la raíz cuadrada que se pudo implementar de forma puramente combinacional, no se encontró ninguno que fuera viable de diseñar para no recurrir a máquinas secuenciales.

Sin embargo, también es posible notar que la única división que se realiza en todo el proceso es entre el valor constante de 2π. La alternativa más factible y sencilla es la de almacenar de forma previa el valor inverso de π, para entonces realizar la multiplicación de la fase por este valor inverso y posteriormente realizar un corrimiento hacia la derecha, consiguiendo la división entre 2. Con esta modificación se superaría la dificultad mayor de esta unidad, y el resto de lógica puede ser implementada también de forma combinacional, como se ilustra en la Figura 3-51.

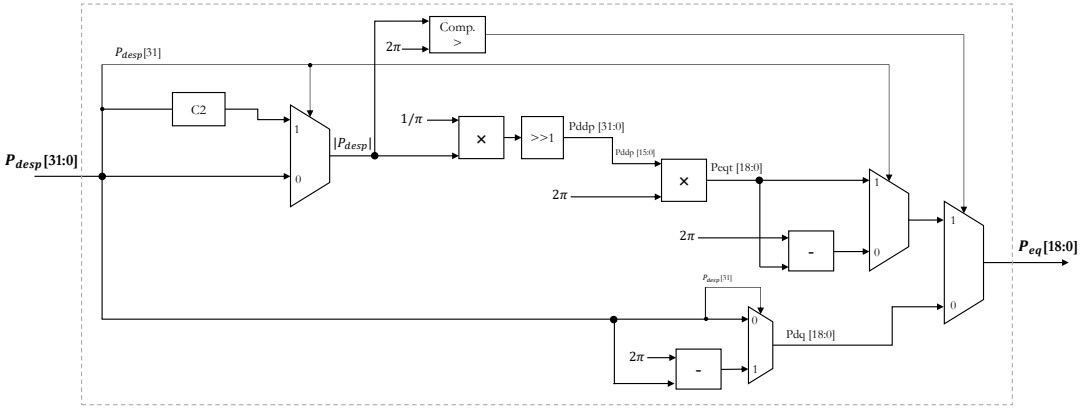


Figura 3-51. Estructura interna del submódulo de equivalencia.

Teniendo a la salida del submódulo de equivalencia el respectivo valor P_{eq} entre 0 y 2π rad, lo siguiente es «redondear» el ángulo obtenido a la fase más cercana que sea conseguible por el arreglo. Lo que se procurará es que cada fase discreta tenga una distribución equitativa de ángulos que se redondeen hacia esta, tanto menores como mayores. Para ilustrar esta explicación se adjunta la Figura 3-52, donde el arreglo tiene 8 fases (0, 45, 90, 135, 180, 225, 270, 315 [°]), y cada valor tiene asociado un rango de fases equitativamente distribuidas que son redondeadas hacia las discretas, siendo indicadas por regiones de colores.

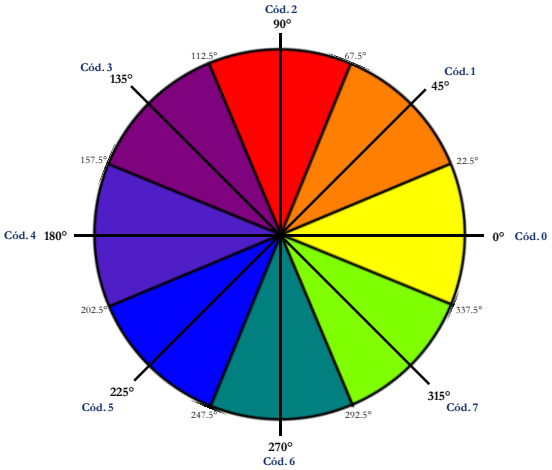


Figura 3-52. Ilustración de regiones de redondeo de fases para un arreglo de 8 fases.

Por otro lado, nótese que en la Figura 3-52 cada fase discreta tiene asignado un código. Los códigos siguen un orden ascendente partiendo del desfasamiento en 0°. Siendo el caso contrario, si se conoce el código de la fase (P_{cod}) y se desea saber su valor real en grados o radianes (P_{dc}), se puede recurrir a la expresión (3.b), recordando que b_a es la cantidad de bits del arreglo y N la cantidad de fases, definida en la ecuación (2.c).

$$P_{dc}[°] = (P_{cod})\left(\frac{360°}{N}\right) \; o \; P_{dc}[rad] = (P_{cod})\left(\frac{2\pi}{N}\right) \tag{3.b}$$

Volviendo a la etapa de redondeo, se puede implementar de distintas maneras completamente combinacionales. Una de ellas es la de la de restar la fase equivalente P_{eq} con todas las fases discretas, para después comparar resultados y redondear según la resta que tenga menor valor; otra alternativa es la de dividir la fase equivalente entre $2\pi/N$, y después contar con lógica que convierta a la fase discreta correspondiente en función del resultado de la división.

Analizando ambas opciones, la segunda opción resulta más sencilla de implementar en cuanto a recursos y complejidad, lo que ayuda a simplificar esta unidad y por ende disminuir la latencia. A continuación se explicará en qué consiste el «*submódulo de redondeo y codificación*» partiendo del diseño en la Figura 3-53. Posteriormente se encuentra un ejemplo.

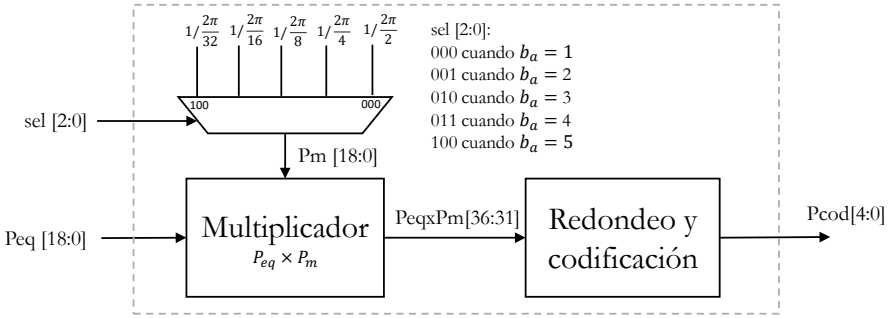


Figura 3-53. Submódulo de redondeo y codificación.

Multiplexor: permite elegir el valor de inverso de $2\pi/N$, denominado P_m, en función del número de bits del arreglo. Este valor se multiplicará por la fase equivalente que proviene del submódulo de equivalencia, para obtener el resultado como si se hubiera hecho la división $P_{eq}/(2\pi/N)$.

Multiplicador: la multiplicación aritmética se realiza entre la fase equivalente (P_{eq}) proveniente del submódulo de equivalencia y P_m. El producto de dos números de 19 bits será uno de 38, pero para la siguiente etapa de redondeo y codificación únicamente se necesitan 5 bits de la parte entera y solo 1 de la parte fraccionaria, por ello es que a la salida del multiplicador sólo están los bits 36 al 31.

Lógica de redondeo y codificación: al resultado del multiplicador se le suma el primer bit de la parte fraccionaria como si fuera un valor entero, de tal manera que si el resultado del producto cuenta con la parte fraccionaria mayor o igual a 0.5 (indicado por el primer bit de fracción), se redondeará al número entero siguiente. Por el formato binario utilizado, a la salida ya se tiene la fase codificada directamente, lista para ser capturada por el puerto serial.

Para comprender mejor el funcionamiento del bloque de redondeo y codificación se explicará un breve ejemplo. Supóngase que la fase equivalente P_{eq} es de 3.61283 rad (207°) y se trabaja con el arreglo de ejemplo de la Figura 3-52, con $b_a = 3$. El procedimiento es el siguiente:

$$P_{eq} = 3.61283 \quad y \quad P_m = 1/\frac{2\pi}{2^3} = 1.27323$$
$$P_{eq} \times P_m = 4.5999 = \underline{00100}.\underline{10011}_b \ldots \quad \text{Redondeo}$$
$$P_{cod} = \overline{00100}_b + 00001_b = 00101_b \quad \text{Codificación}$$

Se observa que el resultado final es la fase 101_b (5_d), cuyo valor corresponde a 3.927 rad o 225°. Esta fase es efectivamente aquella en cuya región se encuentra el ángulo de 207°, tal como es posible observar en la Figura 3-54.

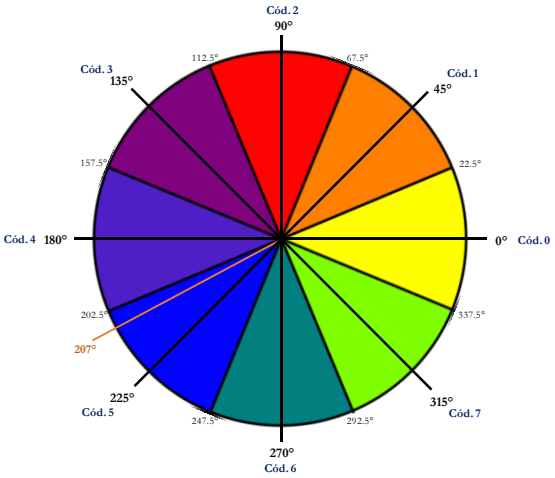


Figura 3-54. Se muestra que 207° pertenecen a la región azul, cuya fase discreta es 225°.

Para terminar de ejemplificar se realizará el análisis del mismo caso, pero haciendo el cambio a $b_a = 5$, es decir, el número de bits más grande con 32 posibles fases discretas. En este caso, el procedimiento es el siguiente:

$$P_{eq} = 3.61283 \quad P_m = 1/\frac{2\pi}{2^5} = 5.09295$$
$$P_{eq} \times P_m = 18.3999 = \underline{10010}.\underline{01100}_b \ldots \quad \text{Redondeo}$$
$$P_{cod} = \overline{10010}_b + 00000_b = 10010_b \quad \text{Codificación}$$

El resultado final es la fase discreta codificada 10010_b (18_d), cuyo valor real, obtenido por la expresión (3.b), corresponde 202.5°, por lo que resulta en un error de sólo 4.5° entre la fase equivalente de 207° y la fase redondeada.

Se finaliza el diseño de las unidades de redondeo y codificación con un diagrama de bloques (Figura 3-55) de los dos submódulos de los que está compuesta, observándose también que cada uno tiene un registro a su salida, para seguir con el objetivo de la implementación del «pipelining» en la etapa de ejecución. Lo consiguiente es la creación de la máquina de control de ejecución y de los puertos seriales. (Nota: estos diseños son posteriormente modificados en las secciones 3.7.9 y 3.7.10)

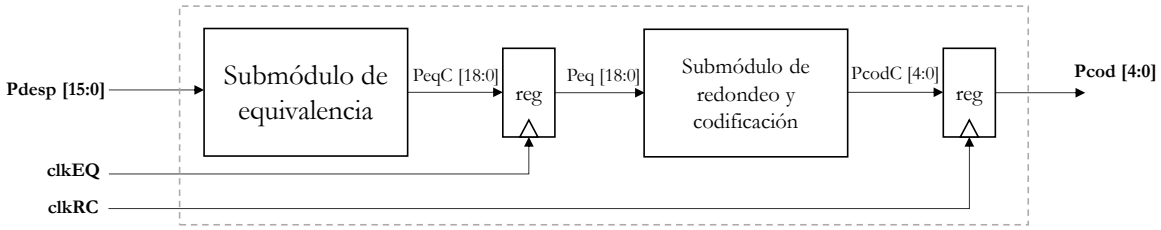


Figura 3-55. Unidad de equivalencia, redondeo y codificación (ERC).

3.7.4 Puertos seriales

Los puertos seriales (resaltados en la Figura 3-56) son las máquinas encargadas del envío de las fases codificadas hacia los dispositivos esclavos. A su salida tienen una señal de reloj y una señal de datos. Su inicialización se realiza a través de un bit de activación y para monitoreo cuentan con una bandera de finalización, que servirá para indicarle a la máquina de control cuando ya han terminado el envío de una fase.

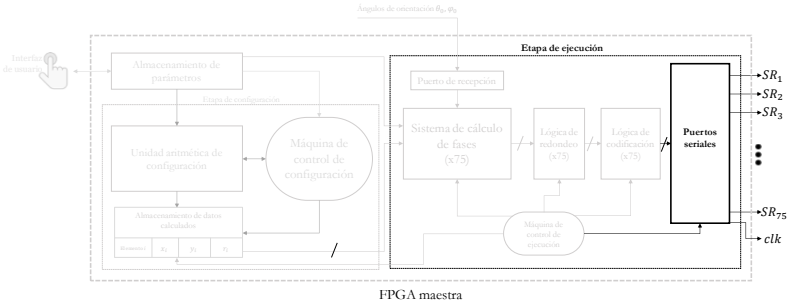


Figura 3-56. Puertos seriales resaltados sobre la estructura interna.

Para estos bloques se ha diseñado la máquina secuencial de la Figura 3-57, cuyos estados y señales respectivas se encuentran detallados en la Tabla 3-11.

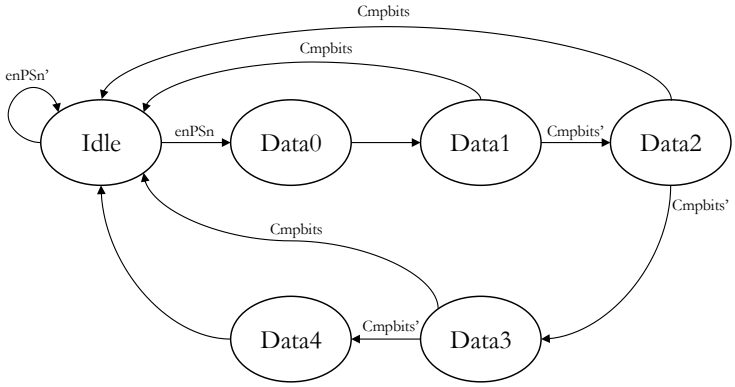


Figura 3-57. Diagrama de estados de la máquina secuencial del puerto serial.

Tabla 3-11. Descripción de señales y estados de la máquina secuencial del puerto serial.

Señales		
Tipo	**Nombre**	**Descripción**
Entrada	**enPSn**: señal de activación del puerto serial *n*	Esta señal proviene de la máquina de control, es la bandera de inicio para el puerto serial.
	Pcod [4:0]: fase codificada a enviar	El valor de la fase codificada que se enviará por el puerto ingresa por esta señal de 5 bits.
	Cmpbits: señal del comparador de bits enviados	La señal se activa cuando el número de bits enviados iguala al valor de n_b.

Salida	**enclkPSn**: señal de activación reloj del puerto serial *n*	Señal que controla el reloj de salida del puerto serial.
	outPSn: señal de dato del puerto serial *n*	Bit de salida del puerto serial.
	sPSn: señal de finalización puerto serial *n*	Bandera de disponibilidad del puerto serial. Se activa siempre que el puerto no esté trabajando.
	clkoutPSn: reloj de salida del puerto serial *n*	Reloj para la comunicación síncrona con el dispositivo esclavo.
	ep [2:0]: estado presente de la máquina secuencial	Código del estado presente de la máquina de estados del puerto serial.
Estados		
Nombre (abreviatura)	**Descripción y transiciones**	**Valor de señales de salida (las señales no explícitas en cada estado tienen valor cero)**
En espera (idle)	El estado de reposo en el que inicia la máquina de estados. Transiciones: **enPSn = 0**, no se ha activado el puerto serial, por lo que permanece en «idle». **enPSn = 1**, se ha recibido señal de inicio, por lo que se pasa a «data0»	**sPSn = 1**
Data 0 (data0)	Envío del bit menos significativo. Transición incondicional hacia «data1»	**enclkPSn = 1** **outPSn = Pcod [0]**
Data 1 (data1)	Envío del segundo bit menos significativo. Transiciones: **Cmpbits = 0**, $b_a > 2$ por lo que se pasa a «data2». **Cmpbits = 1**, $b_a = 2$, por lo que se pasa a «idle».	**enclkPSn = 1** **outPSn = Pcod [1]**
Data 2 (data2)	Envío del tercer bit menos significativo. Transiciones: **Cmpbits = 0**, $b_a > 3$ por lo que se pasa a «data3». **Cmpbits = 1**, $b_a = 3$, por lo que se pasa a «idle».	**enclkPSn = 1** **outPSn = Pcod [2]**
Data 3 (data3)	Envío del cuarto bit menos significativo. Transiciones: **Cmpbits = 0**, $b_a = 5$, por lo que se pasa a «data4». **Cmpbits = 1**, $b_a = 4$, por lo que se pasa a «idle».	**enclkPSn = 1** **outPSn = Pcod [3]**
Data 4 (data4)	Envío del quinto y último bit. Transición incondicional hacia «idle».	**enclkPSn = 1** **outPSn = Pcod [4]**

Se finaliza el diseño del puerto serial con su diagrama de bloques en la Figura 3-58. Obsérvese cómo está compuesto de la máquina secuencial y un comparador. El comparador sirve para detectar cuando el puerto ha finalizado el envío de bits, dados por la cantidad de bits del arreglo (b_a).

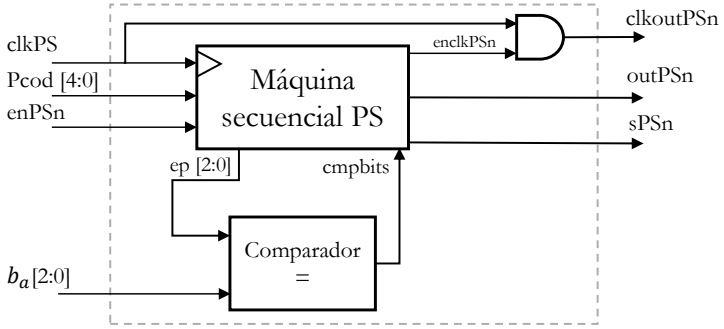


Figura 3-58. Diagrama de bloques del puerto serial.

3.7.5 Puerto de recepción de ángulos de orientación

Hasta el momento se había hablado de forma genérica sobre los ángulos de orientación, asumiéndolos como entradas directas de los bloques en la etapa de ejecución. No obstante, como se mencionó desde la propuesta de diseño, estos están pensados para provenir de algún otro sistema. También se comentó que, dada la velocidad de respuesta deseada, no es viable recibirlos por la misma interfaz de usuario (secc. 3.6), ya que ingresarlos de forma manual sería demasiado lento en comparación a la velocidad del sistema.

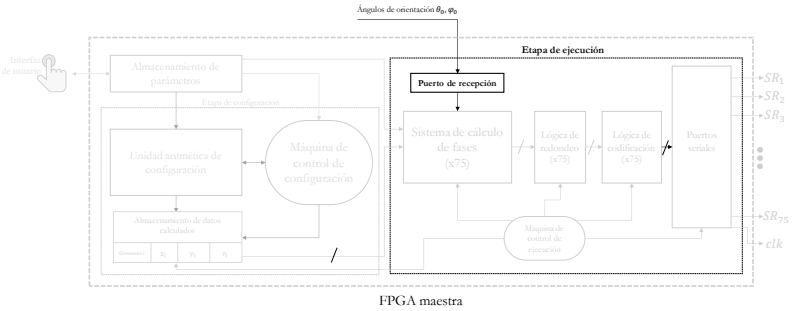


Figura 3-59. Puerto de recepción resaltado sobre la estructura interna.

Dicho lo anterior, el *puerto de recepción de ángulos de orientación*, que se encuentra resaltado en la Figura 3-59, sirve para recibir θ_0 y φ_0 por el protocolo SPI, mismo que permite una velocidad suficiente para recibirlos en un microsegundo. El puerto de recepción está compuesto de dos máquinas, una que funciona como receptor SPI y otra que controla el llenado de los registros de 24 bits para cada ángulo (8 bits para parte entera y 16 para parte fraccionaria). A continuación se explican ambas máquinas.

- Receptor SPI:

Este módulo no fue diseñado propiamente, sino que se tomó de un proyecto de acceso público disponible en «Github» [48]. Dado que la FPGA no envía ninguna información, únicamente funciona como receptor esclavo, necesitando solo 3 líneas de comunicación: **SPIclk**, **MOSI**, **SPIss**. El maestro debe estar configurado para enviar los datos a la FPGA en 8 bits a una tasa máxima de 40 Mbps.

- Máquina de recepción de ángulos de orientación:

Dado que el módulo SPI está diseñado para recibir datos de un byte, la máquina de recepción de ángulos de orientación funciona de forma algorítmica igual que la máquina de recepción de parámetros (secc. 3.6.3). De tal manera que se tienen seis registros de 8 bits, dos conjuntos de 3 concatenados para cada ángulo de orientación. Asimismo, se tiene la bandera **sAO** para indicar cuando se ha terminado de recibir un nuevo par de ángulos y comenzar la etapa de ejecución. La máquina secuencial es controlada por el reloj **clkSPI**. Para entender con detalle el funcionamiento se puede consultar la sección antes mencionada.

Se finaliza el diseño del puerto de recepción de ángulos de orientación con el respetivo diagrama de bloques, que puede consultarse en la Figura 3-60.

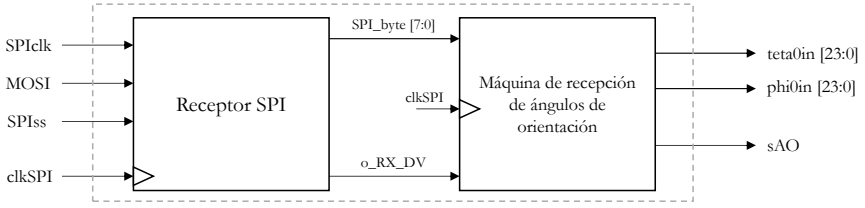


Figura 3-60. Diagrama de bloques del puerto de recepción de ángulos de orientación.

3.7.6 Primer análisis de latencias

En esta sección se analizan las primeras estimaciones de retardos en los bloques respectivos al núcleo. Los tiempos de propagación fueron obtenidos con ayuda del propio Vivado: si se tiene el diseño ya implementado, el software permite ver la latencia del camino crtítico de los módulos, considerando ya el ruteo dentro de la FPGA a utilizar. Las estimaciones demostraron que en la mayoría de bloques se inclumplía la restricción (3.a). Dentro de la Tabla 3-12 se dan algunos detalles e información extra relacionada a las latencias.

Tabla 3-12. Resumen del primer análisis de latencias.

Bloque	Camino crítico	Latencia	Observaciones
Unidad de cálculo de fases	El camino crítico de esta unidad está compuesto de un multiplicador de 18 b, dos multiplicadores de 32 b y dos restadores de 32 bits (Figura 3-48).	178.57 ns	La mayoría del tiempo de propagación se consume en la lógica correspondiente a los multiplicadores, aun cuando estos se han ajustado a los tamaños de los multiplicandos.
Submódulo de equivalencia	Si se cumple la condición de que la magnitud de la fase equivalente es mayor a 2π, es necesario «tomar el camino» que pasa por las operaciones matemáticas posteriores y la lógica de equivalencia. El camino crítico está compuesto por la lógica de transformación (complemento 2), dos multiplicadores de 32 bits, y la lógica de equivalencia (Figura 3-50).	49.283 ns	Aunque las multiplicaciones en este caso se hacen con valores fijos, lo que hace en consecuencia que se reduzca la lógica combinacional, también se podría decir que el optimizar el multiplicador permitiría llegar el tiempo límite de respuesta.
Submódulo de redondeo y codificación	En este módulo se tiene siempre un solo camino: consiste en el multiplicador de 19 bits y el sumador de la lógica de redondeo y codificación (Figura 3-53).	32.683 ns	En este bloque se cumple con el tiempo límite de respuesta establecido. De cualquier forma, si se optimiza el multiplicador, también se tendrá un tiempo de respuesta aún menor.

La conclusión de este primer análisis de tiempos indica que la optimización de los módulos multiplicadores resulta clave para disminuir los retardos en los tres bloques. Asimismo se concluye que, de las dos alternativas posibles (discutidas en secc. 3.7.1), la segunda sigue siendo inviable de implementar.

3.7.7 Rediseño del módulo multiplicador

Existe una gran variedad de trabajos relacionados a la implementación de multiplicadores. Este bloque aritmético es de gran interés para el estudio de las arquitecturas de los procesadores y más especialmente de los procesadores de señales digitales (DSP), ya que es la operación aritmética más importante para diversos algoritmos de procesamiento [49]. A pesar de esta gran variedad y las distintas propuestas, la mayoría de los diseños que se proponen en los trabajos utilizan una estructura secuencial que, aunque quizás podrían ser más rápidos, las etapas secuenciales se tendrían que añadir a las etapas del encauzamiento y esto podría terminar resultando perjudicial, ya que se busca tener un equilibrio adecuado entre el número de etapas para que no se exceda el tiempo que se tarde en obtener la primera fase codificada.

Sin embargo, se encontraron un par de diseños que manejan lógica puramente combinacional [50] [51]. Ambos diseños fueron probados en 8 bits para verificar cuál de ellos resultaba ser más eficiente y si se tenía un tiempo de respuesta menor al que del multiplicador original. El diseño presentado en [51] fue el que menor tiempo de respuesta tuvo, pero dada la inconsistencia y pobre manera en la que se encuentra explicado el diseño, no fue posible extrapolar exitosamente el concepto para implementar el multiplicador en 32 bits.

No obstante, la representación que hacen de los números a multiplicar resultó clave para el diseño de un multiplicador propio (basado en su representación), que a continuación se explica. Los autores del artículo [51] utilizan la representación alternativa de los valores de *a* y *b*, que se consideran el par de valores a multiplicar. Siendo a y b números de 8 bits, se tiene que:

$$\boldsymbol{a = a_e + a_o}, \qquad \boldsymbol{b = b_e + b_o}$$

$$a_e = \{0,\ a[6],\ 0,\ a[4],\ 0,\ a[2],\ 0,\ a[0]\}, \qquad b_e = \{0,\ b[6],\ 0,\ b[4],\ 0,\ b[2],\ 0,\ b[0]\}$$

$$a_o = \{a[7],\ 0,\ a[5],\ 0,\ a[3],\ 0,\ a[1],\ 0\}, \qquad b_o = \{b[7],\ 0,\ b[5],\ 0,\ b[3],\ 0,\ b[1],\ 0\}$$

Con la nueva definición de a y b se pueden modificar la multiplicación como se muestra a continuación:

$$a \times b = (a_e + a_o)(b_e + b_o) = a_e b_e + a_e b_o + a_o b_e + a_o b_o$$

Se tienen entonces cuatro subproductos que deben ser sumados para obtener el resultado final. El diagrama de bloques del multiplicador con la notación original se muestra en la Figura 3-61. Se puede observar que el retardo está compuesto por la etapa de compuertas AND y 4 sumadores en cascada. Además, se necesita un total de 15 sumadores para obtener el producto. Obsérvese cómo en la figura se obtiene el producto parcial «$a_e b_e$» mediante la suma de los subproductos $a_e b[n]$, donde *n* para b_e vale 0, 2, 4 y 6.

Los subproductos, al ser independientes, se pueden obtener de forma paralela, y para sumarse se pueden agrupar en parejas, siempre tomando en cuenta los corrimientos hacia la izquierda que se realizan en cada subproducto. Este procedimiento aplica para el resto de productos parciales. Al final se deben sumar todos los resultados para obtener el valor de la multiplicación, estas sumas también se realizan en parejas.

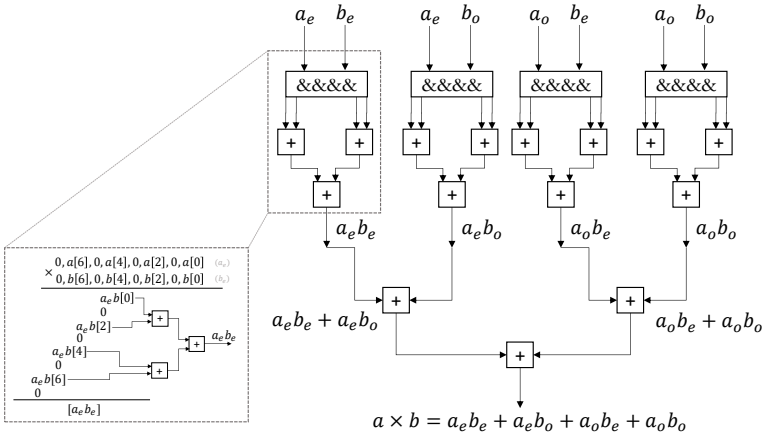


Figura 3-61. Multiplicador alternativo con notación original.

Ahora se explicará la notación alternativa propuesta. Partiendo de la misma representación utilizada por los autores, el producto queda definido como:

$$a \times b = (a_e + a_o)(b_e + b_o) = a_e b_e + a_e b_o + a_o b_e + a_o b_o$$

$$factorizando\ a_e\ y\ a_o \rightarrow a_e(b_e + b_o) + a_o(b_e + b_o) = \boldsymbol{a_e(b) + a_o(b)}$$

Es decir, la multiplicación ahora queda definida por únicamente dos productos parciales, que tendrán que ser sumados para obtener la multiplicación final. El diagrama de bloques de la alternativa propuesta se encuentra en la Figura 3-62, pudiendo notar que ahora el retardo está compuesto por la etapa de compuertas AND y 3 sumadores; además, ahora se necesitan solo 7 sumadores en total para obtener el producto.

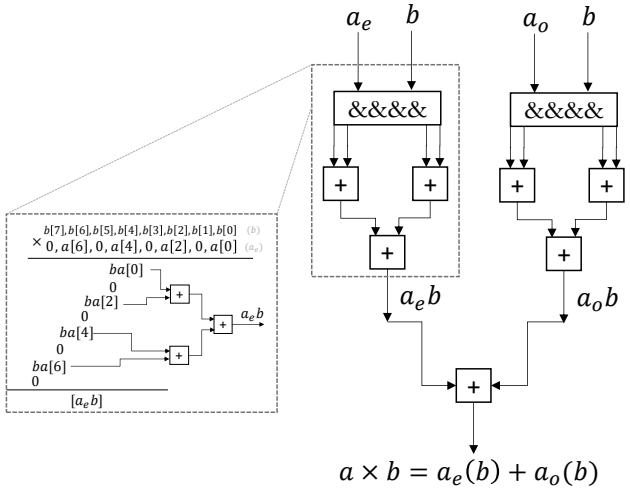


Figura 3-62. Multiplicador alternativo con notación nueva propuesta.

Con la notación propuesta, además de reducir la latencia del módulo multiplicador, se están utilizando la mitad de recursos que en la notación original de los autores del artículo. Para tener un tiempo de respuesta aún menor, los sumadores que se usarán para la adición de subproductos son de tipo «Carry Look-Ahead» (CLA) [52], los que tienen la característica de reducir el tiempo de respuesta en comparación de los sumadores tradicionales, cuya latencia se incrementa proporcionalmente al número de bits, debido a la propagación del acarreo de salida con el de entrada del bit siguiente (como en la Figura 2-22). Los sumadores CLA calculan de forma independiente los acarreos, generando así una especie de paralelismo a cambio de un uso de recursos mayor en comparación. Los sumadores CLA fueron descritos en estilo RTL.

En la Figura 3-63 se muestra la estructura del multiplicador alternativo ya extrapolado para 32 bits, ya que este es el número en el que normalmente se trabajarán los valores salvo algunas excepciones. Cabe recordar que la multiplicación es sensible a los signos de los valores de entrada, aun cuando estos estén representados en complemento 2, por lo que el multiplicador está hecho para trabajar siempre con valores positivos y el signo es tratado de forma externa, tal como ocurrió en el diseño del módulo original (secc. 3.5.1.1). Se tiene una latencia total compuesta por la etapa de compuertas AND y 5 sumadores en cascada, que en la descripción hecha en SV se tradujo a tan solo 12.9 ns, cuando en el diseño del multiplicador original se tenía una latencia que superaba por poco los 45 ns.

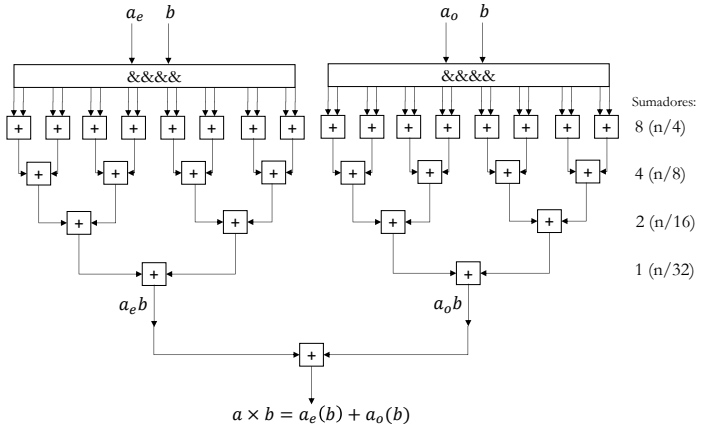


Figura 3-63. Multiplicador alternativo de 32x32 b con notación propuesta.

3.7.8 Rediseño de la unidad de cálculo de fases

Con únicamente sustituir los multiplicadores tradicionales por los alternativos que se diseñaron en la sección anterior, la unidad de cálculo de fases pasó de un retardo de 178.57 ns a 63.22 ns, es decir, aproximadamente un tercio del tiempo que originalmente se requería para el cálculo. No obstante, este retardo sigue siendo mucho mayor que el límite que se tiene planteado de 33 ns (restricción (3.a)). Una alternativa para respetar la condición es subdividir la unidad de cálculo en dos partes, de tal manera que se distribuya la mitad del tiempo en cada una.

En vez de optar por la subdivisión, se decidió buscar la forma de simplificar la operación matemática (2.b), de tal manera que fuera menos compleja computacionalmente, para conseguir una reducción mayor en el tiempo de ejecución y deseablemente en los recursos utilizados. Este planteamiento llevó a una segunda implementación de la unidad, la cual se describe a continuación.

Partiendo de la expresión (2.b), se sustituyeron dentro de esta las expresiones (2.d) y (2.e) (que se utilizan para obtener las coordenadas en los ejes x y y del elemento en cuestión). La sustitución resulta en la expresión (3.c) que a continuación se presenta y de la que posteriormente se comentarán varias cosas.

$$P_{desp_i} = 2\pi\left[\frac{r_i}{\lambda} - \left(\boldsymbol{n_i}\underbrace{\frac{d\cos(\alpha)}{\lambda}}_{\boldsymbol{c}}\cos(\varphi_0)sen(\theta_0) + \boldsymbol{m_i}\underbrace{\frac{B}{\lambda}}_{P_{coef_i}}\cos(\varphi_0)sen(\theta_0)\right) - \boldsymbol{n_i}\underbrace{\frac{d}{\lambda}sen(\alpha)}_{\boldsymbol{h}}sen(\varphi_0)sen(\theta_0)\right] \quad (3.c)$$

($\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{f}$, $\boldsymbol{g}$)

La primera observación es sobre los valores nombrados como **a**, **b** y **g**: estos son constantes para la etapa de ejecución, ya que no están en función de los datos del elemento ni de los ángulos de orientación. Son valores que dependen únicamente de la estructura del arreglo, por lo que pueden ser calculados de manera previa y estar disponibles de forma inmediata para la etapa de ejecución. Por lo tanto, la obtención de estos valores quedará fuera de la unidad aritmética, serán señales constantes de entrada hacia esta.

La segunda consiste en los productos nombrados como **c**, **f** y **h**. Estos productos no involucran a los índices de fila ni de columna ($\boldsymbol{m_i}$ y $\boldsymbol{n_i}$) del elemento en cuestión, sino que únicamente dependen de los ángulos de orientación y de las constantes **a**, **b** y **g**. Esto significa que no tienen que ser recalculados para cada elemento del arreglo, sino que sólo hace falta obtenerlos una única vez para cada par de ángulos de orientación. En otras palabras, la obtención de estos productos se puede hacer mediante multiplicadores que sean únicos para todo el sistema, ya que serán datos que se compartirán entre todos los elementos. Con esto se consigue un ahorro importante de recursos, debido a que los ***núcleos***[9] ahora contendrán menos multiplicadores.

Con esta nueva implementación de la unidad de cálculo, la operación matemática (2.b) se vuelve bastante más simple computacionalmente. Ahora, según la expresión definida como (3.c), sólo será necesario realizar los productos $(c \times n_i)$, $(f \times m_i)$ y $(h \times n_i)$, para posteriormente sumarlos con el valor de r_i/λ y así obtener el valor del desfasamiento requerido en cada elemento.

La Figura 3-64 muestra la estructura redefinida de la unidad artimética compuesta por dos principales bloques: el que es único para todo el sistema (simbolizado $\times_{\boldsymbol{u}}$) y el bloque del que se tendrá uno por núcleo (llamada ahora como unidad aritmética 'UA'). Nótese cómo el valor de $sen(\theta_0)$ ingresa negativo a los multiplicadores, esto es para introducir por adelantado el signo negativo a los productos **c**, **f** y **h**, y al final únicamente tener que sumarlos.

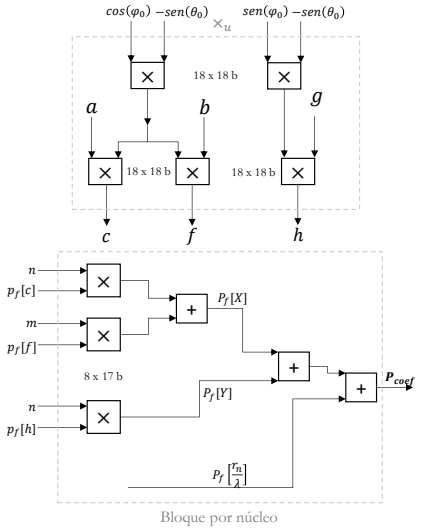


Figura 3-64. Diagrama de la unidad de cálculo de fases rediseñada.

Con respecto al primer valor (r_i/λ) presente en (3.c), no es necesario realizar este cálculo en esta etapa, sino que se puede optar por almacenar el resultado de forma directa desde la etapa de configuración (secc. 3.5), de tal manera que este valor ya se tenga de forma directa como dato almacenado en la memoria de elementos (diseñada en secc. 3.5.2). Por lo tanto, la memoria de elementos pasa a simplificarse, en vista de que ya no será necesario almacenar los datos de posición en x y y, sino que únicamente se requerirán los índices de fila y columna ($\boldsymbol{n_i}$, $\boldsymbol{m_i}$) y la parte fraccionaria de r_i/λ, lo que conlleva a una reducción en el espacio requerido para el almacenamiento de datos (más detalles en sección posterior 3.7.13).

La última observación importante que hay que hacer de la expresión (3.c) es que el resultado de la operación entre corchetes es un factor que se multiplica por 2π para obtener el valor en radianes. Dada la periodicidad de la circunferencia, todo factor mayor a 1 multiplicado por 2π tiene un equivalente, que es un ángulo ubicado entre 0 y 2π. Por ejemplo:

[9] Se usará la denominación alternativa «núcleo» para referirse a cada etapa de ejecución completa con sus respectivos módulos.

$$(1.482)2\pi = 9.3116\,rad = 533.515°$$

$$P_{coef} = p_f[1.482] = 0.482$$

$$Equivalente: P_{coef}2\pi = (0.482)2\pi = 3.0284\,rad = 173.515°$$

El valor de 533.515° del ejemplo anterior no es más que 173.515° desplazado 360°. Esto conlleva a que no sea necesario calcular el coeficiente de forma completa con parte entera y fraccionaria, sino que únicamente haga falta el cálculo de la parte fraccionaria como se hizo en el ejemplo previo. De tal forma se asegura que el valor del coeficiente siempre se encuentre entre (-1, 1), lo que implica que la fase siempre estará entre (-360°, 360°). Con tal modificación se reduce el número de bits requeridos en las señales intermedias, necesitando solo 17 bits para representar y almacenar los números (16 de la parte fraccionaria y uno extra para poder representar en C2). Esto también conlleva a hacer cambios en los submódulos de equivalencia, redondeo y codificación que serán explicados más adelante.

3.7.9 Rediseño de la unidad de equivalencia

Con las modificaciones hechas en la sección anterior, debido a reestructuración de la unidad de cálculo de fases, la unidad de equivalencia (UEq) ya no recibe los mismos valores que en la implementación original. En esta nueva estructura, la unidad aritmética entrega a la salida un valor denominado P_{coef} (coeficiente de fase), el cual siempre estará entre (-1, 1). La función previa de la unidad de equivalencia era convertir la fase cuya magnitud fuera mayor a 360° a una que estuviera entre 0° y 360° (diseñada en secc. 3.7.3), pero debido a que ya no se recibe un valor de fase como tal, y que este ya está truncado entre -1 y 1, la nueva función de esta unidad deberá convertir P_{coef} si su valor es negativo a su equivalente en positivo. En otras palabras, la UEq deberá hacer la operación $P_{ceq} = 1 + P_{coef}$, o conservar el valor intacto si es que originalmente ya es positivo.

La estructura rediseñada de la UEq se muestra en la Figura 3-65. El sumador utilizado para la conversión al equivalente positivo es también de tipo CLA.

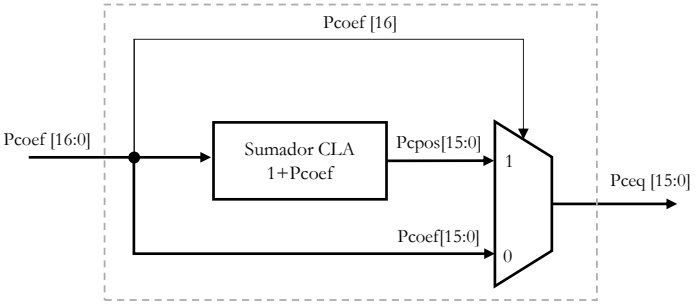


Figura 3-65. Diagrama de la unidad de equivalencia rediseñada.

3.7.10 Rediseño de la unidad de redondeo y codificación

De la expresión (3.c) anterior se puede observar que el valor de k (definida en (2.a)) fue sustituido por $2\pi/\lambda$, introduciendo el valor de $\mathbf{1/\lambda}$ a los productos internos constantes (**a**, **b** y **g**), pero manteniendo el 2π como factor común de toda la expresión (3.c). Por otro lado, si se revisa brevemente el diseño original de la etapa de redondeo y codificación (secc. 3.7.3), es de notarse que esta consiste en la división del valor de la fase sobre $2\pi/2^{b_a}$, es decir, la multiplicación por 2π hecha en (3.c) posteriormente es anulada por la división de la etapa de redondeo. Por tanto, esto resulta en ya no tener que realizar la multiplicación y la eliminación del factor 2π en la etapa de redondeo, convirtiéndose en una división más simple, misma que puede ser hecha con corrimientos que dependerán del número de bits b_a.

Por ende, la unidad de redondeo ahora únicamente realiza una división entre $1/2^{b_a}$, que en consecuencia termina siendo una multiplicación por 2^{b_a}. Recordando que se está trabajando con formato de punto fijo, este producto se puede realizar con un simple corrimiento hacia la izquierda b_a posiciones, obteniendo la señal nombrada P_{eqs}.

El realizar este cambio simplifica de forma importante esta unidad, ya que pasa de tener una multiplicación aritmética a un simple corrimiento que depende del número de bits del arreglo. La etapa posterior de redondeo-codificación se sigue

realizando de la misma forma, mediante la suma de la parte entera de P_{eqs} con el primer bit de la parte fraccionaria (para más detalles se puede consultar la sección dedicada 3.7.3). En la Figura 3-66 se muestra el diagrama reestructurado.

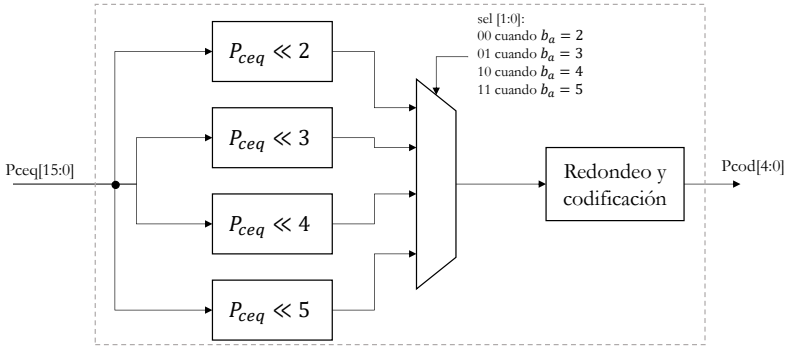


Figura 3-66. Diagrama de la unidad de redondeo y codificación rediseñada.

3.7.11 Segundo análisis de latencias

Habiendo planteado la reestructuración de las tres etapas de las que se compone el núcleo de ejecución (unidad aritmética, unidad de equivalencia y unidad de redondeo y codificación), lo siguiente fue describirlas formalmente en HDL, con el fin de analizar las nuevas latencias que se presentan en cada etapa y corroborar si estas ya satisfacen el límite de tiempo establecido en (3.a). Obsérvese la Tabla 3-13 siguiente.

Tabla 3-13. Resumen del segundo análisis de latencias.

Bloque	Latencia	Observaciones
Multiplicador único	25.733 ns	Se respeta el tiempo límite de 33 ns.
Unidad aritmética	20.2 ns	Se respeta el tiempo límite de 33 ns.
Submódulo de equivalencia	9.688 ns	Se respeta el tiempo límite de 33 ns.
Submódulo de redondeo y codificación	5.114 ns	Se respeta el tiempo límite de 33 ns.

Analizando las nuevas latencias presentadas en las tres etapas del núcleo, ahora es posible implementar la etapa de ejecución como se tenía pensado originalmente, esto es, teniendo registros al final de cada etapa funcionando a un reloj con un periodo de 33 ns. No obstante, gracias a la reducción considerable de retardos, se puede incrementar aún más la frecuencia de operación. Para tal propósito se necesita una mejor distribución de las latencias y por ende se requieren reajustar las ubicaciones de los registros que dividen las etapas del encauzamiento. La estructura definitiva se muestra en la Figura 3-67 y posterior a ella se encuentran los respectivos análisis.

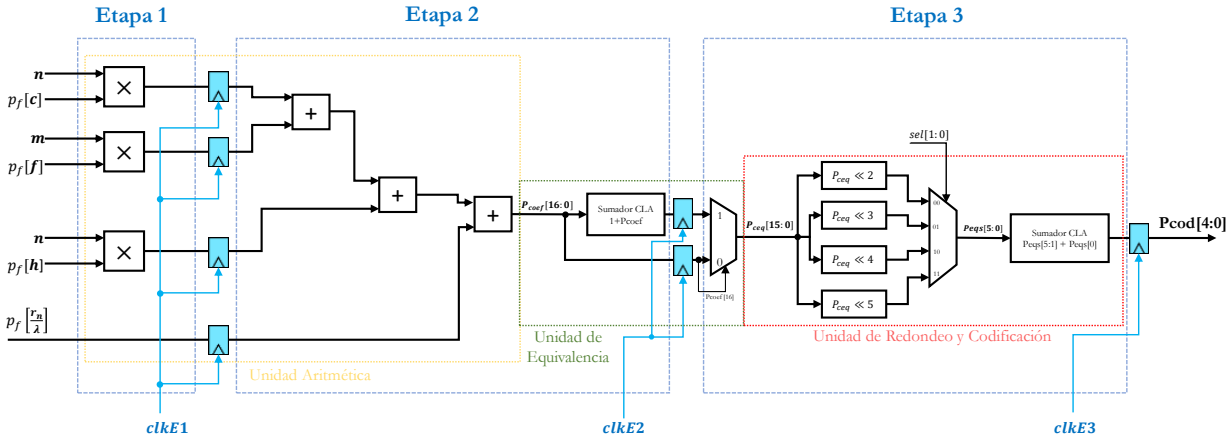


Figura 3-67. Diagrama de bloques de las tres etapas del núcleo de ejecución. Se muestran los segmentos del «pipelining» y las unidades lógicas.

Tabla 3-14. Descripción y latencias de etapas del encauzamiento.

Etapa	Latencia	Comentarios
1	12.5 ns	Compuesta de los primeros multiplicadores de la unidad aritmética.
2	10.21 ns	Los tres sumadores de la unidad aritmética y el sumador de la unidad de equivalencia.
3	5.312	Compuesta del multiplexor a la salida de la unidad de equivalencia y la unidad de redondeo y codificación.

Con la reestructuración de la Tabla 3-14 anterior, el núcleo de ejecución ahora puede trabajar a una frecuencia máxima de 66.666 MHz (considerando la latencia más grande de 12.5 ns y reservando un margen de 2.5 ns). Las unidades seguirán siendo las mismas que como se han mencionado hasta el momento, simplemente estarán subdivididas por los registros tal como se muestra en el diagrama anterior.

3.7.12 Reestructuración del núcleo de ejecución

Un análisis respecto a los puertos seriales (que se explica más adelante dentro de este mismo apartado) conllevó a una reestructuración en el núcleo de ejecución. Primero hay que recordar que se piensa trabajar con el envío de datos de forma serial a una frecuencia de 150 MHz o Mbps, la cual permite, en el caso del mayor número de señales de control, satisfacer el tiempo de respuesta deseado.

La deterioración de las señales eléctricas que contienen los datos y el reloj a tales frecuencias es más notoria incluso en distancias cortas, por lo que se tendrá que analizar si no se sufren estos efectos en la implementación física de los dispositivos esclavos y la FPGA maestra. En caso de las señales a la frecuencia deseada sufran un gran deterioro en la implementación física, se buscará optar por alguna alternativa para compensar la disminución forzosa en la tasa de transferencia.

De momento se hablará de una frecuencia de 142.856 MHz, con el fin de tener un valor entero de periodo (7 ns). Con tal frecuencia de operación, los puertos seriales necesitarían de dos a cinco ciclos de reloj para el envío de cada fase codificada (reqs. en 3.1), que traducido a tiempo va de 14 ns hasta 35 ns. Asimismo, la frecuencia a la que estará trabajando el núcleo de ejecución tendrá un periodo de 14 ns[10].

En el caso de $b_a = 5$, el núcleo de ejecución termina siendo de alguna manera «subutilizado», ya que debe esperar a que su respectivo puerto serial termine el envío de cada fase codificada. Con base en este caso se planteó una reestructuración alternativa del núcleo de ejecución, buscando aprovechar mejor la segmentación encauzada y procurar no tener que detener el funcionamiento del núcleo. Esta reestructuración se describe en la presente sección.

La nueva estructura tiene como propósito sacar mejor provecho del núcleo, utilizando el tiempo que tendría que esperar a que el puerto serial termine el envío de datos, para seguir realizando cálculos de elementos pertenecientes a otros esclavos. En la Figura 3-68 el núcleo tiene asignados tres puertos seriales en vez de uno solo.

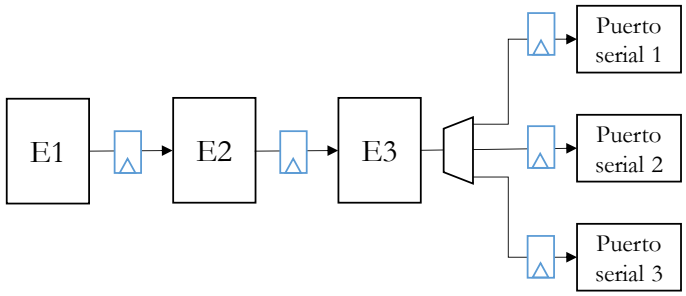


Figura 3-68. Reestructuración del núcleo de ejecución.

Para ilustrar de una manera más sencilla el flujo de los datos dentro del núcleo y de los puertos seriales se adjunta la Tabla 3-15, que muestra los eventos que ocurren en cada ciclo de reloj de la máquina de control de ejecución. Se considera un

[10] Dadas estas condiciones, el núcleo de ejecución siempre trabajará a la mitad de frecuencia que los puertos seriales **clkEjec = clkPS/2**.

arreglo de 5 bits y las frecuencias de operación previamente mencionadas, recalcando que los puertos seriales funcionan al doble de frecuencia que el núcleo de ejecución**.**

Tabla 3-15. Flujo de datos en el núcleo de ejecución para un arreglo de 5 bits. La notación Ex_y indica que los datos del elemento y asignado al esclavo x están siendo procesados en la etapa indicada.

Periodo	Etapa 1 Unidad aritmética P1	Etapa 2 Unidad aritmética P2	Etapa 3 Unidades E, R y C	Puerto serial 1	Puerto serial 2	Puerto serial 3
T1	$E1_1$	//	//	//	//	//
T2	$E2_1$	$E1_1$	//	//	//	//
T3	$E3_1$	$E2_1$	$E1_1$	//	//	//
T4	$E1_2$	$E3_1$	$E2_1$	$E1_1[0]$	//	//
				$E1_1[1]$	//	//
T5	$E2_2$	$E1_2$	$E3_1$	$E1_1[2]$	$E2_1[0]$	//
				$E1_1[3]$	$E2_1[1]$	//
T6	$E3_2$	$E2_2$	$E1_2$	$E1_1[4]$	$E2_1[2]$	$E3_1[0]$
				//	$E2_1[3]$	$E3_1[1]$
T7	$E1_3$	$E3_2$	$E2_2$	$E1_2[0]$	$E2_1[4]$	$E3_1[2]$
				$E1_2[1]$	//	$E3_1[3]$
T8	$E2_3$	$E1_3$	$E3_2$	$E1_2[2]$	$E2_2[0]$	$E3_1[4]$
				$E1_2[3]$	$E2_2[1]$	//
T9	$E3_3$	$E2_3$	$E1_3$	$E1_2[4]$	$E2_2[2]$	$E3_2[0]$
				//	$E2_2[3]$	$E3_2[1]$
T10	$E1_4$	$E3_3$	$E2_3$	$E1_3[0]$	$E2_2[4]$	$E3_2[2]$
				$E1_3[1]$	//	$E3_2[3]$
T11	$E2_4$	$E1_4$	$E3_3$	$E1_3[2]$	$E2_3[0]$	$E3_2[4]$
				$E1_3[3]$	$E2_3[1]$	//

De la tabla anterior se puede observar cómo el núcleo de ejecución se encuentra todo el tiempo trabajando, sin detenerse a esperar a que el puerto serial finalice el envío de datos. En este caso, ahora es cada puerto el que se detiene un ciclo de reloj a esperar a que esté lista su siguiente fase codificada, lo que se puede ver como si cada dato tomara 5+1 ciclos de reloj en enviarse. Calculando el tiempo para distribuir las fases con un reloj de 7 ns de periodo y 20 elementos por cada esclavo, se obtiene lo siguiente: $t_{sr} = (20)(5+1)(7\ ns) = 840\ ns$.

Para esta alternativa, en general se hace un mejor uso de los recursos disponibles, además de reducir a un tercio la cantidad de núcleos de ejecución necesaria para el total de dispositivos esclavos. Aunque en principio todo suena como ventajas, se hará el análisis de esta alternativa para el caso en que el arreglo es de 2 bits, obsérvese la Tabla 3-16.

Tabla 3-16. Flujo de datos en el núcleo de ejecución para un arreglo de 2 bits. La notación Ex_y indica que los datos del elemento y asignado al esclavo x están siendo procesados en la etapa indicada.

Periodo	Etapa 1 Unidad aritmética P1	Etapa 2 Unidad aritmética P2	Etapa 3 Unidades E, R y C	Puerto serial 1	Puerto serial 2	Puerto serial 3
T1	$E1_1$	//	//	//	//	//
T2	$E2_1$	$E1_1$	//	//	//	//
T3	$E3_1$	$E2_1$	$E1_1$	//	//	//
T4	$E1_2$	$E3_1$	$E2_1$	$E1_1[0]$	//	//
				$E1_1[1]$	//	//
T5	$E2_2$	$E1_2$	$E3_1$	//	$E2_1[0]$	//
				//	$E2_1[1]$	//
T6	$E3_2$	$E2_2$	$E1_2$	//	//	$E3_1[0]$
				//	//	$E3_1[1]$
T7	$E1_3$	$E3_2$	$E2_2$	$E1_2[0]$	//	//
				$E1_2[1]$	//	//
T8	$E2_3$	$E1_3$	$E3_2$	//	$E2_2[0]$	//
				//	$E2_2[1]$	//
T9	$E3_3$	$E2_3$	$E1_3$	//	//	$E3_2[0]$
				//	//	$E3_2[1]$
T10	$E1_4$	$E3_3$	$E2_3$	$E1_3[0]$	//	//
				$E1_3[1]$	//	//
T11	$E2_4$	$E1_4$	$E3_3$	//	$E2_3[0]$	//
				//	$E2_3[1]$	//

En este caso (mostrado en Tabla 3-16) es posible notar que, aunque el núcleo sigue funcionando en todo momento, los puertos seriales se ven en la necesidad de esperar 4 ciclos de reloj a que esté lista su siguiente fase, lo que también se puede ver como si cada fase requiriera de 2+4 ciclos de reloj para enviarse. En resumen, se llega a la misma situación que ocurrió en el caso de $b_a = 5$ comentado anteriormente. En conclusión, con la reestructuración de la Figura 3-68 el tiempo de distribución de las fases codificadas (t_{sr}) ya no depende el número de bits b_a, sino que se convierte en un valor constante definido por la siguiente expresión (3.d).

$$t_{sr} = 6\ e_{es}\ T_{sr} \tag{3.d}$$

Donde e_{es} es el número de elementos que controla cada esclavo, y T_{sr} es el periodo de la frecuencia a la que operan los puertos seriales/registros de corrimiento.

Con los análisis hechos anteriormente se llega al siguiente cuestionamiento: «¿Qué se prefiere?»: tener un t_{sr} que esté en proporción de b_a a cambio tener un núcleo por cada esclavo; o tener un tiempo de distribución igual al t_{sr} del caso más grande (cuando $b_a = 5$), que permanezca constante para todo valor de b_a, a cambio de utilizar **un tercio** del total de núcleos, ahorrando el equivalente en recursos lógicos.

Después de discutir esta situación con los tutores del trabajo, se ha decidido optar por la segunda opción debido a que a este sistema se busca implementar en un satélite tipo «CubeSat»[11], en donde un ahorro de recursos lógicos implica una disminución en consumo energético; y es que la energía eléctrica es un recurso valioso y escaso en satélites de tal tipo.

Finalmente, ya que se ha decidido la estructura que tendrá el núcleo de ejecución y los módulos internos que lo componen, se puede diseñar la máquina que controlará el flujo de datos de esta etapa, a este circuito secuencial se le nombrará «Máquina de control de ejecución» y se diseñará más adelante. A continuación se hablará sobre la reestructuración que se tuvo que hacer en la etapa de configuración con base en las modificaciones hasta ahora hechas en la etapa de ejecución.

3.7.13 Reestructuración de la etapa de configuración

Dada la reestructuración que ha sufrido la etapa de ejecución (detallada en las secciones previas 3.7.7 – 3.7.12), resulta necesario modificar ciertas cosas de la etapa de configuración (diseñada originalmente en la secc. 3.5). Los cambios están relacionados a la manera en que se llenan los datos en las memorias y qué datos se almacenan. Como una breve recapitulación, se menciona que originalmente se almacenaban los datos de n_{es} elementos concatenados como una sola localidad conjunta de memoria, teniendo un total de e_{es} localidades; los datos eran la posición en x, posición en y y distancia al alimentador (x_i, y_i, r_i; sec. 2.1.3) representados en números de 32 bits, lo que resultaba en **96 bits** de información por cada elemento.

Ahora se almacenan los índices de fila y de columna (m_i y n_i, dados en 8 bits), además de la parte fraccionaria de r_i/λ (en 16 bits), necesitando un total de **32 bits** por cada elemento (un tercio de la cantidad original). Además, ya no se necesita guardar la información de n_{es} elementos en una localidad, sino solo de $n_{es}/3$ elementos (debido a que ahora solo se tiene un núcleo por cada tres puertos seriales), pero con un total de $3e_{es}$ localidades (el triple de localidades que originalmente). Por tanto, la reestructuración de la etapa de configuiración modificó a los siguientes módulos:

- Búfers temporales: se convertirán en registros de 32 bits, teniendo un total de $n_{es}/3$ para enviar su concatenación a una localidad de memoria cuando todos hayan sido llenados (detalles en secc. 3.5.2).
- Memoria de elementos: pasará a ser una única memoria con $3e_{es}$ localidades, cada una de $\frac{32}{3}n_{es}$ bits.
- Unidad aritmética de configuración: se añadirá la obtención de $r_i \times \frac{1}{\lambda}$ y se mantendrá el resto de bloques, ya que todos los resultados siguen siendo necesarios para el funcionamiento del algoritmo aunque ya no se almacenen en la memoria.
- Máquina de control de configuración: se modificará el funcionamiento de la máquina para almacenar los nuevos datos, según la reestructura aquí mencionada.

Se finalizan las modificaciones hechas a la etapa de configuración con el nuevo diagrama de bloques con las respectivas nuevas estructuras (Figura 3-69). Los multiplicadores dentro de la unidad aritmética de configuración también fueron sustituidos por los nuevos diseñados en la sección 3.7.7.

[11] Los «CubeSat» son un estándar de nanosatélites con unidades base de 10 cm³. La popularización de estos se ha incrementado debido a su relativo bajo costo de desarrollo y fabricación en relación a los satélites de mayor tamaño [57].

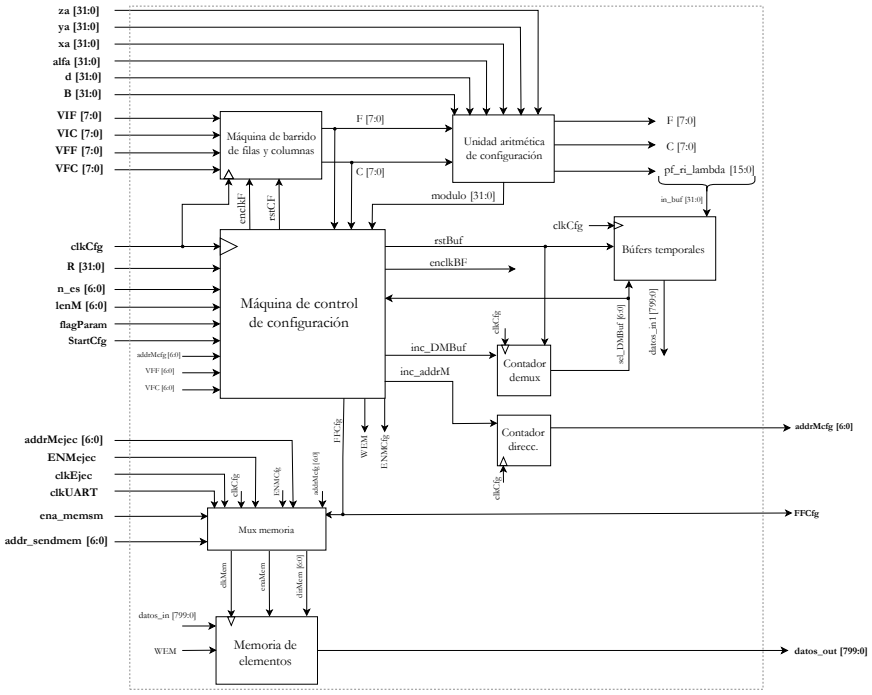


Figura 3-69. Interconexión de módulos para la etapa de configuración reestructurada.

3.7.14 Máquina de control de ejecución

Esta máquina, al igual que la máquina de control de configuración (secc. 3.5.3), tiene como propósito el controlar la segmentación encauzada y el envío de datos hacia los registros de corrimiento. Se comenzará el diseño de este circuito con una descripción de las tareas generales que deberá realizar:

a) Controlar la lectura de la memoria de elementos, para tener en cada momento los datos respectivos de $n_{es}/3$ elementos a la vez.
b) Activar el registro de la primera etapa (**clkE1**) después de que se han realizado los productos correspondientes a la primera parte de la unidad aritmética.
c) Activar el registro de la segunda etapa (**clkE2**) después de que se han obtenido las sumas correspondientes a la segunda parte de la unidad aritmética y de la unidad de equivalencia.
d) Activar el reloj del registro del puerto serial (**clkPSn**) una vez que se han obtenido la fase codificada correspondiente a este.
e) Dar la señal de inicio a los puertos seriales para que envíen los bits necesarios de las fases codificadas hacia los dispositivos esclavos.
f) Repetir el proceso las veces que sean necesarias hasta completar las e_{es} fases de cada dispositivo esclavo, y finalizar activando la señal de reloj a los registros que estarán ubicados a las salidas de los decodificadores de los dispositivos esclavos, repolarizando todos los elementos al mismo tiempo y así evitando «glitches».

En la Figura 3-70 se encuentra el diagrama de estados simplificado de esta máquina, mientras que en la Tabla 3-17 se explica el significado de las señales de entrada y salida, así como la descripción de los estados de los que se compone.

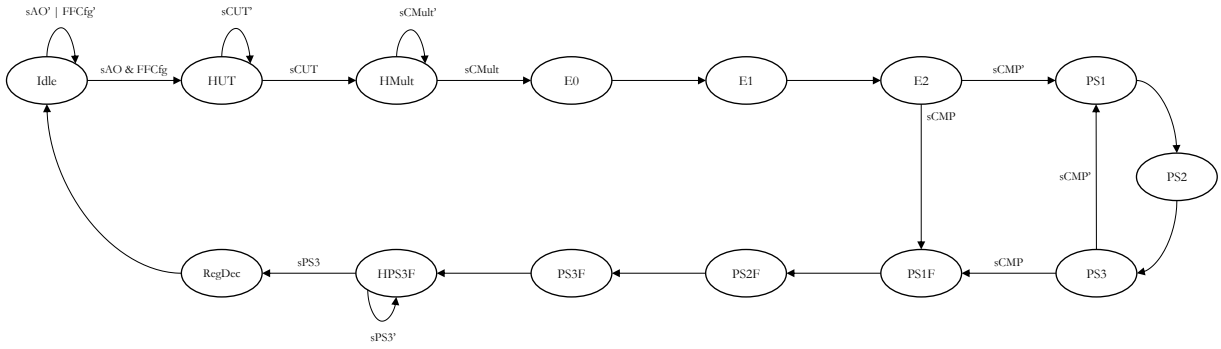


Figura 3-70. Diagrama de estados de la máquina de control de ejecución.

Tabla 3-17. Descripción de señales y estados de la máquina de control de ejecución.

Señales		
Tipo	**Nombre**	**Descripción**
Entrada	**sAO**: señal de ángulos de orientación	Esta señal es salida del puerto de recepción, sirve para indicar que se han terminado de recibir los ángulos de orientación.
	FFCfg: señal de finalización de configuración	Proviene de la etapa de configuración, se activa cuando esta máquina ha terminado de llenar la memoria de elementos.
	sCUT: primera bandera contador 60 ns	Se activa cuando el contador ha llegado a 30 ns, lo cual representa el tiempo suficiente para que la unidad trigonométrica haya terminado de obtener los valores de seno y coseno de ambos ángulos, y estén listos para utilizarse.
	sCMult: segunda bandera contador 60 ns	Se activa cuando el contador ha vuelto a alcanzar 30 ns, lo cual representa el tiempo suficiente para que el bloque de multiplicación único haya obtenido los productos c, f y h.
	sCMP: señal comparador de registro de direcciones	Se activa cuando el comparador detecta que el registro de direcciones de la memoria ha alcanzado un valor igual a $3n_{es}$.
	SPSn: señal de finalización de puerto serial *n*	Esta es una bandera presente en cada puerto serial, la cual indica que están disponibles para enviar un nuevo dato.
Salida	**clkAO**: reloj registro ángulos de orientación	Activador de reloj de los registros para capturar los ángulos de orientación.
	enC6ons: activación contador 60 ns	Bandera de inicio para activar el contador de 60 ns.
	ENMEjec: activación de reloj de memoria de elementos	Señal que activa la memoria de elementos.
	enclkE1: activación de reloj etapa 1	Señal para los registros a la salida de la etapa 1.
	enclkE2: activación de reloj etapa 2	Señal para los registros a la salida de la etapa 2.
	clkPSnreg: reloj del registro puerto serial *n*	Señal de reloj para los registros que está a la entrada del puerto serial *n*.
	enPSn: señal de activación del puerto serial *n*	Señal tipo bandera que le indica el inicio de envío de datos al puerto serial *n*.
	rstRD: reset registro de direcciones	Señal de reinicio para el registro contador de direcciones.
	clkRegDec: reloj registro decodificadores	Señal de reloj para los registros a la salida de los codificadores en los dispositivos esclavos.
	reconfig: bandera de reconfiguración	Señal que indica cuando la máquina se encuentra funcionando activamente.
	enclkRD: activación de reloj de registro de direcciones.	Señal que sirve para incrementar el registro de direcciones.
Estados		
Nombre (abreviatura)	**Descripción y transiciones**	**Valor de señales de salida (las señales no explícitas en cada estado tienen valor cero).**

En espera (idle)	El estado de reposo en el que inicia la máquina de estados. Transiciones: **sAO' | sCfg'**, significa que no se han recibido un par de ángulos de orientación nuevos o que el sistema aún no está configurado, por lo tanto, se permanece en «idle». **sAO & sCfg**, significa que el sistema está configurado y se han recibido un par de ángulos de orientación, por lo que se pasa a «HUT».	Todas las señales de salida son cero.
Hold UT (HUT)	Estado de espera. Se inicializa el contador de 60 ns y se permance en este estado hasta que el contador le indica a la máquina que han pasado los 30 ns de retardo de la unidad trigonométrica. También se activa la señal de reloj para fijar los ángulos de orientación. Transiciones: **sCUT = 0**, significa que aún no han pasado 30 ns, y permanece en el mismo estado. **sCUT = 1**, significa que han pasado 30 ns y se pasa a «HMult».	**clkAO = 1** **enCUT = 1** **reconfig = 1**
Hold Mult (HMult)	Estado de espera. Se espera que el contador de 60 ns indique que han vuelto a pasar 30 ns necesarios para la obtención de productos en el multiplicador único. Se utiliza también para reiniciar el regisro de dirección y el selector del demux. Transiciones: **sMult = 0**, significa que aún no han pasado 30 ns, y permanece en el mismo estado. **sMult = 1**, significa que han pasado 30 ns y se pasa a «HMult».	**rstRD = 1** **reconfig = 1**
Etapa 0 (E0)	Comienza la segmentación encauzada. Sólo se activa la señal de reloj de memorias, para leer los datos de los primeros elementos a controlar y empezar a procesarlos en la primera etapa. Transición incondicional hacia «E1».	**reconfig = 1**
Etapa 1 (E1)	Primera etapa del encauzamiento. Se activa el registro de la primera etapa y el reloj de la memoria para leer el segundo conjunto de datos. Transición incondicional hacia «E2»	**ENMEjec = 1** **enclkRD = 1** **enclkE1 = 1** **reconfig = 1**
Etapa 2 (E2)	Segunda etapa del encauzamiento. Se activan los registros de las primeras dos etapas, así como el reloj de lectura de las memorias, obteniendo el tercer conjunto de datos. Transición incondicional hacia «PS1»	**ENMEjec = 1** **enclkRD = 1** **enclkE1 = 1** **enclkE2 = 1** **reconfig = 1**
Puerto Serial 1 (PS1)	Para este estado ya se tiene listo el primer conjunto de fases codificadas. Se realiza la activación del primer puerto serial y de los registros de los mismos. También se incrementa el selector del demux. Transición incondicional hacia «PS2».	**ENMEjec = 1** **enclkRD = 1** **enclkE1 = 1** **enclkE2 = 1** **enPS1 = 1** **clkPS1reg = 1** **reconfig = 1**
Puerto Serial 2 (PS2)	Se tiene listo el segundo conjunto de fases codificadas. Se activa el segundo puerto serial y el registro del mismo. Se incrementa el selector del demux. Transición incondicional hacia «PS3».	**ENMEjec = 1** **enclkRD = 1** **enclkE1 = 1** **enclkE2 = 1** **enPS2 = 1** **clkPS2reg = 1** **reconfig = 1**
Puerto Serial 3 (PS3)	Se tiene listo el tercer conjunto de fases codificadas. Se activa el tercer puerto serial y el registro del mismo. Se incrementa el selector del demux. Transiciones: **sCMP = 0**, significa que el registro de direcciones aún no alcanza su valor máximo, por lo que se pasa a «PS1».	**ENMEjec = 1** **enclkRD = 1** **enclkE1 = 1** **enclkE2 = 1** **enPS3 = 1** **clkPS3reg = 1** **reconfig = 1**

	sCMP = 1, significa que el registro de direcciones ya se encuentra en su valor máximo, por lo que se pasa a «PS1F».	
Puerto Serial 1 Final (PS1F)	La última vez que se activará el primer puerto serial. Dado que los últimos datos ya se encuentran procesándose en la primera etapa, ya no hace falta activar la memoria, pero sí el resto de etapas, así como el puerto serial 1. Transición incondicional hacia «PS2F».	**ENMejec = 1** **enclkE1 = 1** **enclkE2 = 1** **enPS1 = 1** **clkPS1reg = 1** **reconfig = 1**
Puerto Serial 2 Final (PS2F)	La última vez que se activará el segundo puerto serial. Dado que los últimos datos ya se encuentran procesándose en la segunda etapa, ya no hace falta activar la memoria ni E1, pero sí la E2, así como el puerto serial 2. Transición incondicional hacia «PS2F».	**enclkE2 = 1** **enPS2 = 1** **clkPS2reg = 1** **reconfig = 1**
Puerto Serial 3 Final (PS3F)	La última vez que se activará el tercer puerto serial. Ya no hace falta activar ninguna etapa del encauzamiento, así que solo se activa el puerto serial 3. Transición incondicional hacia «WPS3F».	**enPS3 = 1** **clkPS3reg = 1** **reconfig = 1**
Hold PS3 Final (HPS3F)	Estado de espera hasta que el puerto serial 3 finalice de enviar los datos antes de mandar la última señal de control. Transiciones: **sPS3 = 0**, aún no termina el envío de datos, por lo que permance en «WPS3F». **sPS3 = 1**, ha concluido el envío de datos, por lo que se pasa a «RegDecod».	**reconfig = 1**
Registro Decodificadores (RegDec)	En este estado se manda la señal de reloj para los registros que se encuentran a la salida de los decodificadores en los dispositivos esclavos. Se finaliza la repolarización de los elementos con esta señal de control. Transición incondicional hacia «idle».	**clkRegDec = 1** **reconfig = 1**

Con la máquina de control de ejecución establecida, se finaliza el diseño con la estructura interna a manera de diagrama de bloques en la Figura 3-71.

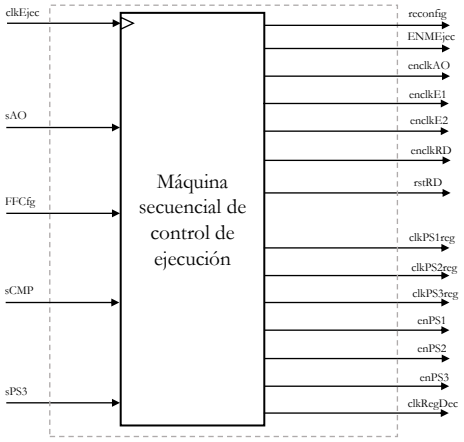


Figura 3-71. Diagrama de bloques de la máquina de control de ejecución.

3.7.15 Interconexión de módulos

Para finalizar el diseño de la etapa de ejecución, ya se ha terminado con el diseño de todos los módulos internos, en la Figura 3-72 se ilustra el diagrama de bloques de esta etapa completada.

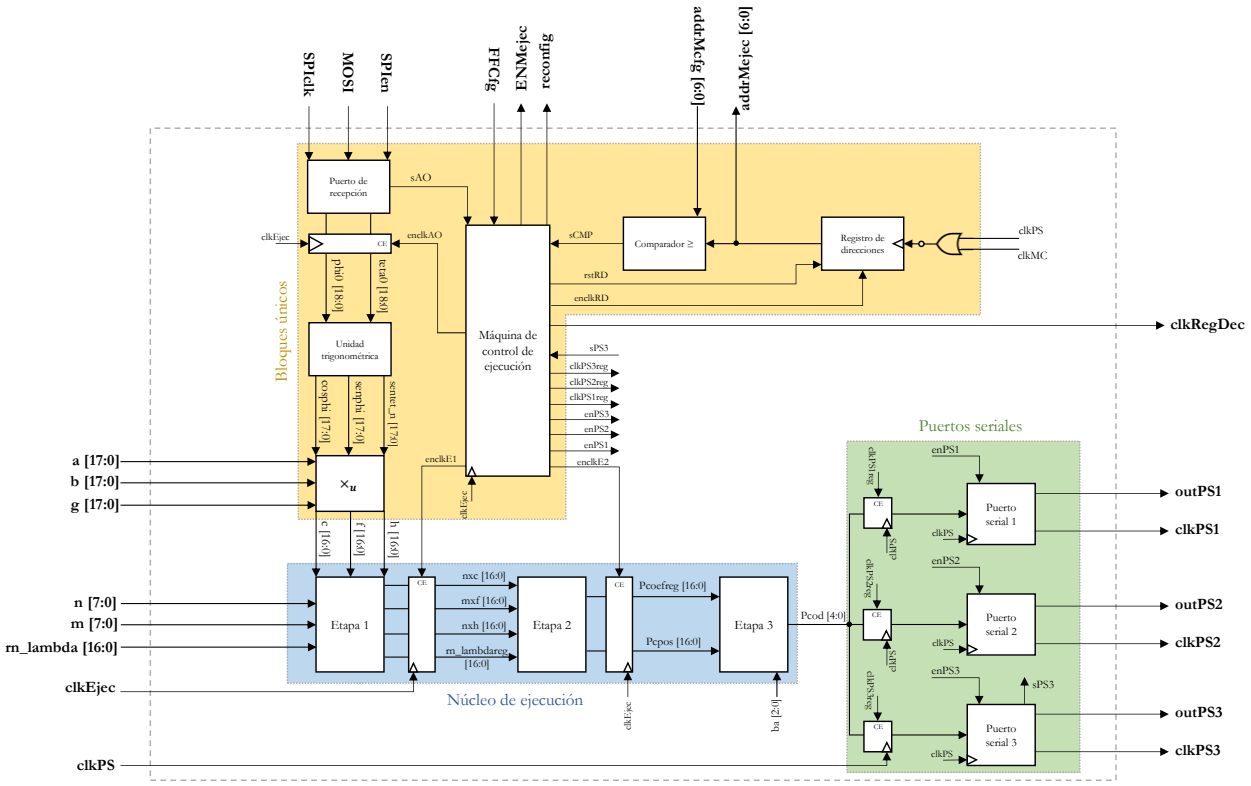


Figura 3-72. Interconexión de módulos para la etapa de ejecución.

3.8 DISEÑO DE LOS DISPOSITIVOS ESCLAVOS

Ya que se ha finalizado con el diseño de la FPGA maestra, el único pendiente son los *dispositivos esclavos*. Como se estipuló en el concepto original de arquitectura (secc. 3.3), los esclavos estarán a cargo de la conversión de los códigos de fases en las señales de control respectivas para los diodos PIN de los elementos del arreglo (controlados por los conmutadores SPDT). Las fases codificadas se reciben por los puertos seriales, llenando un registro de corrimiento cuyos bits son las entradas a los decodificadores. A la salida de los decodificadores también se tienen registros que sirven para sincronizar la repolarización de los elementos. El esquema general se muestra en la Figura 3-73.

Las señales de entrada son **clkRegDec**, **SR** y **clk**. La señal **clkRegDec** es una señal única compartida entre todos los dispositivos esclavos, mientras que la señal **SR** es el dato de entrada único para cada esclavo, al igual que **clk**.

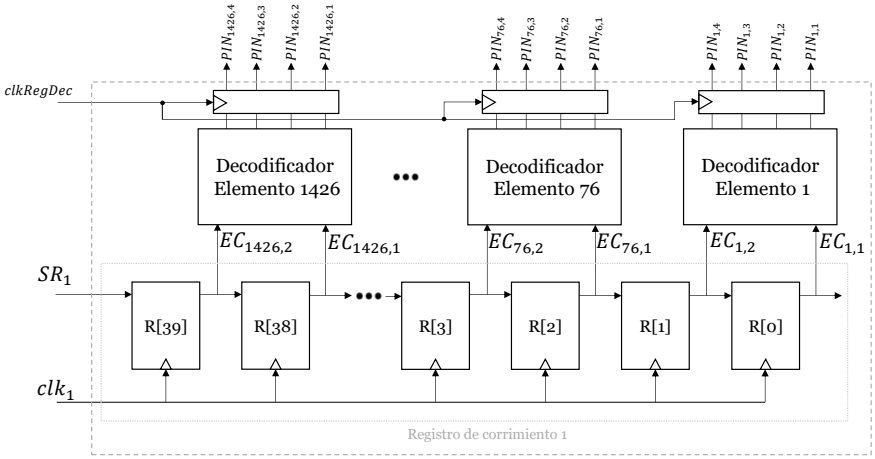


Figura 3-73. Estructura interna del dispositivo esclavo.

El objetivo original planteaba que una FPGA equivaliera a un dispositivo esclavo. Se propuso que cada esclavo controlara hasta **20 elementos**, aunque esto solo era posible si la cantidad de pines de la FPGA es suficiente. Según los requerimientos (secc. 3.1) se tienen dos extremos de escenarios: para el caso de $d_{e_{mín}} = 4$, cada esclavo debe contar con **80** pines, mientras que para $d_{e_{máx}} = 32$ se necesitan **640**.

Las tarjetas FPGA pensadas para los esclavos [53] permitirían hasta 96 señales de control si se reservan 4 para el resto de señales. Siendo así, para cumplir con los casos donde $d_e > 4$, se podría interconectar más de una FPGA para formar un solo esclavo. Para ejemplificar obsérvese la Figura 3-74, que contempla la interconexión de 7 FPGA para un esclavo, dado un arreglo con $b_a = 5$ y $d_e = 32$ cada una controla a tres elementos (excepto la última).

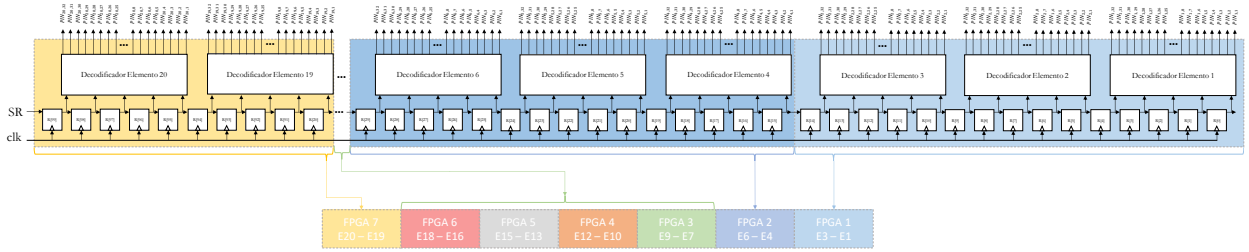


Figura 3-74. Dispositivo esclavo conformado por 7 FPGA.

Por otro lado, los decodificadores están en función de la matriz de funcionamiento del arreglo (explicada al final de secc. 2.1.3). Cada decodificador tendrá de 2 a 5 entradas ($b_{a_{mín}}$ y $b_{a_{máx}}$) y de 4 a 32 salidas ($d_{e_{mín}}$ y $d_{e_{máx}}$). Si estos se diseñaran con lógica combinacional, los dispositivos tendrían que ser reconfigurados y reprogramados con código hecho específicamente para cada arreglo. La otra opción es implementarlos con memorias, donde la dirección de memoria corresponda al código de la fase, y el dato contenido en la localidad sean las salidas que controlan los conmutadores SPDT.

De nuevo tomando las FPGA propuestas que cuentan con máximo 96 pines de salida, considérese la siguiente Tabla 3-18, que contiene la cantidad de elementos que es capaz de controlar una sola FPGA funcionando como esclavo, para todos los posibles casos de d_e.

Tabla 3-18. Capacidad de control y utilización de pines para una FPGA esclavo.

Diodos PIN/elemento (d_e)	Capacidad de control (CC)[12]	Pines utilizados (máx. 96)	N.° FPGA para un esclavo	Diodos PIN/elemento (d_e)	Capacidad de control (CC)	Pines utilizados (máx. 96)	N.° FPGA para un esclavo
32 ($d_{e_{máx}}$)	3	96	6.666666667	17	5	85	4
31	3	93	6.666666667	16	6	96	4
30	3	90	6.666666667	15	6	90	4
29	3	87	6.666666667	14	6	84	4
28	3	84	6.666666667	13	7	91	2.857142857
27	3	81	6.666666667	12	8	96	2.857142857
26	3	78	6.666666667	11	8	77	2.857142857
25	3	**75**	6.666666667	10	9	90	2.857142857
24	4	96	5	9	10	90	2
23	4	92	5	8	12	96	2
22	4	88	5	7	12	84	2
21	4	84	5	6	16	96	2
20	4	80	5	5	18	90	1.111111
19	5	95	4	4 ($d_{e_{mín}}$)	20	80	1
18	5	90	4				

La tabla anterior muestra cómo para mantener una capacidad de control cuyo valor sea entero, la utilización de pines es más o menos aprovechada en cada caso, teniendo un máximo de 96 y un mínimo de 75. Asimismo, se observa que se tienen 12 posibles valores de CC: 3, 4, 5, 6, 7, 8, 9, 10, 12, 16, 18 y 20. Esta tabla se propuso como alternativa de integrar de manera sencilla la escalabilidad a los dispositivos esclavos, de forma que les sea posible controlar desde 3 hasta 20 elementos, según la limitación de pines y capacidad de control dada en función del valor de d_e.

[12] La capacidad de control se refiere a la cantidad máxima de elementos que puede controlar una de las tarjetas FPGA propuestas como esclavos, dada la cantidad de diodos pin de los elementos y los pines disponibles.

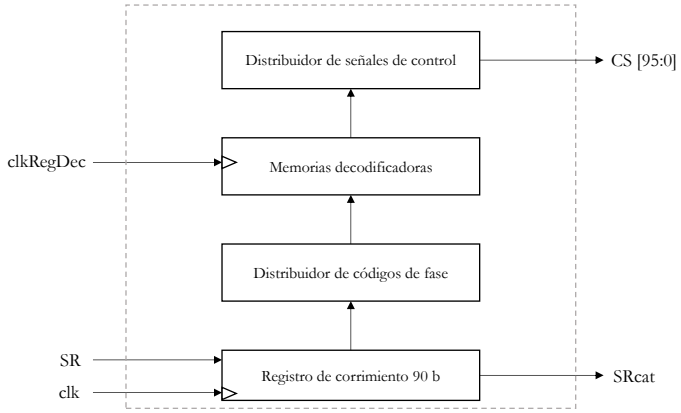


Figura 3-75. Diagrama de bloques del dispositivo esclavo.

Habiendo hecho los análisis anteriores, en la Figura 3-75 se muestra la estructura interna del dispositivo esclavo. Ahora se observa la adición de dos nuevos módulos: el *distribuidor de señales de control* y el *distribuidor de códigos de fase*. El primero sirve para sacar los datos de las memorias hacia los pines de salida de la FPGA, en función de la CC del esclavo; mientras que el segundo funciona para distribuir los códigos de las fases presentes en el registro de corrimiento hacia las direcciones de los decodificadores, que ahora son memorias. En los siguientes apartados se explica a detalle el diseño de los módulos internos del dispositivo esclavo.

3.8.1 Registro de corrimiento

Este bloque está compuesto de un registro con entrada serial y salida paralela (SIPO), cuya función es almacenar los códigos de las fases enviadas por la FPGA maestra hacia el dispositivo esclavo. Para poder cubrir la capacidad de control mayor, el registro tendrá un tamaño de 90 bits (**R [89:0]**), aunque en la mayoría de casos no se lleguen a utilizar todos.

Adicionalmente a la salida paralela, también se cuenta con la salida del último bit recibido (**SRcat**), con el fin de poder interconectar varias FPGA para formar un esclavo si llegase a ser necesario, como fue propuesto en la Figura 3-74 al inicio del diseño. El bit que será la señal **SRcat** dependerá de la capacidad de control y del número de bits del arreglo. Véase la siguiente Tabla 3-19 que contiene la expresión que define al bit de salida, así como algunos ejemplos.

Tabla 3-19. Casos de ejemplo para el bit de salida del registro serial.

CC	b_a	$SRcat = R[90 - CCb_a]$
3	5	R [75]
5	4	R [70]
10	3	R [60]

Por lo tanto, el registro de corrimiento, además de tener un registro SIPO, cuenta con un *Módulo SRcat* que permite elegir el bit de salida mediante los parámetros CC y b_a. En la Figura 3-76 se encuentra el diagrama de bloques correspondiente.

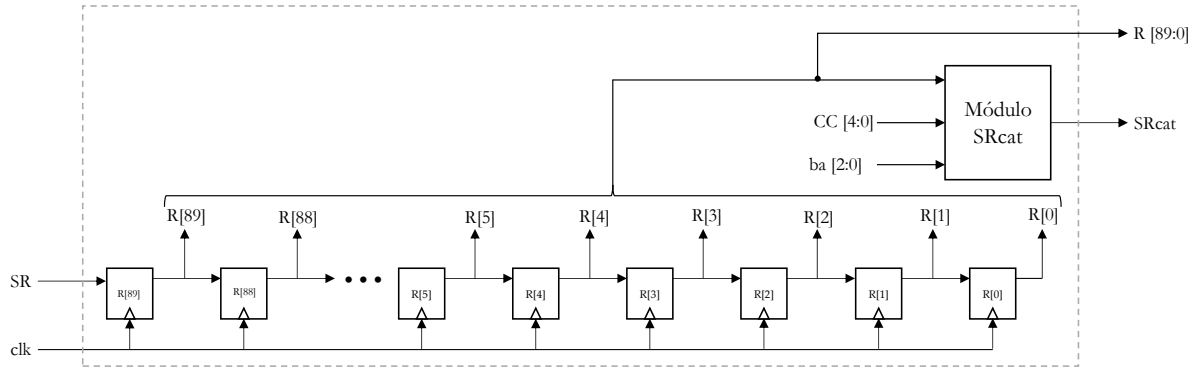


Figura 3-76. Estructura interna del registro de corrimiento.

3.8.2 Decodificadores

La función de los decodificadores del dispositivo esclavo es, de forma simple, convertir el código de la fase de cada elemento a las respectivas señales de control, mismas que manipulan la etapa de potencia y permiten repolarizar los diodos PIN. Recapitulando del final de la sección 2.1.4, la forma en que se deben polarizar los diodos para cada posición discreta depende de la matriz de polarización del arreglo, ergo, para el sistema de control esta matriz está dada como una constante.

La forma más sencilla que se halló de implementación para estos bloques fue hacerlos con memorias, de manera que el código de la fase corresponde a la dirección y el contenido en tal localidad a las señales de control. Para modificar su funcionamiento sin una mayor intervención en el código HDL, basta entonces con modificar los contenidos de las memorias, para que estos coincidan con la matriz de funcionamiento del arreglo.

Los decodificadores estarán hechos partiendo de la Tabla 3-18 presentada anteriormente. El tamaño de cada memoria se obtiene considerando el valor de d_e máximo para cada valor de CC; asimismo, la cantidad de memorias de cada tamaño es definida por la diferencia entre cada CC y la CC anterior. Por ejemplo, para la capacidad de control CC = 3, se tiene un $d_{e_{máx}}$= 32, por lo que se tendrán 3 memorias de 32 bits; para la capacidad de control siguiente CC = 4, se tiene un $d_{e_{máx}}$= 24, por lo que se requiere 4 – 3 = 1 memoria de 24 bits, pues se reutilizan las 3 memorias que ya se tienen de 32 bits, aunque solo se necesiten máximo 24 de esos 32 bits. La misma analogía puede ser aplicada para el resto de valores de la capacidad de control, dando lugar a la Tabla 3-20, que contiene el listado de las memorias que funcionarán como decodificadores.

Tabla 3-20. Memorias propuestas para funcionar como decodificadores.

Cantidad	Tamaño de datos [bits]
3	32
1	24
1	19
1	16
1	13
1	12
1	10
1	9
2	8
4	6
2	5
2	4

La estructura de las memorias se encuentra en la Figura 3-77. Cada una cuenta con un bus de direcciones de 5 bits (**addr_memnb [4:0]**) ($b_{a_{máx}}$) y un tamaño de bits definido por la Tabla 3-20 anterior. La señal de reloj corresponderá a la señal **clkRegDec**, que originalmente estaba pensada para activar a los registros a las salidas de los decodificadores (como aparece en la Figura 3-73). Los datos de salida de cada memoria (**out_memnb [bits-1: 0]**) serán las señales de control para la etapa de potencia. Cada combinación de los parámetros d_e y b_a llevará a considerar solo ciertos bits de entrada y salida de cada memoria. Estas consideraciones corresponderán a las tareas del distribuidor de códigos de fase y del distribuidor de señales de control, explicados más adelante. La señal de activación de cada memoria (**ena_memnbm**) es controlada por el módulo *activador de memorias*, que está en función de la capacidad de control, pues solo las memorias requeridas para cada CC permanecerán activas.

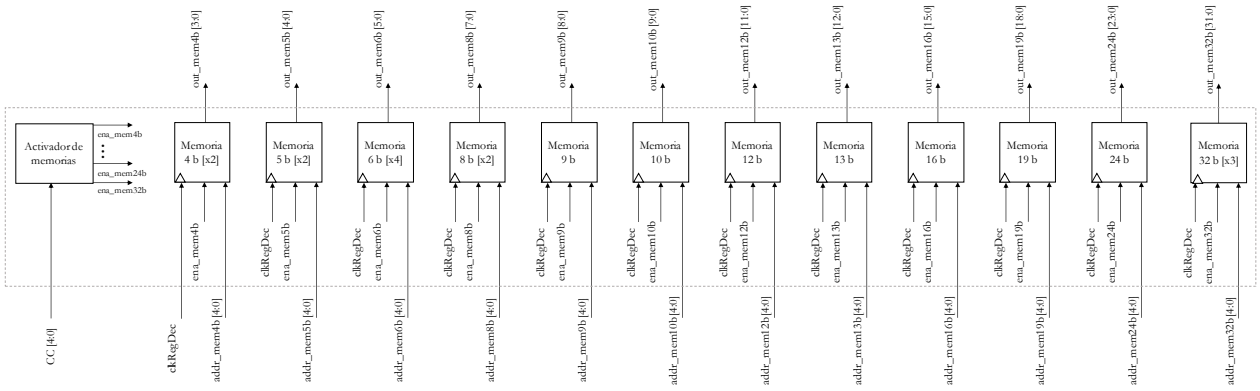


Figura 3-77. Estructura interna de las memorias decodificadoras.

3.8.3 Distribuidor de códigos de fase

Este bloque tiene como función distribuir correctamente los códigos de fases presentes del registro SIPO hacia los buses de direcciones de las memorias, en función de la capacidad de control (CC) y del número de bits del arreglo (b_a). La distribución hecha por este módulo se realiza mediante multiplexores y concatenaciones. Para entender de forma fácil la manera en que se asignan las señales del registro **R [89:0]** hacia las direcciones de memorias, se puede consultar la Tabla 3-21 que contiene algunos ejemplos para distintos parámetros.

Tabla 3-21. Casos de ejemplo para el distribuidor de códigos de fase.

b_a	CC	Buses memorias
2	3	addr_mem32b3 [4:0] = {3'b000, R[89:88]} addr_mem32b2 [4:0] = {3'b000, R[87:86]} addr_mem32b1 [4:0] = {3'b000, R[85:84]}
3	5	addr_mem19b1 [4:0] = {2'b00, R[89:87]} addr_mem24b1 [4:0] = {2'b00, R[86:84]} addr_mem32b3 [4:0] = {2'b00, R[83:81]} addr_mem32b2 [4:0] = {2'b00, R[80:78]} addr_mem32b1 [4:0] = {2'b00, R[77:75]}
4	7	addr_mem32b1 [4:0] = {1'b0, R[89:86]} addr_mem32b2 [4:0] = {1'b0, R[85:82]} addr_mem32b3 [4:0] = {1'b0, R[81:78]} addr_mem24b1 [4:0] = {1'b0, R[77:74]} addr_mem19b1 [4:0] = {1'b0, R[73:70]} addr_mem16b1 [4:0] = {1'b0, R[69:66]} addr_mem13b1 [4:0] = {1'b0, R[65:62]}

Se explicará el primer renglón de la Tabla 3-21 para una mejor comprensión. En este, debido a los valores de $b_a = 2$ y CC = 3, solo se tomarán en cuenta los 6 bits más significativos del puerto serial (**R [89:84]**). De esos 6 bits, cada par corresponde a cada una de las tres fases: la primera se guarda en **R [85:84]**, la segunda en **R [87:86]** y la última en **R [89:88]**. Por otro lado, las tres memorias utilizadas para decodificar son las de 32 bits, cuyas direcciones de memoria (al igual que todas) son de 5 bits; entonces, se concatenan 3 bits a la izquierda (3'b000) de cada par de bits de fase para tener una dirección de memoria del tamaño requerido. Para las demás memorias que no se utilizan, el bus de dirección se mantiene completamente en cero y permanecen inactivas.

Así bien, el módulo distribuidor de códigos de fase se tiene un multiplexor para cada memoria, que asigna los bits correspondientes según cada caso, concatenando ceros en los casos necesarios para tener los 5 bits de dirección que todas las memorias poseen. Véase la Figura 3-78, que contiene la estructura interna del distribuidor de códigos de fase.

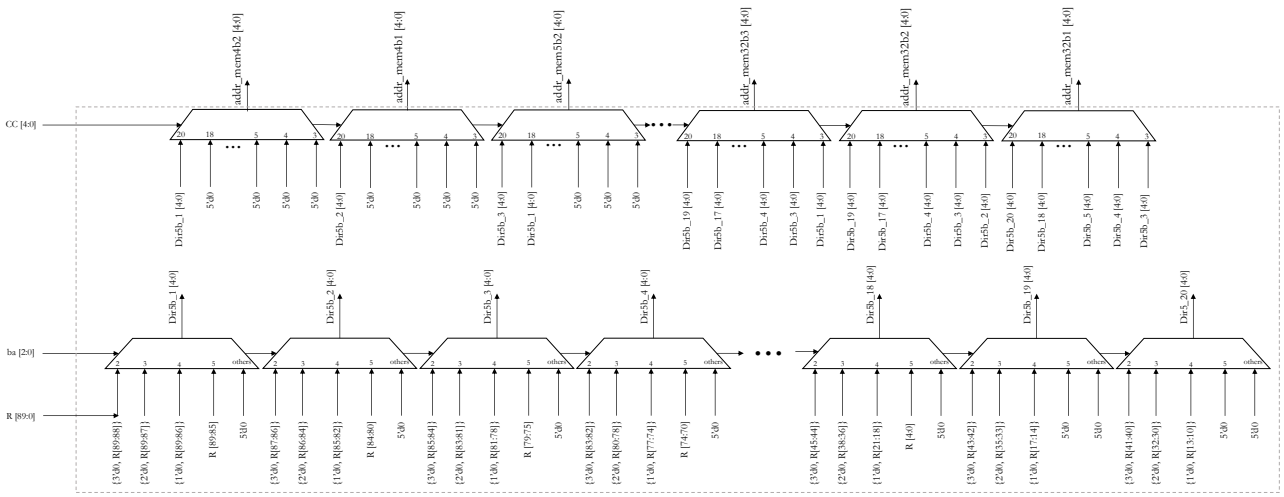


Figura 3-78. Estructura interna del distribuidor de códigos de fase.

3.8.4 Distribuidor de señales de control

Este bloque tiene como propósito elegir qué señales de salida de las memorias irán hacia las salidas directas de la FPGA, es decir, las señales de control para la etapa de potencia. La forma en que el distribuidor asigna las señales de salida depende directamente de la capacidad de control, pues solo con este parámetro se puede conocer cuántas y cuáles señales de qué memorias serán las correspondientes salidas del esclavo. Para cada valor de CC se utiliza el valor de uso de pines más grande, según la Tabla 3-18. Por ejemplo, para una CC de 4, el uso de pines mayor es de 96.

Para comprender de manera fácil el funcionamiento del distribuidor de señales, al igual que se ha hecho anteriormente, se incluye la Tabla 3-22, que cuenta con la asignación de pines de salida para tres casos de ejemplo. Al bus de 96 bits de salida se le nombrará **SC [95:0]**.

Tabla 3-22. Casos de ejemplo para el distribuidor de señales de control.

CC	Uso de pines máx.	$d_{e_{máx}}$	Señales de control de salida
5	95	19	SC [95:0] = {1'b0, out_mem19b1[18:0], out_mem24b1[18:0], out_mem32b3[18:0], out_mem32b2[18:0], out_mem32b1[18:0]}
8	96	12	SC [95:0] = {out_mem12b1[11:0], out_mem13b1[11:0], out_mem16b1[11:0], out_mem19b1[11:0], out_mem24b1[11:0], out_mem32b3[11:0], out_mem32b2[11:0], out_mem32b1[11:0]}
12	96	8	SC [95:0] = {out_mem8b2[7:0], out_mem8b1[7:0] ,out_mem9b1[7:0], out_mem10b1[7:0], out_mem12b1[7:0], out_mem13b1[7:0], out_mem16b1[7:0], out_mem19b1[7:0], out_mem24b1[7:0], out_mem32b3[7:0], out_mem32b2[7:0], out_mem32b1[7:0]}

Para terminar de ejemplificar, se explicará de forma verbal el caso del primer renglón, con una CC = 5. Para este valor de CC, según la Tabla 3-18, se tiene un uso de pines máximo de 95 y mínimo de 85; un valor d_e máximo de 19 y mínimo de 17. En otras palabras, la capacidad de control de 5 se utiliza para elementos en el rango de 19 a 17 diodos PIN. En el caso de d_e = 17, únicamente se requerirían 85 pines, sin embargo, resulta más fácil que el distribuidor mantenga las señales del caso más grande (d_e = 19) y el usuario conozca que las señales sobrantes no se utilizan y él mismo las descarte.

Siendo así, **SC [95:0]** es la concatenación de los 19 bits ($b_{a_{máx}}$) menos significativos de las 5 memorias decodificadoras, y el bit más significativo como un cero constante (1'b0) debido a que la utilización máxima es de 95. Entonces, de esas 95 señales efectivas, cada grupo de 19 corresponde a cada uno de los 5 elementos, como lo ilustra la Figura 3-79.

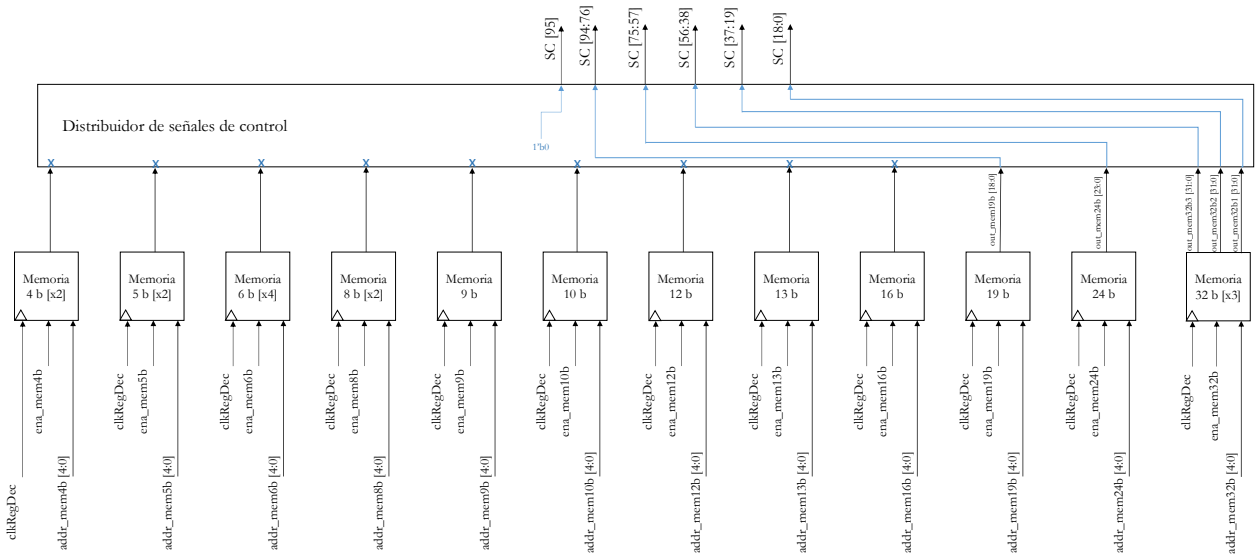


Figura 3-79. Señales de salida asignadas por el distribuidor de señales de control para una CC = 5.

Por el lado contrario, si se tiene un valor $d_{e_{mín}} = 17$, solamente es necesario descartar (dejar desconectados) los dos bits más significativos de cada conjunto de 19. Esto aplica para todos los rangos $d_{e_{mín}}$ y $d_{e_{máx}}$ de toda capacidad de control, lo que ciertamente conlleva a cierto «desperdicio» de pines, aunque estos nunca llegan a ser suficientes para poder cubrir todos los diodos de otro elemento. Por ejemplo, para el caso de CC = 5 y $d_e = 17$, se inutilizan 11 pines de los 96 disponibles, pero esas 11 señales que se desperdician no son suficientes para controlar otro elemento, y esto es así para el resto de los casos donde existe algo de «subutilización».

El distribuidor de señales de control fue descrito a modo de tabla de verdad con las concatenaciones de la Tabla 3-22 y el resto de casos bajo la misma analogía. Se infirió un multiplexor para cada bit del bus CS [95:0], siendo el selector el parámetro de capacidad de control.

3.8.5 Interconexión de módulos

Como se estableció en la Figura 3-75, el dispositivo esclavo está compuesto de los cuatro módulos que fueron diseñados en las secciones anteriores: el registro de corrimiento, los decodificadores, el distribuidor de códigos de fase y el distribuidor de señales de control. La interconexión de estos bloques se muestra en la Figura 3-80, y con esta se concluye el diseño del dispositivo esclavo, recordando que todos los requeridos son réplicas exactas de un diseño único.

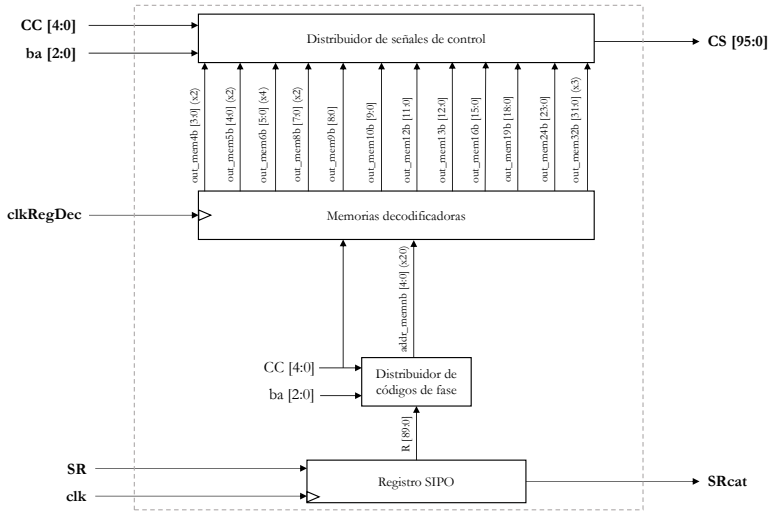


Figura 3-80. Estructura interna del dispositivo esclavo.

Antes de finalizar, se considera relevante hacer un par de comentarios importantes acerca del diseño de estos dispositivos. Dada la manera en que fueron diseñados, se tiene una escalabilidad un poco más restrictiva en comparación a la FPGA maestra, como ejemplo el hecho de tener que escoger entre valores fijos de la capacidad de control sin completa libertad, y el hecho de que el diseño esté hecho en cierta forma «a la medida» de la tarjeta que se había elegido para implementar los esclavos.

Esas y el resto de decisiones de diseño fueron sujetas a análisis y discusiones con los tutores de tesis, con el objetivo de no complicar ni alargar el diseño como ocurrió con la FPGA maestra, aunque siempre procurando conservar las características de escalabilidad y reconfigurabilidad, que son parte indispensable del diseño de este sistema. De cualquier forma, las estructuras base de los módulos internos también están pensadas para ser extrapoladas de manera fácil, para dispositivos que deseablemente cuenten con una cantidad de pines mayor y, por ende, una mayor capacidad de control.

Con el término de esta sección se finaliza el diseño del sistema de control. En el **Anexo 1** se encuentra un diagrama del sistema de control completo, con la interconexión de los bloques de la tarjeta maestra y de un dispositivo esclavo. En el **Anexo 2** se presenta el diagrama esquemático obtenido del proyecto en Vivado. Los siguientes capítulos hablarán sobre los experimentos hechos con el sistema, los resultados y los respectivos análisis y conclusiones del trabajo.

Nota: para una mayor veracidad y como manera de dejar constancia de la realización de este proyecto, en el repositorio de Github del siguiente enlace se pueden consultar algunos de los principales códigos en HDL, que fueron descritos para el proyecto de la FPGA maestra: https://github.com/Ernest0o/FPGAmaster.

CAPÍTULO 4
EXPERIMENTACIÓN

En el presente capítulo se relata la etapa de experimentos realizados en el sistema diseñado. Se describe la metodología de pruebas llevada a cabo, se habla a detalle sobre cómo se realizaron experimentos con distintos valores de parámetros y de ángulos de orientación. Asimismo, dentro de este capítulo se detalla el módulo prototipo con el que se realizaron las pruebas y sus características, al igual que un módulo adicional que fue creado como manera de corroborar el funcionamiento del sistema. Los resultados de la etapa de experimentación están alojados y comentados en el capítulo correspondiente; en este, a grandes rasgos, sólo se comenta la manera en que las pruebas fueron realizadas y el prototipo del sistema implementado.

4.1 MÓDULO PROTOTIPO

Para poder visualizar el funcionamiento completo del sistema de control, es necesario contar un dispositivo que funja como maestro, y la cantidad de esclavos necesaria para cubrir el total de señales de control para un arreglo en concreto. Las características que deben tener las FPGA para cada papel se enlistan en las siguientes tablas.

Tabla 4-1. Recursos necesarios para la FPGA maestra.

Característica	Cantidad	Detalles
Recursos lógicos	37,200 LUT	La cantidad mínima requerida de recursos para ser la tarjeta maestra del sistema. Incluye los recursos necesarios para 30 núcleos de ejecución, los de la etapa de configuración, y los de la interfaz de usuario y carga de datos.
Pines libres	190	Para contar con 90 puertos seriales, además de los pines para el puerto de recepción de ángulos de orientación y los de la interfaz de usuario.

Tabla 4-2. Recursos para una FPGA esclava.

Característica	Cantidad	Detalles
Recursos lógicos	1,600 LUT	La cantidad mínima requerida de recursos para todos los componentes de un dispositivo esclavo
Pines libres	84 - 644	Para poder satisfacer el caso más simple y el más grande de los establecidos en los requerimientos.

Por el lado de la FPGA maestra no existe mayor inconveniente puesto que es una única tarjeta. Sin embargo, por los requerimientos (secc. 3.1) se necesitan varias decenas de dispositivos esclavos para cubrir los 1500 elementos. Realmente no se contó con el presupuesto suficiente para adquirir tal cantidad de tarjeta; esto conllevó a que el módulo prototipo para la etapa de experimentación sea algo bastante más sencillo, pero que de cualquier forma sirva para visualizar el funcionamiento base del sistema.

Con este propósito se optó por un prototipo compuesto por únicamente dos dispositivos, uno que funcionará como maestro y uno como esclavo. Utilizando tarjetas ya disponibles en el laboratorio, el módulo prototipo está compuesto de los siguientes dos componentes:

Para la FPGA maestra se utilizará una tarjeta Nexys Video de Digilent [54], con una FPGA XC7A200T de Xilinx que cuenta con 215,360 celdas lógicas. Aunque únicamente posee 36 pines de propósito general, téngase presente que solo se trabajará con un esclavo, por lo que es posible utilizar los switches incluidos en la tarjeta para poder escoger qué puertos seriales tener en los pines de salida.

Para el dispositivo esclavo la tarjeta elegida es una Basys 3 de Digilent [55], con una FPGA XC7A35T de Xilinx que cuenta con 33,280 celdas lógicas. Esta también cuenta con solo 36 pines de propósito general, pero es posible alternar las señales de salida mediante los switches con los que también cuenta.

Para el dispositivo encargado de enviar los ángulos de orientación, se eligió un microcontrolador Tiva TM4C1294 de Texas Instruments [56], que cuenta con el periférico dedicado para el protocolo SPI para facilitar la comunicación con la FPGA maestra.

La conexión entre ambas tarjetas se realizará mediante cables tipo «dupont», siempre manteniendo presente que estos no son la mejor opción para envío de señales en ese rango de frecuencias. Lo ideal será que, en una implementación real del sistema, las conexiones entre la FPGA maestra y los dispositivos esclavos sea mediante pistas en un circuito impreso, lo que les permitirá comunicarse a la frecuencia máxima del sistema de 130 MHz. Considerando la limitación anterior, se utilizará un reloj de 110 MHz para los puertos seriales (**clkPS**). El resto serán: 55 MHz para el núcleo de ejecución (**clkEjec**), 5 MHz para la etapa de configuración (**clkCfg**), 150 MHz para el módulo SPI (**clkSPI**), 20 MHz para el llenado de parámetros (**clkFast**) y 10 MHz para el puerto UART (**clkUART**). Se finaliza este apartado con un diagrama simplificado de las interconexiones entre las tres tarjetas del módulo prototipo.

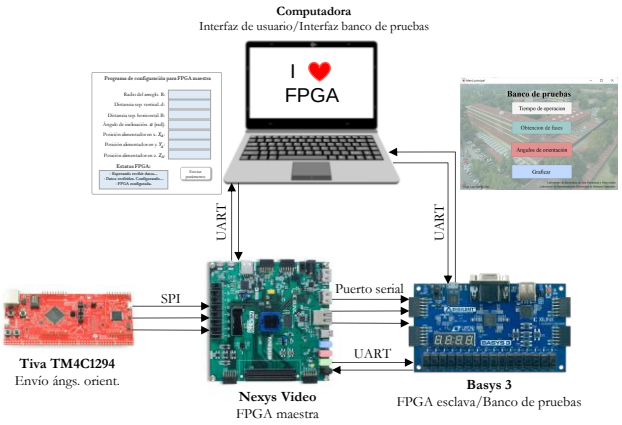


Figura 4-1. Conexiones de tarjetas para el módulo prototipo

4.2 MÓDULO DE CORROBORACIÓN DE FUNCIONAMIENTO

Para verificar que el sistema funciona de forma correcta se puede seguir este procedimiento: para el tiempo de respuesta basta con tan solo monitorear el tiempo en el que la bandera de **reconfig** de la máquina de control de ejecución permanece en alto; para las fases finales, se pueden comparar de forma visual una por una las señales de control a la salida del esclavo con las obtenidas por Matlab. Este proceso se tendría que repetir para la serie de experimentos propuestos.

Como es posible notar, la anterior metodología sería bastante tardada e impráctica, ya que se tendría que hacer de forma cuidadosa y es propensa a errores. El *módulo de corroboración de funcionamiento/banco de pruebas* se creó para facilitar todas estas tareas[13]. Es una funcionalidad extra que se añadió al dispositivo esclavo, y se encarga de hacer el análisis de las señales de control del esclavo, además de la medición del tiempo de respuesta del sistema. El módulo se comunica con Matlab a través de UART y allí es donde se hacen la mayoría de análisis, por lo que en esencia, dentro de la FPGA se ejecutan las siguientes tareas:

1. Para la medición del tiempo de respuesta, el banco de pruebas fuerza el proceso de ejecución en la FPGA maestra y mide mediante un contador el periodo en alto de la señal **reconfig**. Este valor es enviado a Matlab y convertido a unidades de tiempo para desplegarse en pantalla.

2. Para el análisis de fases, el módulo de corroboración envía los códigos de las fases almacenadas en el registro **R[89:0]** del esclavo hacia Matlab, donde allí se pueden hacer comparaciones y obtener diversos resultados que serán mencionados más adelante. Por supuesto, para estos análisis Matlab también debe ser configurado con los parámetros del arreglo y saber qué esclavo se está analizando en todo momento.

[13] El diseño de este banco de pruebas estuvo a cargo de un compañero de servicio social llamado Jorge Badillo, a quien le agradezco personalmente su tiempo, apoyo y dedicación.

Finalmente, en la Figura 4-2 se puede visualizar una captura de pantalla de la ventana principal del banco de pruebas.

Figura 4-2. Menú principal del banco de pruebas.

4.3 PROTOCOLO DE PRUEBAS

Además de la obtención del tiempo de respuesta, las pruebas para corroborar el funcionamiento del sistema se resumen en la corroboración de las fases entregadas por los dispositivos esclavos, dado un arreglo de ciertos parámetros y un par de ángulos de orientación en determinado momento. Para llevar a cabo los experimentos y poder llegar a resultados concluyentes, es necesario variar distintos valores: ya sean los parámetros de la geometría del arreglo, la cantidad de bits de los elementos, los ángulos de orientación, o bien la cantidad de esclavos a analizar.

Después de consultar con los tutores del trabajo, se sugirió que las pruebas sean realizadas para un solo arreglo de prueba con las características mostradas en la siguiente Figura 4-3. Además de los parámetros allí mostrados, las demás características son: frecuencia de operación $f = 30\,GHz$ y separación entre elementos $d = B = 6\,mm$.

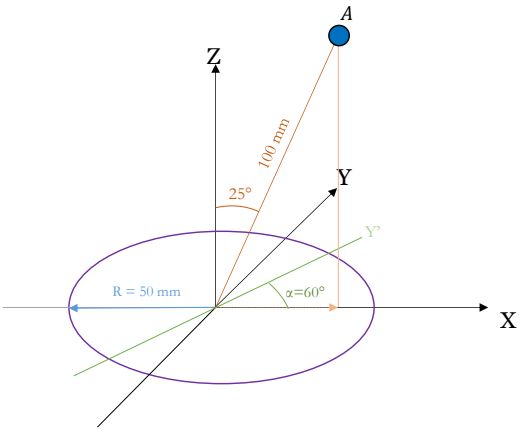


Figura 4-3. Diagrama del arreglo de prueba para la etapa de experimentación.

Partiendo de la ilustración anterior se obtendrán las coordenadas del alimentador, la longitud de onda en mm y el número de elementos. Comenzando por el alimentador, cuya posición está definida por un vector de magnitud 100 mm con inclinación de 25° con respecto al eje Z, que además se encuentra sobre el eje X, por lo que su componente en Y es igual a cero ($y_a = 0$).

Realizando la obtención de componentes mediante trigonometría básica se tiene lo siguiente:

$$x_a = 100sen(25°);\;\; z_a = 100cos(25°)$$

$$x_a = 42.2618\,mm;\;\;\; z_a = 90.6307\,mm$$

La longitud de onda electromagnética para una frecuencia de 30 GHz es $\lambda = \frac{C}{f} = \frac{299792458\,\frac{m}{s}}{30\times10^9\,Hz} = \; 0.01\,m = 10\,mm.$

Finalmente, con los parámetros ya obtenidos se realizó el algoritmo de configuración con la ayuda del programa hecho en Matlab. De la simulación se obtuvo que el arreglo posee $e_a = 253$ elementos. Entonces, se propopnen $n_{es} = 15$ dispositivos esclavos para el control del arreglo, por lo que cada esclavo estará controlando a $253/15 = 17$ elementos. Respecto a los «núcleos de ejecución» (secc. 3.7.12), se necesitarán solo $15/3 = 5$ para controlar a los 15 esclavos. Las características del arreglo de prueba se resumen dentro de la Tabla 4-3.

Tabla 4-3. Descripción de parámetros del arreglo de prueba.

Parámetro	Valor
Radio del arreglo (R)	50 mm
Sep. vertical (d)	6 mm
Sep. horizontal (B)	6 mm
Inclinación eje Y' (α)	60°
Posición alimentador en x (x_a)	42.2618 mm
Posición alimentador en y (y_a)	0 mm
Posición alimentador en z (z_a)	90.6307 mm
Longitud de onda (λ)	10 mm
Cantidad de núcleos (n_{nuc})	5

Volviendo a los experimentos a realizar, puesto que los parámetros del arreglo son únicos, serán los ángulos de orientación θ_0, φ_0 y el número de bits b_a los valores que se varíen. Se decidió hacer pruebas con tres pares de ángulos de orientación y variar el número de bits desde 2 hasta 5, lo que dará un total de 12 experimentos distintos y 180 esclavos analizados. Las pruebas están enlistadas y enumeradas en la Tabla 4-4.

Tabla 4-4. Listado de experimentos a realizar.

N.° experimento	b_a	Ángulos de orientación
2-a	2	$\theta_0 = 0° \; y \; \varphi_0 = 0°$
2-b		$\theta_0 = 15° \; y \; \varphi_0 = 180°$
2-c		$\theta_0 = 30° \; y \; \varphi_0 = 180°$
3-a	3	$\theta_0 = 0° \; y \; \varphi_0 = 0°$
3-b		$\theta_0 = 15° \; y \; \varphi_0 = 180°$
3-c		$\theta_0 = 30° \; y \; \varphi_0 = 180°$
4-a	4	$\theta_0 = 0° \; y \; \varphi_0 = 0°$
4-b		$\theta_0 = 15° \; y \; \varphi_0 = 180°$
4-c		$\theta_0 = 30° \; y \; \varphi_0 = 180°$
5-a	5	$\theta_0 = 0° \; y \; \varphi_0 = 0°$
5-b		$\theta_0 = 15° \; y \; \varphi_0 = 180°$
5-c		$\theta_0 = 30° \; y \; \varphi_0 = 180°$

Habiendo establecido los parámetros que se utilizarán en los experimentos a realizar, el protocolo a seguir para cada uno de ellos será el siguiente:

1. Se configurará la tarjeta maestra mediante la interfaz de usuario, ingresando los parámetros de la Tabla 4-3, y colocando el número de bits indicado por el n.° de experimento (Tabla 4-4).
2. Se enviarán los ángulos de orientación indicados por el n.° de experimento, que serán almacenados por el sistema.
3. Mediante el módulo de corroboración (secc. 4.2) se forzará el sistema a hacer una reorientación con los últimos ángulos recibidos (en paso 2). Se capturará en Matlab el tiempo de respuesta y las fases del esclavo elegido por los switches de la tarjeta. Los resultados de interés se mencionan en la sección de Análisis.
4. El paso 3 se repetirá sucesivamente hasta completar los 15 dispositivos esclavos. Después se volverá al paso 1 para modificar parámetros y se repetirá hasta terminar los 12 experimentos propuestos.

4.4 ANÁLISIS

La etapa de experimentación tiene el propósito de corroborar el funcionamiento del sistema diseñado, mediante pruebas con el módulo prototipo y el protocolo propuestos en el presente capítulo. Los resultados que se desean obtener se mencionan a detalle en la presente sección.

Respecto al tiempo de respuesta, se verificará que se mantenga cercano al tiempo deseado de 1 microsegundo, aun considerando que el prototipo estará trabajando a una frecuencia 17 % menor que la máxima posible y que el número de elementos que controla cada esclavo es 15 % menor que el máximo establecido. Los resultados de interés serán los siguientes:

- Tiempo de respuesta medio, mínimo, máximo para los 12 experimentos de la Tabla 4-4 – Se espera que los resultados ronden cerca de la unidad de microsegundo (requerimientos en secc. 3.1).

- Variación del tiempo de respuesta en función de los bits del arreglo (b_a) – Se espera que, como se decidió en la sección 3.7.12, el tiempo de respuesta se mantenga invariante a la cantidad de bits.

Asimismo, el tiempo de respuesta de la FPGA maestra (t_{rm}) deberá tener un valor esperado de 1.109 us, cuyo valor se obtiene a partir de la siguiente expresión (4.a), que toma en cuenta las etapas de la segmentación encauzada de la etapa de ejecución y la cantidad de cálculos que realiza cada núcleo. El tiempo de respuesta total del sistema (t_{tr}) será la suma del tiempo de la FPGA maestra y del dispositivo esclavo (t_{res}).

$$t_{rm} = (10 + 3e_{es})T_{clkEjec} \tag{4.a}$$

Respecto a las fases entregadas por el dispositivo esclavo, estas son los resultados más importantes, ya que permitirán ratificar la correcta elección del formato numérico y de todos los componentes del sistema, que trabajan en conjunto para llegar a estos valores finales. Los resultados y análisis serán los siguientes:

- Respecto a las fases codificadas del sistema y las obtenidas por Matlab, se obtendrá por cada experimento: el número de fases que fueron distintas, el porcentaje de fases distintas del total, el error absoluto máximo y el medio. Más detalles se pueden consultar en la Tabla 4-5.

Tabla 4-5. Expresiones para cálculo de errores entre el sistema y Matlab.

Resultado	Expresión
Cantidad de fases codificadas distintas entre Matlab (ml) y experimentales (exp)	$nP_{cod_{dif}} = \sum_{i=0}^{e_a} D_i \,;\, D_i = \begin{Bmatrix} 1 \; si \; P_{cod_{i_{ml}}} \neq P_{cod_{i_{exp}}} \\ 0 \; si \; P_{cod_{i_{ml}}} = P_{cod_{i_{exp}}} \end{Bmatrix}$
Porcentaje de fases distintas	$\%P_{cod_{dif}} = (nP_{cod_{dif}}/e_a)(100\,\%)$
Vector de errores entre fases codificadas	$P_{cod_{err}}[i] = \left\lvert P_{cod_{i_{exp}}} - P_{cod_{i_{ml}}} \right\rvert$
Error absoluto máximo	$Error_{cod_{máx}} = máx\left(P_{cod_{err}}\right)$
Error absoluto mínimo	$Error_{cod_{mín}} = mín\left(P_{cod_{err}}\right)$
Error absoluto medio	$Error_{cod_{med}} = \frac{\sum P_{cod_{err}}}{e_a}$

- Respecto a la orientación dada por las fases codificadas del sistema y las obtenidas por Matlab, los ángulos θ_0, φ_0 se obtendrán por esclavo y por experimento mediante el *algoritmo de prueba de orientación*, que es mencionado más adelante.

El *algoritmo de prueba de orientación* sirve para corroborar el lóbulo pricipal emitido por las fases discretas de varios elementos, considerando sus coordenadas de posición y distancias al alimentador. De forma simple, se realiza un barrido para distintos valores de θ, φ en incrementos ligeros, comprobando mediante la suma de E_i (expresado en Tabla 4-6) la magnitud del campo generado por los elementos para ese par de ángulos ($C_{\theta,\varphi}$), valor que es almacenado iterativamente en una matriz.

Tabla 4-6. Expresiones para el cálculo de campo en un punto dado por los ángulos θ, φ.

$\varphi_{ind_i} = -kr_i$
$\varphi_{dc_i} = (P_{cod_i})\left(\frac{2\pi}{N}\right)$
$\varphi_{el_i} = \varphi_{ind_i} + \varphi_{dc_i}$
$E_i = e^{j\varphi_{el_i}}\, e^{jkx_i sen(\theta)\cos(\varphi)}\, e^{jky_i sen(\theta) sen(\varphi)}$
$C_{\theta,\varphi} = \sum_{i=0}^{n} E_i$

Después de finalizar el barrido, se ubica el primer máximo de todos los valores de campo de la matriz y sus equivalentes de θ, φ, siendo los ángulos para los que la energía reflejada por los elementos en conjunto fue mayor. Este algoritmo será aplicado para cada esclavo analizado (17 elementos), y para los 253 elementos de cada experimento. Con los resultados se podrán analizar las orientaciones de Matlab y del sistema, lo que a su vez hará posible analizar el error de orientación entre ambos casos y en función del número de bits (b_a). El código del algoritmo fue programado en Matlab y se adjunta a continuación:

```
FasesMtrxRad = FasesCodMtrx.*(2*pi/dba); % Matriz de fases discretas en radianes

 for i_esc = 1: 1: 15 % 15 esclavos por experimento.
    k= (2*pi)/lambda;

    P_ind = rnMtrx(i_esc, :).*(-k); % Vector de fases inducidas
    P_dc = FasesMtrxRad(i_esc, :); % Vector de fases en rad

    P_el = P_ind + P_dc;

    phi_vec = (0:0.25:359); % en grados con incremento de 0.25°
    phi_vecrad = phi_vec.*(pi/180); % en radianes
    teta_vec = (0:0.05:60); % en grados con incremento de 0.05°
    teta_vecrad = teta_vec.*(pi/180); % en radianes
    E_sum = 0;
    suma_mtrx = zeros(numel(teta_vec), numel(phi_vec));
    for t = 1:1:numel(teta_vec) % ← Barrido en teta
            tetha= teta_vecrad(t);
            for p = 1 : 1 : numel(phi_vec) % ← Barrido en phi
                phi= phi_vecrad(p);
                E_sum = 0; % Reinicio de suma
                for i = 1 : 1: 17 % 17 elementos por esclavo
                      Ei = (exp(1j*P_el(i))*exp(1j*k*xnMtrx(i_esc,
i)*sin(tetha)*cos(phi))*exp(1j*k*ynMtrx(i_esc, i)*sin(tetha)*sin(phi)));
                      E_sum = E_sum + Ei;
                end
                suma_mtrx (t, p) = norm(E_sum); % Guardar magnitud de E_sum
            end
    end

    [a, b] = find(suma_mtrx == max(max(suma_mtrx)), 1); % Encontrar primer máximo

    text = ['Esclavo: ', num2str(i_esc), ' TetaOmáx: ', num2str(teta_vec(a)), ' PhiOmáx: ',
num2str(phi_vec(b)), ' [°]']; % Despliega máximo para cada esclavo

    disp(text);

end
```

CAPÍTULO 5
RESULTADOS

En el presente capítulo se analizarán y comentarán los resultados obtenidos en la etapa de experimentación, sobre las pruebas que fueron propuestas en la sección 4.3. Los resultados a analizar serán aquellos mencionados en la seccción 4.4, que se consideran como los más relevantes de la etapa de experimentación.

5.1 FASES CODIFICADAS

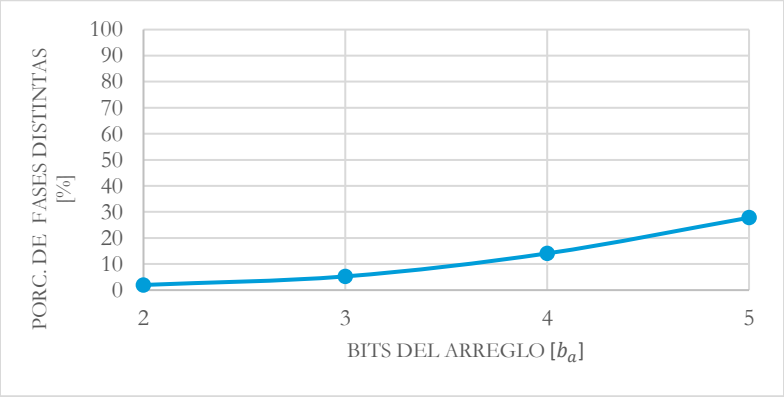
Las fases entregadas por cada esclavo son el resultado más importante, pues estas son las que controlan la correcta orientación de la antena hacia el par de ángulos deseados; si no se cuenta con un conjunto de fases correcto, ningún otro aspecto del sistema tiene relevancia. Las fases codificadas de cada esclavo analizado fueron enviadas hacia Matlab para realizar los análisis cuyos resultados son aquí comentados.

Se comenzará con el análisis de las diferencias entre las fases codificadas obtenidas por Matlab y las del sistema. Para cada experimento se analizaron las 253 fases y se compararon con las de Matlab. Los resultados pueden encontrarse en la Tabla 5-1. Posteriormente se encuentran los comentarios respectivos.

Tabla 5-1. Diferencias de fases redondeadas entre Matlab y el sistema.

N.° experimento	Cantidad de fases distintas $[nP_{cod_{dif}}]$	Porcentaje de fases distintas $[\%P_{cod_{dif}}]$	Error absoluto máx. $[Error_{cod_{máx}}]$	Error absoluto mín. $[Error_{cod_{máx}}]$	Error medio $[Error_{cod_{med}}]$
5-a	85	33.59	1	0	0.3359
4-a	31	12.25	1	0	0.1225
3-a	13	5.13	1	0	0.0513
2-a	5	1.97	1	0	0.0197
5-b	70	27.66	1	0	0.2766
4-b	35	13.83	1	0	0.1383
3-b	14	5.53	1	0	0.0553
2-b	2	0.79	1	0	0.0790
5-c	56	22.13	1	0	0.2213
4-c	41	16.20	1	0	0.1620
3-c	13	5.13	1	0	0.0513
2-c	8	3.16	1	0	0.0316

De la Tabla 5-1 anterior es posible realizar unas cuantas observaciones: primero, la cantidad de fases que difirieron entre Matlab y el sistema en el peor de los casos fue 85, lo que representó un 33.6 % del total. Por otro lado, el vector de diferencias absolutas tuvo un valor máximo de 1, implicando que las fases redondeadas que no fueron iguales no difireron en más de una unidad (corroborando lo que se estipuló en un inicio, dentro de la sección 3.4).



Gráfica 5-1. Porcentaje de fases distintas entre Matlab y el sistema contra el número de bits.

La Gráfica 5-1 muestra el comportamiento del porcentaje de fases distintas respecto al número de bits. Se puede ver un incremento importante de la cantidad conforme a un aumento de b_a, lo que ocurre debido a que el «margen de redondeo» se vuelve más estrecho; en otras palabras, el error acumulado por el formato numérico del sistema conlleva a que el redondeo lleve a una fase discreta distinta, suceso que se duplica en proporción directa a la cantidad de fases discretas (N).

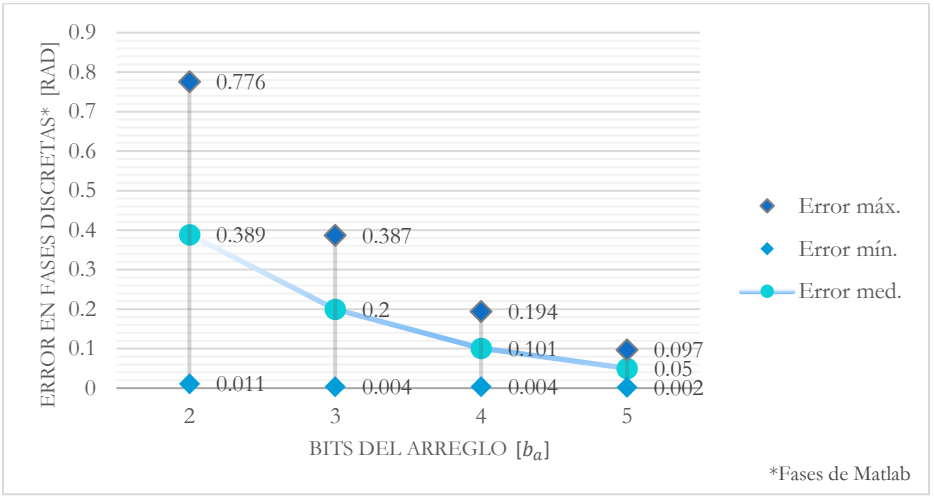
En principio, la tendencia mostrada en la Gráfica 5-1 podría resultar preocupante, pues en $b_a = 5$ se puede afirmar que una de cada tres fases del sistema no coincide con la obtenida por Matlab. Por otro lado, no se debe olvidar que las fases discretas no difieren en más de una unidad.

Para analizar el error en las fases entregadas desde otra perspectiva, en la Tabla 5-2 se colocaron los errores en términos de radianes, es decir, mediante las diferencias de la fase real calculada por Matlab (calculada con la expresión (2.b)) y la fase discreta obtenbida con Matlab y con el sistema ($P_{dc_{err}} = \left|P_{dc_i} - P_{desp_{i_{ml}}}\right|\ [rad]$).

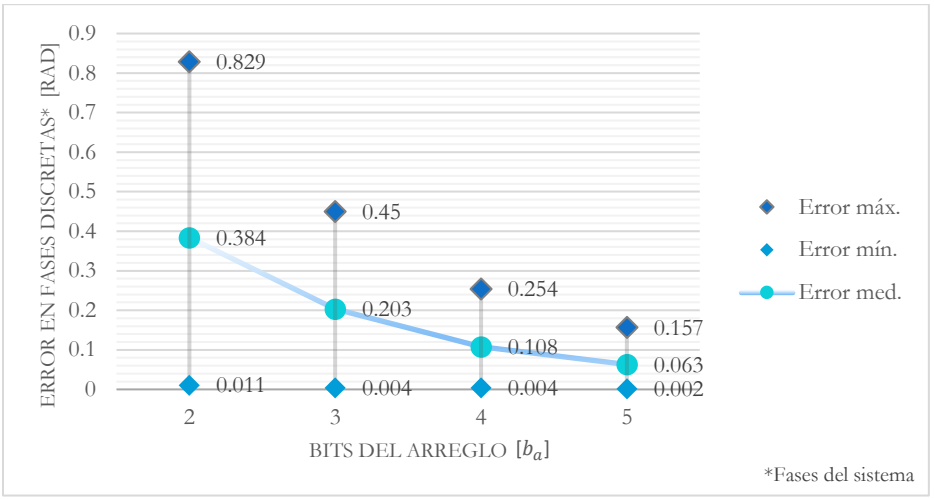
Tabla 5-2. Comparación de errores entre Matlab y el sistema (en radianes).

Experimento	Error máximo [rad]		Error mínimo [rad]		Error medio [rad]		Error cuadrático medio [rad]	
	Matlab	Sistema	Matlab	Sistema	Matlab	Sistema	Matlab	Sistema
5-a	0.0966	0.1517	0.0042	0.0042	0.0523	0.0670	0.0035	0.0062
4-a	0.1921	0.2544	0.0058	0.0058	0.0991	0.1052	0.0127	0.0151
3-a	0.3868	0.4527	0.0058	0.0058	0.2010	0.2038	0.0527	0.0549
2-a	0.7641	0.8390	0.0058	0.0058	0.3850	0.3865	0.1972	0.1995
5-b	0.0979	0.1478	0.0016	0.0016	0.0505	0.0637	0.0033	0.0059
4-b	0.1947	0.2407	0.0051	0.0051	0.0996	0.1069	0.0130	0.0158
3-b	0.3824	0.4469	0.0051	0.0051	0.1979	0.2011	0.0516	0.0541
2-b	0.7803	0.7905	0.0243	0.0243	0.3802	0.3803	0.1947	0.1949
5-c	0.0975	0.1723	0.0001	0.0001	0.0464	0.0583	0.0030	0.0054
4-c	0.1962	0.2667	0.0007	0.0007	0.1032	0.1112	0.0142	0.0173
3-c	0.3920	0.4502	0.0008	0.0008	0.2012	0.2027	0.0527	0.0539
2-c	0.7846	0.8569	0.0042	0.0042	0.4021	0.3847	0.1949	0.1987

De la Tabla 5-2 anterior obsérvese la columna del error máximo del sistema, en el peor de los casos se presentó un error más grande por 0.0748 rad comparado con el de Matlab, lo que comprueba experimentalmente el error máximo establecido dentro del apartado de Elección del formato numérico. Por otro lado, las siguientes dos gráficas Gráfica **5-2** y Gráfica **5-3** contienen los datos promedio de error máximo, mínimo y medio de las fases discretas de Matlab y del sistema respectivamente, con el fin de poder comparar los resultados de forma más intuitiva.



Gráfica 5-2. Error en fases discretas de Matlab contra el número de bits.



Gráfica 5-3. Error en fases discretas del sistema contra el número de bits.

Mediante los dos gráficos Gráfica **5-2** y Gráfica **5-3** anteriores resulta más sencillo comparar las diferencias etre las fases codificadas obtenidas por Matlab y las obtenidas por el sistema. Aunque previamente se vio que los resultados pueden llegar a discrepar incluso en una de cada tres fases (Tabla 5-1), las gráficas demuestran que el error medio, mínimo y máximo entre el valor real y el discreto son bastante parecidos. Se puede asumir que, como se dejó aclarado desde un inicio (al final de la secc. 3.4), las diferencias ocurren en casos en los que el valor real se encuentra muy cercano al límite entre dos discretos, lo que implica que redondear hacia la fase superior o inferior afecte de forma poco significativa.

5.2 ORIENTACIÓN

La sección anterior sirvió para comparar los valores de fase obtenidos por el sistema contra los obtenidos por Matlab, concluyendo que, aunque las fases del sistema pueden llegar a ser hasta un 33 % distintas de las de Matlab, los errores obtenidos por el redondeo no son tan diferentes, lo que desde una perspectiva confirma el correcto cálculo del sistema. En esta sección, las fases son analizadas desde otro punto de vista, utilizando las expresiones de la Tabla 4-6 y el algoritmo de prueba de orientación para verificar que las fases calculadas por el sistema orienten la antena hacia los ángulos deseados.

El algoritmo fue probado para cada esclavo analizado, es decir, para las fases de los 17 elementos de cada dispositivo. Asimismo, se ejecutó con las fases de los 253 elementos, pudiendo obtener resultados de forma «individual» para cada esclavo y para todo el arreglo en cada experimento. Los resultados agrupados por experimento se encuentran dentro de la Tabla 5-3.

Tabla 5-3. Ángulos de orientación experimentales y esperados.

Experimento	Ángulos de orientación obtenidos con algoritmo de prueba de orientación (θ_0, φ_0) [°]	Ángulos de orientación esperados (θ_0, φ_0) [°]
5-a	0.05, 210.25	0.0, 0.0
4-a	0.05, 215.25	
3-a	0.1, 186.75	
2-a	0.15, 286.5	
5-b	15, 180	15, 180
4-b	15, 180	
3-b	15, 180	
2-b	15.15, 180	
5-c	29.95, 180	30, 180
4-c	29.9, 180	
3-c	29.95, 180	
2-c	30.2, 180	

Directamente de la tabla anterior se puede observar que las fases experimentales conllevan a una alta exactitud en los ángulos de orientación, teniendo un error de apenas **0.2°** en el peor de los casos. En contraste, los experimentos con terminación «-a» se observaron diferencias muy grandes entre los valores experimentales y esperados de φ_0. Esto es debido

a que un valor de θ_0 = 0° implica que el vector deberá apuntar directamente en dirección del eje Z, sin importar el valor de φ_0, por lo que para esos experimentos en concreto el único ángulo de interés es θ_0.

Obsérvense las figuras Figura **5-1**, Figura **5-2** y Figura **5-3**, que muestran los vectores generados con los ángulos de orientación de cada experimento. Como referencia se colocó un vector de color gris, cuyos ángulos de orientación son los esperados. Las diferencias visuales en las figuras son tan sutiles que es posible ver encimados a la mayoría de vectores, todos muy cercanos al vector de referencia, lo que indica visualmente una alta exactitud en los ángulos de orientación experimentales.

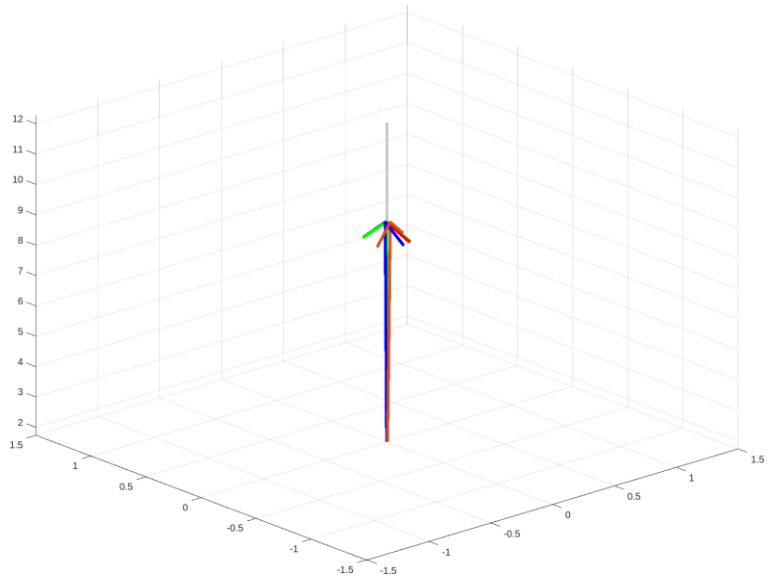


Figura 5-1. Vectores con ángulos de orientación experimentales, el vector gris es la referencia en θ_0=0°, φ_0=0°.

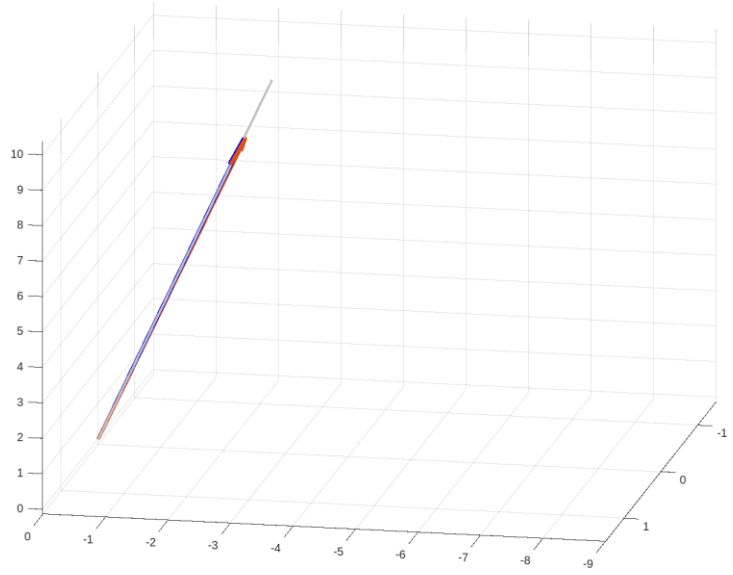


Figura 5-2. Vectores con ángulos de orientación experimentales, el vector gris es la referencia en θ_0=15°, φ_0=180°.

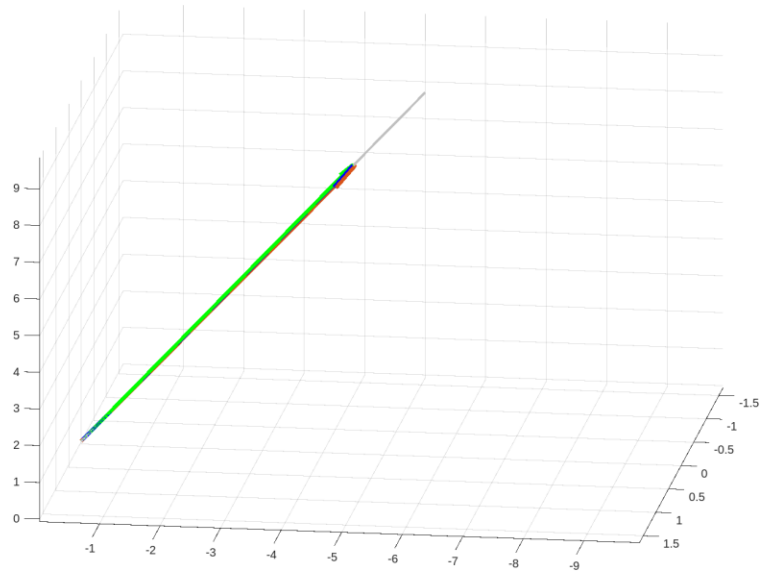


Figura 5-3. Vectores con ángulos de orientación experimentales, el vector gris es la referencia en θ_0=30°, φ_0=180°.

5.3 TIEMPO DE RESPUESTA

El tiempo de respuesta fue obtenido mediante el monitoreo de la señal **reconfig**, fue capturado para cada esclavo analizado de cada experimento, obteniendo 15 valores por cada uno. Los tiempos promedio de cada experimento fueron concentrados en la Tabla 5-4 siguiente, aunque estos solo consideran el tiempo en que la FPGA maestra tarda en calcular y distribuir las fases hacia los esclavos, pero no se toma en cuenta el tiempo adicional de la lógica combinacional del dispositivo esclavo (t_{es})[14].

Tabla 5-4. Tiempo de respuesta promedio para cada experimento.

Experimento	Tiempo promedio de respuesta [µs]
5-a	1.128
4-a	1.11
3-a	1.11
2-a	1.098
5-b	1.099
4-b	1.12
3-b	1.10
2-b	1.11
5-c	1.18
4-c	1.115
3-c	1.097
2-c	1.102
Promedio	1.114
Mínimo	1.097
Máximo	1.128

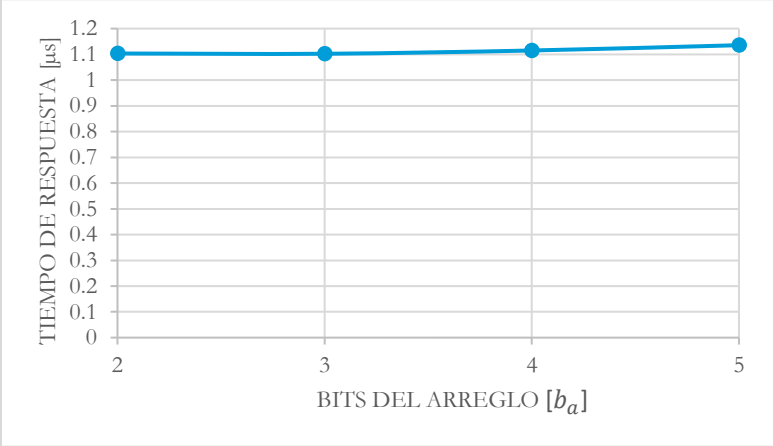
Según lo indica la Tabla 5-4, el tiempo de respuesta se mantiene prácticamente constante. Sin importar los parámetros del experimento en cuestión, la velocidad de respuesta del sistema se mantiene cercana a la velocidad de un microsegundo que se planteó en los requerimientos (secc. 3.1). Además, el tiempo de respuesta de la FPGA maestra (t_{rm}) es muy semejante al que se previó anteriormente con la expresión teórica (4.a).

[14] La latencia del dispositivo esclavo se obtuvo experimentalmente con un analizador lógico. Tuvo un valor máximo t_{es}= 38.4 ns.

Recuérdese que el sistema se trabajó a un 17 % menos de la frecuencia de operación máxima, además de que los esclavos controlaron 3 elementos menos de los 20 máximos establecidos originalmente. Dado que se ha corroborado experimentalmente la validez de la expresión (4.a), se procede a utilizarla para calcular el tiempo de respuesta que la FPGA maestra tendría con la máxima frecuencia de operación ($f_{clkEjec_{máx}} = 65\ MHz$) y la capacidad máxima de e_{es}= 20 elementos por esclavo.

$$t_{rm} = \big(10 + 3(20)\big)(1/65 \times 10^6) = \mathbf{1.077\ \mu s}$$

Para el caso anterior el tiempo se mantiene, de igual forma, muy cercano al microsegundo, lo que indica que el sistema podrá trabajar a una velocidad eficiente incluso en el caso mayor de los establecidos en la sección de Determinación de requerimientos.



Gráfica 5-4. Tiempo de respuesta experimental contra el número de bits.

Se finaliza este apartado con un análisis de la Gráfica 5-4, donde se compara el tiempo de respuesta promedio de los 3 experimentos respectivos para cada valor de b_a. Es notorio que este permanece relativamente constante sin importar la cantidad de bits, lo que concuerda con la decisión de diseño tomada en 3.7.12, donde se optó por tener un tiempo de respuesta invariable para todo b_a, a cambio de tener un tercio de la cantidad total original de núcleos. La Figura 5-4 final muestra el tiempo de operación obtenido por el banco de pruebas para uno de los experimentos.

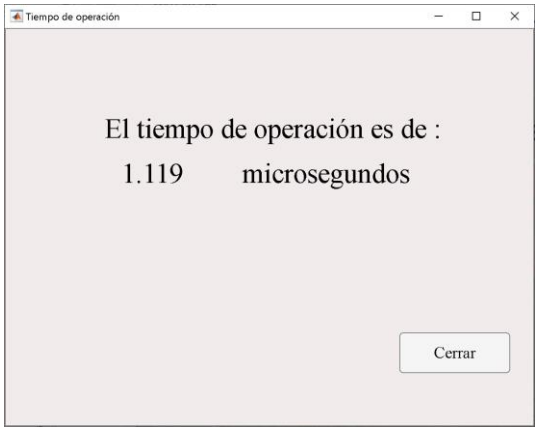


Figura 5-4. Tiempo de respuesta del sistema obtenido por el banco de pruebas para uno de los experimentos.

CAPÍTULO 6
CONCLUSIONES

Se diseñó e implementó un sistema de control digital para antenas en arreglos de fase, encargado de proporcionar las señales de control de polarización para los elementos de arreglos de distintos parámetros y geometrías. La arquitectura del sistema fue diseñada para servir como un modelo escalable y reconfigurable, siendo capaz de funcionar para un rango amplio de parámetros de operación de antenas, cumpliendo con los requisitos que fueron estipulados.

La etapa de experimentación fue llevada a cabo con un módulo prototipo compuesto por dos FPGA y un microcontrolador. Allí se obtuvo un tiempo de respuesta del sistema de aproximadamente 1.2 μs, lo que compite e incluso supera a los trabajos encontrados en el estado del arte, además de que se mostró muy cercano a la velocidad de respuesta deseada. De igual manera, las fases entregadas por el sistema de control, aunque no fueron exactamente iguales a las obtenidas por una alta exactitud numérica como la de Matlab, consiguieron una orientación muy cercana a los valores esperados, con apenas 0.2° de error de orientación en el peor de los casos.

Respecto al diseño del sistema, la arquitectura se implementó bajo una topología tipo «maestro-esclavos»: la FPGA maestra está encargada de todos los cálculos, mientras que las FPGA esclavas sirven como decodificadores y distribuidores de señales de control. Esta alternativa de diseño fue elegida después de una discusión sobre un par de propuestas, donde se estimó que esta ofrecería mayores beneficios. Las decisiones de diseño de cada bloque y componente estuvieron basadas en los conceptos de «escalabilidad» y «reconfigurabilidad», que fueron los requisitos de este trabajo.

En lo que respecta a la reconfigurabilidad, se cuenta una interfaz de usuario que permite configurar de manera sencilla el arreglo que se estará controlando, además de proveer de funciones para corroborar la recepción de datos y los cálculos internos de la primera etapa de funcionamiento. Estas características son de utilidad para que el usuario pueda monitorear y verificar las tareas iniciales que realiza el sistema.

La escalabilidad del sistema de control reluce a través de sus «núcleos» y «dispositivos esclavos», los cuales hacen posible aumentar o disminuir la capacidad de control para arreglos de mayor o menor cantidad de elementos, siempre y cuando la cantidad de recursos lógicos lo permita. Ambos están diseñados para añadirse o removerse de manera relativamente sencilla, sin tener que hacer una mayor alteración del código HDL. Asimismo, es posible variar la capacidad de control de los dispositivos esclavos sin tener que intervenir directamente en el código, con tan solo modificar algunos de sus parámetros.

De igual manera, el controlador está pensado para recibir los ángulos de orientación desde otro dispositivo mediante el protocolo SPI, por lo que puede ser fácilmente integrado a un sistema para trabajar como esclavo, siguiendo la filosofía de que este formará parte de un satélite tipo «CubeSat». Para el mismo propósito, se le añadieron distintas señales tipo bandera, que permiten dar seguimiento de la etapa del funcionamiento en que se encuentra y cuánto demora en su proceso de ejecución.

Otro de los requisitos importantes fue el tiempo de respuesta. Para conseguir una velocidad cercana a la requerida, los componentes lógicos de la etapa de ejecución tuvieron que ser simplificados y optimizados, recurriendo a bloques aritméticos de menor latencia como sumadores tipo CLA, multiplicadores mejorados, así como la implementación de la técnica de paralelismo temporal conocida como «pipeline». Dichas modificaciones, aunque en consecuencia llevaron a un rediseño de la mayoría de bloques que ya se tenían implementados hasta el momento, tuvo un impacto positivo incluso en la etapa de configuración, donde se redujo considerablemente la cantidad de datos que se almacenan en memoria.

Por otro lado, resulta importante mencionar que, aunque el sistema de control satisface los objetivos originales del trabajo y posee varias funcionalidades extra, es posible añadirle ciertas mejoras tanto funcionalmente como también para monitoreo. Las mejoras que fueron pensadas a lo largo del desarrollo se plasman a continuación como posible trabajo a futuro.

- Envío de fases calculadas por el sistema hacia la interfaz de usuario: para verificar que la etapa de ejecución se encuentre entregando resultados correctos, sería útil tener una función que permita enviar las fases obtenidas por la FPGA maestra hacia la interfaz, pudiendo así verificarlas de forma externa. El banco de pruebas utilizado analiza las fases recibidas en el esclavo, mas no las fases en las distintas etapas de cálculo dentro del maestro.

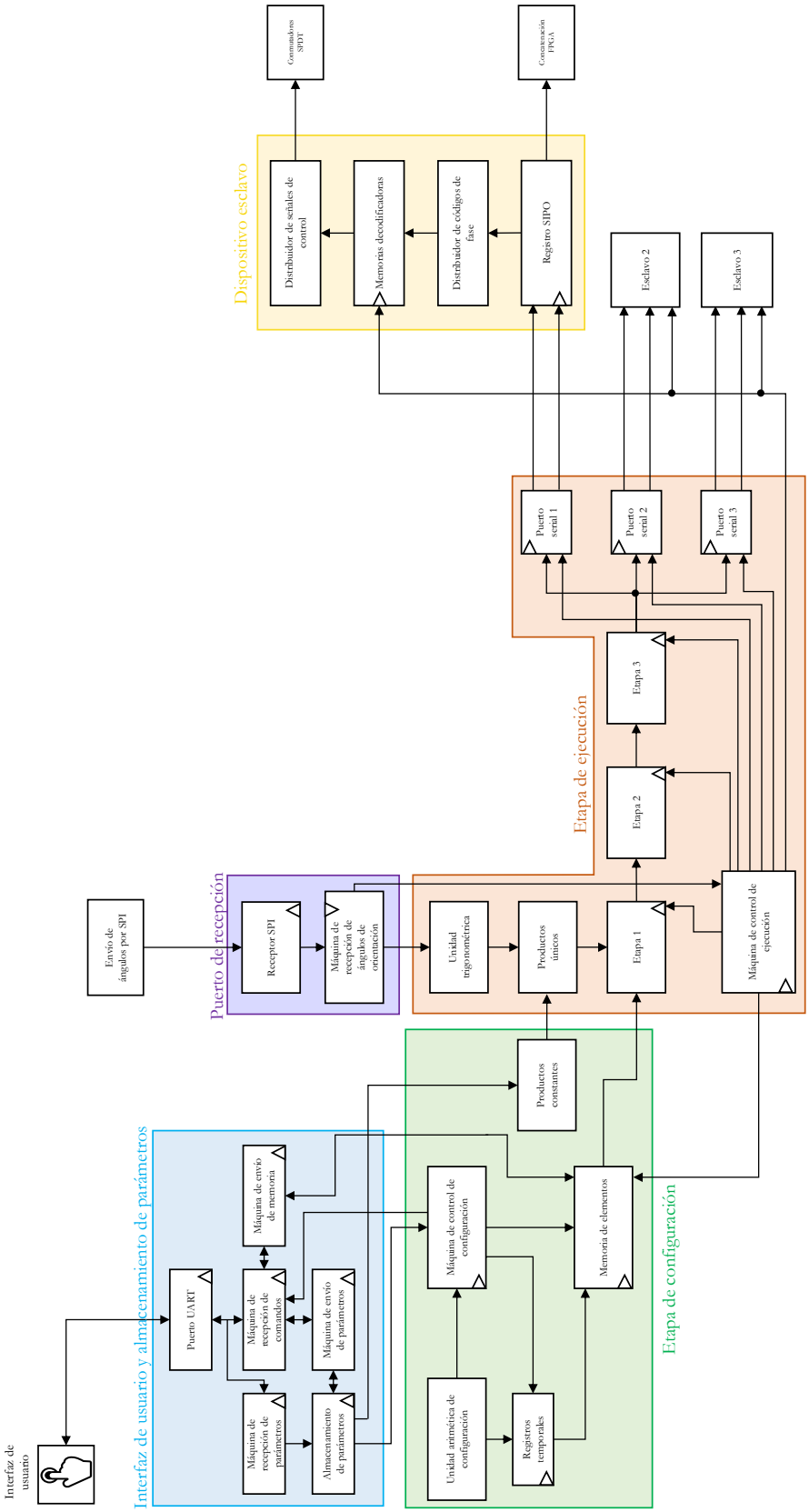
- Carga directa de las fases discretas hacia la FPGA maestra: esta funcionalidad permitiría eliminar el error en los cálculos del sistema, de manera que las fases redondeadas se envíen desde algún otro dispositivo directamente hacia la tarjeta y esta las distribuya hacia los esclavos. Dependiendo de la forma en que se reciban las fases, es posible que el tiempo de respuesta sea más lento, pero sería útil para tener la orientación con la mayor exactitud posible gracias a que los cálculos sean hechos de forma externa.

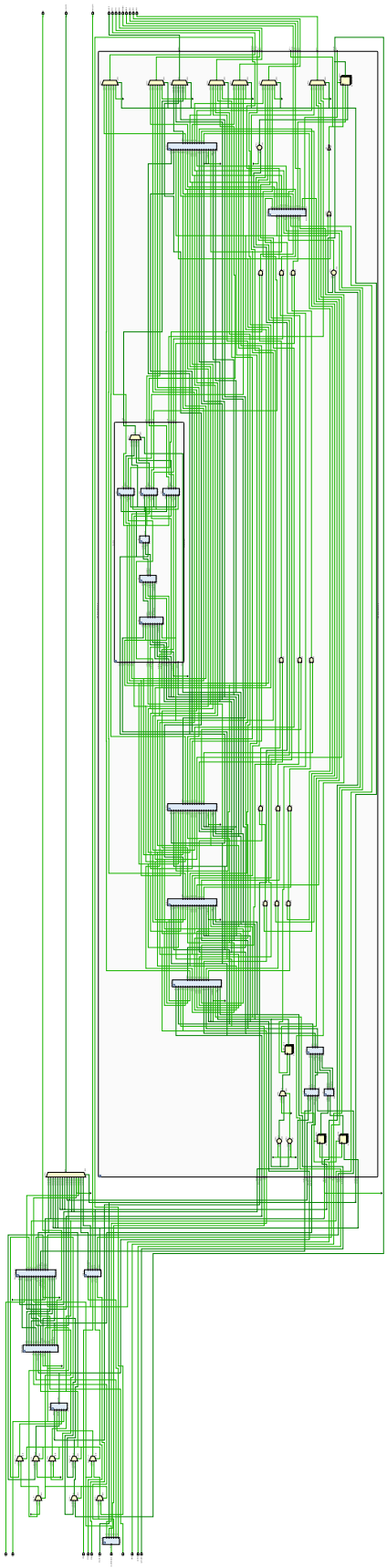
- Rutinas de barrido/escaneo: una funcionalidad que permita al sistema generar sus propios ángulos de orientación, para lograr algún tipo de barrido o escaneo. Esta característica permitiría otorgarle cierta independencia, ya que el diseño actual está hecho para recibir siempre los ángulos de forma externa.

- Tolerancia a fallas: como está pensado para formar parte de una misión espacial, añadir tolerancia a fallas a los módulos más importantes del sistema sería algo indispensable.

Finalmente, se espera que el trabajo a futuro sea retomado en algún momento pues, como se comentó en un inicio, este es uno de los contados documentos que hablan a detalle sobre un sistema de control para tal tipo de antenas. De igual manera, se desea que en un futuro sea posible implementar el sistema completo de forma física para un arreglo real y no solamente un prototipo.

ANEXO 1. DIAGRAMA DE BLOQUES DEL SISTEMA





REFERENCIAS

[1] L. Chapman, «Radio Canadá Internacional,» 12 diciembre 2014. [En línea]. Available: https://www.rcinet.ca/es/2014/12/12/un-11-de-diciembre-de-1901-marconi-y-la-telegrafia/. [Último acceso: 7 diciembre 2020].

[2] F. Y. A. Z. E. Payam Nayeri, «Reflectarray Developments,» de *Reflectarray Antennas: Theory, Designs, and Application*, John Wiley & Sons, 2018, pp. 1-5.

[3] F. Y. S. X. M. L. X. Pan, «Mode analysis of 1-Bit reflectarray element using p-i-n diode at W-band,» EEE International Symposium on Antennas and Propagation & USNC/URSI National Radio Science Meeting, San Diego, CA, 2017.

[4] M. O. R. J. D. S. V. Hum, «An evolvable antenna platform based on reconfigurable reflectarrays,» NASA/DoD Conference on Evolvable Hardware (EH'05), Washington, DC, 2005.

[5] J. H. C. Q. a. L. L. W. Xue, «Design of 1-Bit Digital Coding Reconfigurable Reflectarray Using Aperture-Coupled Elements Controlled by PIN Diodes,» International Conference on Microwave and Millimeter Wave Technology (ICMMT), Chengdu, 2018.

[6] S. W. G. L. S. X. F. Y. Xiaotian Pan, «On-chip Reconfigurable Reflectarray for 2-D Beam-steering at W-band,» IEEE MTT-S International Wireless Symposium (IWS), Chengdu, 2018.

[7] M. J. W. H. Ward Silver, «PIN diodes,» de *The ARRL Extra Class License Manual for Ham Radio*, ARRL, 2008, p. 58.

[8] X. Pan, F. Yang, S. Xu y M. Li, «A 10 240-Element Reconfigurable Reflectarray with Fast Steerable Monopulse Patterns,» *IEEE Transactions on Antennas and Propagation,* vol. 69, nº 9142314 , pp. 173-181 , 2021.

[9] Z. Wang, Y. Ge, J. Pu, X. Chen, G. Li, Y. Wang, K. Liu, H. Zhang y Z. Chen, «1 bit electronically reconfigurable folded reflectarray antenna based on p-i-n diodes for wide-angle beam-scanning applications,» *IEEE Transactions on Antennas and Propagation,* vol. 68, nº 9036087, pp. 6806-6810, 2020.

[10] H. Yang, F. Yang, X. Cao, S. Xu, J. Gao, X. Chen y M. Li, «A 1600-element dual-frequency electronically reconfigurable reflectarray at X/Ku-band,» *IEEE Transactions on Antennas and Propagation,* vol. 65, nº 7902179, pp. 3024-3032 , 2017.

[11] H. Zhang, X. Chen, Z. Wang, Y. Ge y J. Pu, «A 1-Bit Electronically Reconfigurable Reflectarray Antenna in X Band,» *IEEE Access,* vol. 7, nº 8719903, pp. 66567-66575, 2019.

[12] H. Yang, F. Yang, S. Xu, Y. Mao, M. Li, X. Cao y J. Gao, «A 1-Bit 10×10 Reconfigurable Reflectarray Antenna: Design, Optimization, and Experiment,» *IEEE Transactions on Antennas and Propagation,* vol. 64, nº 7448838, pp. 2246-2254 , 2016.

[13] R. L. Schmid, D. B. Shrekenhamer, O. F. Somerlock, A. C. Malone, T. A. Sleasman y R. S. Awadallah, «S-band GaAs FET Reconfigurable Reflectarray for Passive Communications,» *IEEE Radio and Wireless Symposium,* nº 9050009, pp. 91-93, 2020.

[14] A.H. Systems, «Antenna Terms and Definitions,» 2022. [En línea]. Available: https://www.ahsystems.com/notes/antenna-terms-and-definitions.php. [Último acceso: 10 septiembre 2022].

[15] Wireless solutions Mexico, «Tipos de Antenas y Funcionamiento,» 2022. [En línea]. Available: https://www.wni.mx/index.php/wirelesstech/62-antenassoporte. [Último acceso: 10 septiembre 2022].

[16] Cisco, «Coverage area of a meraki access point,» 7 julio 2022. [En línea]. Available: https://community.meraki.com/t5/New-to-Meraki/Coverage-area-of-a-meraki-access-point/m-p/155179. [Último acceso: 10 septiembre 2022].

[17] A. W. Rudge, K. Milne, A. D. Olver y P. Knight, «Gain, Directivity and Efficiency,» de *The Handbook of Antenna Design, Volumen1*, Londres, Peter Perergrinus Ltd., 1982, pp. 19-23.

[18] CENOS, «Simulating a Parabolic Antenna for Satellite Communications,» 7 octubre 2021. [En línea]. Available: https://www.cenos-platform.com/post/simulating-a-parabolic-antenna-for-satellite-communications. [Último acceso: 11 septiembre 2022].

[19] Kurth Electronic, «How do Wi-Fi antennas work and what are they good for?,» 2021. [En línea]. Available: https://www.kurthelectronic.de/fachwissen/how-do-wi-fi-antennas-work-and-what-are-they-good-for/?lang=en. [Último acceso: 11 septiembre 2022].

[20] Kratos, «Radar Antennas,» 2021. [En línea]. Available: https://www.kratosdefense.com/products/space/antennas/radar-antennas. [Último acceso: 11 septiembre 2022].

[21] Starlink, «Starlink Technology,» 2022. [En línea]. Available: https://www.starlink.com/technology. [Último acceso: 11 septiembre 2022].

[22] RadarTutorial, «Parabolic Antenna,» 2022. [En línea]. Available: https://www.radartutorial.eu/06.antennas/Parabolic%20Antenna.en.html. [Último acceso: 12 septiembre 2022].

[23] J. M. Huidobro, «Antenas de telecomunicaciones - Arrays,» *Revista digital de ACTA,* pp. 12, 13, 2013.

[24] P. Delos, «Advanced Technologies Pave the Way for New Phased Array Radar Architectures,» Analog Devices, 2022. [En línea]. Available: https://www.analog.com/ru/technical-articles/advanced-technologies-pave-the-way-for-new-phased-array-radar-architectures.html. [Último acceso: 12 septiembre 2022].

[25] Branch Education, «How does Starlink satellite Internet work,» 19 agosto 2022. [En línea]. Available: https://youtu.be/qs2QcycggWU. [Último acceso: 12 septiembre 2022].

[26] Topper learning, «Phase difference,» 16 octubre 2013. [En línea]. Available: https://www.topperlearning.com/doubts-solutions/mam-what-is-the-difference-between-phase-difference-and-path-differencecan-you-also-explain-me-with-a-example-nyizd488. [Último acceso: 12 septiembre 2022].

[27] K. Benson, «Phased Array Beamforming ICs Simplify Antenna Design,» Analog Devices, 2022. [En línea]. Available: https://www.analog.com/en/analog-dialogue/articles/phased-array-beamforming-ics-simplify-antenna-design.html. [Último acceso: 12 septiembre 2022].

[28] D. Liu, H. Nakano, X. Qing y T. Zwick, «Microstrip Patch Antennas,» de *Handbook of Antenna Technologies*, Springer Reference, 2016, pp. 835-841.

[29] Y. Yang y X. Zhu, «A Wideband Reconfigurable Antenna With 360° Beam Steering for 802.11ac WLAN Applications,» *IEEE Transactions on Antennas and Propagation,* vol. 66, pp. 600-608, 2018.

[30] J. Wang, V. Manohar y Y. Rahmat-Samii, «Enabling the Internet of Things With CubeSats,» *IEEE Antewnnas & Propagation,* nº 1045-9243, p. 8, 2020.

[31] P. Anand, S. Sharma, D. Sood y C. Tripathi, «2012 1st International Conference on Emerging Technology Trends in Electronics, Communication & Networking,» de *Design of compact reconfigurable switched line microstrip phase shifters for phased array antenna*, Surat, India, 2012.

[32] M. L. Overton, «IEEE Floating Point Representation,» de *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, 2001, pp. 17-20.

[33] Digilent, «Verilog HDL: Overview,» 2014. [En línea]. Available: https://learn.digilentinc.com/Documents/241. [Último acceso: 27 marzo 2021].

[34] I. Grout, «HDL Design Entry,» de *Digital Systems Design with FPGAs and CPLDs*, Elsevier, 2008, p. 339.

[35] R. Lipsett, C. Schaefer y C. Ussery, «Behavioral Description,» de *VHDL: Hardware Description and Design*, Kluwer, 1990, pp. 65-67.

[36] Xilinx, «Xilinx product families,» 2020. [En línea]. Available: https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html#productTable. [Último acceso: 21 enero 2021].

[37] Microchip´, «HV5623 Datasheet,» 2019. [En línea]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/HV5623-32-Channel-Serial-to-Parallel-Converter-with-Open-Drain-Outputs-Data-Sheet-20005702A.pdf. [Último acceso: 19 diciembre 2020].

[38] PLDkit, «XMF5 XILINX FPGA MODULE,» 2020. [En línea]. Available: https://pldkit.com/xmf5. [Último acceso: 3 enero 2021].

[39] Newark, «XC7S100-2FGGA484I,» 2020. [En línea]. Available: https://mexico.newark.com/xilinx/xc7s100-2fgga484i/fpga-spartan-7-338-i-o-fpbga-484/dp/17AH9550. [Último acceso: 20 diciembre 2020].

[40] M. Cantrell, «Isolating SPI for High Bandwidth Sensors,» Analog Devices, 2022. [En línea]. Available: https://www.analog.com/en/technical-articles/isolating-spi-for-high-bandwidth-sensors.html. [Último acceso: 21 junio 2021].

[41] Electrical Technology, «Binary Multiplier – Types & Binary Multiplication Calculator,» 2017. [En línea]. Available: https://www.electricaltechnology.org/2018/05/binary-multiplier-types-binary-multiplication-calculator.html. [Último acceso: 28 enero 2021].

[42] J.-M. Muller, «The CORDIC Algorithm,» de *Elementary Functions: Algorithms and Implementation*, Birkhäuser, 2006, pp. 133-140.

[43] T. Sutikno, «An Optimized Square Root Algorithm for Implementation in FPGA Hardware,» *TELKOMNIKA (Telecommunication Computing Electronics and Control),* vol. 8, nº 1, pp. 1-8, 2010.

[44] Xilinx, «Block Memory Generator,» 2021. [En línea]. Available: https://www.xilinx.com/products/intellectual-property/block_memory_generator.html. [Último acceso: 3 febrero 2021].

[45] Python Org, «tkinter — Python interface to Tcl/Tk,» 13 febrero 2022. [En línea]. Available: https://docs.python.org/3/library/tkinter.html#module-tkinter. [Último acceso: 13 febrero 2022].

[46] PythonHosted, «Pyserial documentation,» 2015. [En línea]. Available: https://pythonhosted.org/pyserial/. [Último acceso: 15 febrero 2022].

[47] Xilinx, «Spartan-7 FPGAs Data Sheet,» 9 marzo 2019. [En línea]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds189-spartan-7-data-sheet.pdf. [Último acceso: 22 agosto 2021].

[48] Github, «SPi-slave,» 31 enero 2022. [En línea]. Available: https://github.com/nandland/spi-slave/blob/master/Verilog/source/SPI_Slave.v. [Último acceso: 18 agosto 2022].

[49] K. Sowmya, P. B. Babu y D. Nandan, «Survey on the Impact of FSM Design for High-Performance Architecture Evaluation,» de *Micro-Electronics and Telecommunication Engineering*, Springer, 2019, pp. 439-445.

[50] G. M. Chaudhary y F. Kharbash, «High Performance Fast Multiplier,» de *2008 IEEE Region 5 Conference*, Kansas City, MO, USA, 2008.

[51] K. Raahemifar y M. Ahmadi, «Fast 32-bit digital multiplier,» de *IEEE International Conference on Electronics, Circuits and Systems*, Malta, Malta , 2001.

[52] R. Jain, «Adder with Look-Ahead Carry,» de *Modern Digital Electronics*, McGraw-Hil, 2003, pp. 202-204.

[53] «QMTECH Spartan 7 XC7S15 Core Board,» FPGA Cookbook , 2022. [En línea]. Available: https://www.fpga-cookbook.com/xilinx-development-boards/xilinx-spartan-7-xc7s15/qmtech-spartan-7-xc7s15-ddr3-core-board/. [Último acceso: 24 enero 2022].

[54] Xilinx, «Nexys Video Artix-7 FPGA: Trainer Board for Multimedia Applications,» Xilinx, 2022. [En línea]. Available: https://www.xilinx.com/products/boards-and-kits/1-cfdwic.html. [Último acceso: 13 mayo 2022].

[55] Xilinx, «Basys 3 Artix-7 FPGA Board,» 2022. [En línea]. Available: https://www.xilinx.com/products/boards-and-kits/1-54wqge.html. [Último acceso: 13 mayo 2022].

[56] Texas Instruments, «ARM® Cortex®-M4F-Based MCU TM4C1294 Connected LaunchPad™ Evaluation Kit,» 2022. [En línea]. Available: https://www.ti.com/tool/EK-TM4C1294XL. [Último acceso: 13 mayo 2022].

[57] C. Cappelletti, S. Battistini y B. Malphrus, «The CubeSat Standard,» de *CubeSat Handbook: From Mission Design to Operations*, Elsevier, 2021, pp. 1-3.