UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FíSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

UN SINTETIZADOR DE MÚLTIPLES HACES, BASADO EN FPGA, PARA ARREGLOS DE ANTENAS EN FASE, CON APLICACIONES EN RADIOASTRONOMÍA Y TELECOMUNICACIONES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

FRANCISCO EMILIO CASADO CASTRO

PROFESOR GUÍA:
RICARDO FINGER CAMUS

MIEMBROS DE LA COMISIÓN:
NICOLÁS REYES GUZMÁN
ANDRÉS CABA RUTTE

SANTIAGO DE CHILE
2018

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: FRANCISCO EMILIO CASADO CASTRO
FECHA: 2018
PROF. GUÍA: RICARDO FINGER CAMUS

UN SINTETIZADOR DE MÚLTIPLES HACES, BASADO EN FPGA, PARA ARREGLOS DE ANTENAS EN FASE, CON APLICACIONES EN RADIOASTRONOMÍA Y TELECOMUNICACIONES

La síntesis de haces resuelve el problema de generar patrones de radiación más complejos que los que se puede generar con antenas individuales o arreglos de estas. Para ello se manipula la fase y la amplitud de la alimentación de cada elemento del arreglo, lo que permite focalizar la emisión o recepción de la potencia en ciertos lugares del espacio. Tanto en radioastronomía, como en las telecomunicaciones, esta tecnología da la posibilidad de observar el espacio de manera selectiva y/o en múltiples direcciones de manera simultanea, lo cual permite disminuir tiempos de barrido (para el caso de radiotelescopios) o descongestionar canales creando enlaces directivos.

Este trabajo presenta la implementación de un sintetizador digital de múltiples haces, utilizando un arreglo planar de de 4×4 antenas tipo parche, capaz de generar tres patrones patrones directivos, cuyo procesamiento se realiza íntegramente en una FPGA (*Field Programmable Gate Array*). El arreglo está sintonizado para operar en 5,81 GHz, con un ancho de banda de 40 MHz. La señal es convertida a banda base por un conjunto de tarjetas de electrónica analógica.

Se han realizado dos pruebas experimentales para verificar el funcionamiento del sistema: una emulando condiciones ideales de excitación, de modo de eliminar interferencias ajenas al experimento, y otra situando el arreglo en un extremo de una habitación vacía, siendo iluminado por una fuente puntual en el otro extremo.

Los resultados permiten concluir que la síntesis de haces es satisfactoria, generando haces cuyo lóbulo principal es de 20° de ancho, con la capacidad rotar la dirección en que apuntan en todo el hemisferio visible del arreglo.

A María y Francisco

Agradecimientos

Al laboratorio de Ondas milimétricas: Roberto Fuentes, Diego Benavente, Christopher Muñoz, Valeria Tapia, José Pizarro, Franco Curotto, Camilo Avilés, Ignacio Barrueto, Rocío Molina, David Monasterio, Claudio Jarufe, Rafael Rodríguez, Pablo Astudillo, Francisco Navarrete.

A quienes trabajan en el cerro Calán, especialmente a la Sra. Mari y Sra. Nelly.

A los profesores Nicolás Reyes, Marcos Orchard, Pablo Medina y Ricardo Finger.

A David Torres y Enrique Palacios.

A los ElectroTutores.

A Fernanda Fica, Andrés Caba, Matías Mattamala, Camila Fernández, Rubí Ruiz, Marco Oporto, Vicente Matus, Kenzo Lobos, Ricardo Ramos, Javier Rojas, Ignacio Maldonado, José Ogalde, Miguel Patiño.

Bancamiento eterno a Constanza Villegas, Rodrigo Pérez Dattari, Sebastián Sepúlveda, Gabriel San Martín, Lerko Araya, Diego Alvarado, Lucas Neira, Camilo Saldaña.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Introducción

La radioastronomía ha estado relacionada con las comunicaciones desde su nacimiento, cuando Karl G. Jansky observó por primera vez señales de radio-frecuencia provenientes del espacio, en 1930, mientras estudiaba perturbaciones electromagnéticas que afectaban un enlace de radio transatlántico. Ambas disciplinas han evolucionado y siempre han compartido algunos de sus problemas y soluciones. Actualmente, uno de los desafíos para la próxima generación de radio-telescopios consiste en construir receptores *multi-pixel*. Por otro lado, las comunicaciones inalámbricas apuntan a implementar sistemas de múltiples entradas y múltiples salidas (Multiple-Input/Multiple-Output, MIMO), con el objetivo de implementar antenas inteligentes y proveer la infraestructura para la próxima generación de telecomunicaciones.

Estos problemas tienen una solución común: la síntesis de haces (*beamforming*), una tecnología que hace posible la síntesis de patrones de radiación orientables electrónicamente usando arreglos de receptores y/o transmisores. Mas aún, esquemas de *beamforming* digital (DBF) múltiple permitirían que telescopios de rastreo (*survey telescopes*) aumenten su resolución y disminuyan los tiempos de barrido, así como las comunicaciones inalámbricas mejorarían las tasas de transmisión al tener la posibilidad de multiplexar el espacio. Las ventajas de esta tecnología, en relación a su versión analógica, es que puede ser implementada en hardware orientado a aplicaciones específicas, como arreglos lógicos programables, lo que le otorga rapidez y flexibilidad, a costa de un uso extensivo de recursos. Hoy en día, el principal desafío es diseñar e implementar soluciones de bajo costo y alto desempeño, lo que se está logrando por medio de la migración progesiva de la mayor parte del procesamiento a *hardware* digital.

En este sentido, el Laboratorio de Ondas Milimétricas (millimeter-Wave Laboratory, mmWL) de la Universidad de Chile es un lugar propicio para abordar este tipo de problemas, ya que desarrolla instrumentación astronómica de punta, con aplicaciones desde *back-end* hasta *front-end* analógico y digital, motivado principalmente por la llegada de grandes observatorios al norte de nuestro país, siendo el Atacama Large Millimeter Array uno de los principales. Sumado a esto, recientemente, el laboratorio se ha embarcado en la tarea de transferir el conocimiento científico adquirido y producir aplicaciones cotidianas, como por ejemplo, comunicaciones inalámbricas de alta frecuencia y cámaras de visión de radio, un dispositivo capaz de hacer *imaging* en el espectro de las comunicaciones inalámbricas.

El principal objetivo de este trabajo es implementar un esquema de DBF múltiple para un arreglo en fase receptor de 4×4 antenas, que sea capaz de sintetizar múltiples haces para estudiar fuentes monocromáticas. Paralelamente, se pretende generar una plataforma para

explorar soluciones en instrumentación astronómica en el mmWL, así como también para permitir la colaboración con otras áreas en el estudio de comunicaciones inalámbricas que se desarrollen en el Departamento de Ingeniería Eléctrica.

Estructura del escrito

Los siguientes capítulos de este trabajo detallan el trabajo como sigue: el capítulo 1 resume el marco teórico y el estado del arte de las técnicas de síntesis de haces, con énfasis en implementaciones basadas en FPGA. El capítulo 2 la metodología de pruebas y el *hardware* desarrollado durante este trabajo. Más adelante, el capítulo 3 detalla el diseño digital implementado en la FPGA y el *software* desarrollado para interactuar con la plataforma. Luego, el capítulo 4 muestra los resultados de la implementación y pruebas en condiciones ideales y reales. Finalmente, este escrito cierra con un capítulo de conclusiones, donde se revisan los objetivos alcanzados.

Capítulo 1

Marco Teórico y Estado del Arte

1.1. Arreglos y Filtros Espaciales

Un arreglo consiste en un conjunto de N elementos radiantes distribuidos en una región del espacio en posiciones $\mathbf{p}_n$, con $n = 0, 1, ..., N-1$. Cuando existen ondas propagándose por dicha región del espacio, una señal $f_n(t, \mathbf{p}_n)$ se manifiesta en cada elemento del arreglo, lo que da origen a un vector de señales $\mathbf{f}(t, \mathbf{p})$. Si este vector se procesa con un filtro vectorial $\mathbf{h}(\tau) = [h_0(\tau), \ldots, h_{N-1}(\tau)]$ y luego se suman todas las señales filtradas, se obtiene el sistema que muestra la figura 1.1:

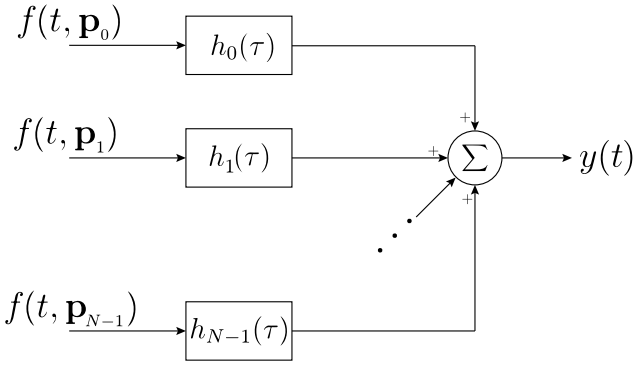


Figura 1.1: Diagrama general de un arreglo para procesar señales.

Si los filtros son lineales e invariantes en el tiempo (LTI), entonces el proceso completo también es LTI. Por lo tanto, en el dominio de la frecuencia, este proceso se expresa como:

$$\begin{aligned} Y(\omega) &= \mathcal{F}\{y\}(\omega), \\ &= \mathcal{F}\{\mathbf{h}\}(\omega) \cdot \mathcal{F}\{\mathbf{f}\}(\omega), \\ &= \mathbf{H}(\omega) \cdot \mathbf{F}(\omega), \end{aligned} \tag{1.1}$$

donde $\mathcal{F}\{\cdot\}(\omega)$ corresponde a la transformada de Fourier. Este resultado es importante, ya que permite diseñar procesadores de arreglos para realizar filtros espaciales, según como se elija $\mathbf{H}(\omega)$.

A continuación se desarrolla un caso sencillo que servirá para introducir notación y conceptos importantes del diseño de procesadores de arreglos. El análisis de arreglos en el caso general se realiza considerando elementos isotrópicos, ya que para el caso en que todos los elementos son iguales, la función de transferencia del procesador de arreglo se puede separar en una multiplicación de dos factores independientes: la respuesta del elemento, que depende de su geometría, y la respuesta del arreglo, que depende sólo de la distribución espacial de los elementos.

Sea el arreglo con elementos localizados arbitrariamente en las posiciones $\mathbf{p}$, que recibe un frente de onda plano, monocromático, cuya dirección de propagación está dada por el vector $\mathbf{a}$, como lo muestra la figura 1.2: Dado que el frente de onda no llega al mismo tiempo a cada

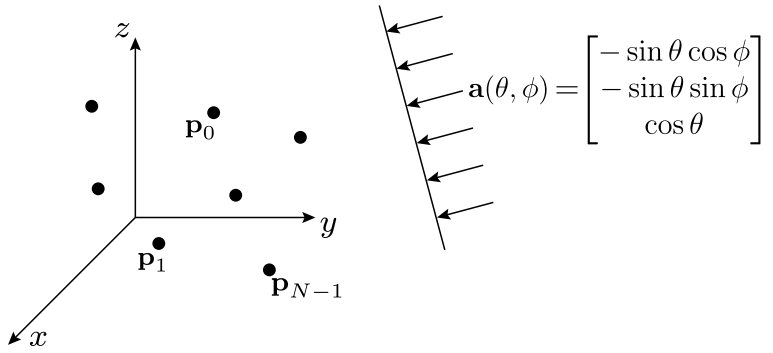


Figura 1.2: Ejemplo simple de un arreglo arbitrario que recibe un frente de onda plano.

elemento, se tendrá un vector de señales $\mathbf{f} = [f(t-\tau_0), \ldots, f(t-\tau_{N-1})]$, donde cada desfase τ_n se calcula como:

$$\tau_n = \frac{\mathbf{a} \cdot \mathbf{p}_n}{c}, \tag{1.2}$$

con c igual a la velocidad de propagación de la onda[1]. Este desfase corresponde al tiempo que demora la onda en recorrer una distancia $\|\mathbf{p}_n \cdot \mathbf{a}\|$ en la dirección $\mathbf{a}$.

En el dominio de la frecuencia, cada componente de $\mathbf{f}$ toma la forma:

$$F_n(\omega) = \mathrm{e}^{-j\omega\tau_n} F(\omega). \tag{1.3}$$

Se define, entonces, el vector de número de onda $\mathbf{k}$ como:

$$\mathbf{k} = \frac{\omega}{c}\mathbf{a} = \frac{2\pi}{\lambda}\mathbf{a}, \tag{1.4}$$

con λ igual a la longitud de onda asociada a la frecuencia ω. Notese que $\|\mathbf{k}\| = 2\pi/\lambda$. Con esto se define el *array-manifold-vector* $\mathbf{v}(k)$ como:

$$\mathbf{v}(\mathbf{k}) = \begin{bmatrix} \mathrm{e}^{-j\mathbf{k}\cdot\mathbf{p}_0} \\ \mathrm{e}^{-j\mathbf{k}\cdot\mathbf{p}_1} \\ \vdots \\ \mathrm{e}^{-j\mathbf{k}\cdot\mathbf{p}_{N-1}} \end{bmatrix}. \tag{1.5}$$

[1] c es la velocidad de la luz para el caso de ondas electromagnéticas en el vacío.

Este vector contiene toda la información espacial del arreglo, independiente de la señal $f(t)$ presente en el medio. Así, se puede escribir $\mathbf{F}(\omega)$como:

$$\mathbf{F}(\omega) = F(\omega) \cdot \mathbf{v}(\mathbf{k}) \tag{1.6}$$

Finalmente, si a las señales de este arreglo se le aplican filtros de la forma:

$$\mathbf{H}(\omega) = \frac{1}{N}\mathbf{v}^*(\mathbf{k}) \tag{1.7}$$

entonces se obtendrá que $y(t) = f(t)$. Este procesador se conoce como *beamformer convencional* o *delay-and-sum beamformer*. Notese que la expresión en el dominio del tiempo de este filtro corresponde a *deltas de Dirac* desplazados, $h_n(\tau) = \frac{1}{N}\delta(\tau + \tau_n)$.

Nuevamente, como el sistema es LTI, al considerar las siguientes funciones base:

$$\begin{aligned} f_n(t, \mathbf{p}_n) &= \mathrm{e}^{j(\omega t - \mathbf{k}\cdot\mathbf{p}_n)} \\ \iff \mathbf{f}(t, \mathbf{p}) &= \mathrm{e}^{j\omega t} \cdot \mathbf{v}(\mathbf{k}), \end{aligned} \tag{1.8}$$

entonces, la respuesta del sistema es:

$$y(t, \mathbf{k}) = \mathbf{H}(\omega) \cdot \mathbf{v}(\mathbf{k}) \cdot \mathrm{e}^{j\omega t}, \tag{1.9}$$

a partir de lo cual se define la *función de transferencia frecuencia-espacio* del procesador del arreglo como:

$$\Upsilon(\omega, \mathbf{k}) = \mathbf{H}(\omega) \cdot \mathbf{v}(\mathbf{k}), \tag{1.10}$$

Esta función de transferencia contiene toda la información, tanto en el dominio de la frecuencia, como en el dominio espacial, por lo que se la puede evaluar a una frecuencia determinada para obtener el *patrón de radiación*, B, del procesador:

$$B(\omega, \theta, \phi) = \left.\Upsilon(\omega, \mathbf{k})\right|_{\mathbf{k}=\frac{2\pi}{\lambda}\cdot\mathbf{a}(\theta,\phi)}, \tag{1.11}$$

1.1.1. Arreglo lineal uniforme

Sea un arreglo de N elementos que se ubican en el eje z de un sistema de coordenadas arbitrario, como el que muestra la figura 1.3 y que la separación entre dos elementos es d. El vector de posiciones está dado por:

$$p_{xn} = p_{yn} = 0, \quad \forall n, \tag{1.12}$$

$$p_{zn} = \left(n - \frac{N-1}{2}\right) \cdot d, \quad n = 0, \ldots, N-1, \tag{1.13}$$

así, el *array-manifold-vector* se escribe como:

$$\mathbf{v}(k_z) = \left[\mathrm{e}^{j\left(\frac{N-1}{2}\right)k_z d}, \mathrm{e}^{j\left(\frac{N-1}{2}-1\right)k_z d}, \ldots, \mathrm{e}^{-j\left(\frac{N-1}{2}\right)k_z d}\right] \tag{1.14}$$

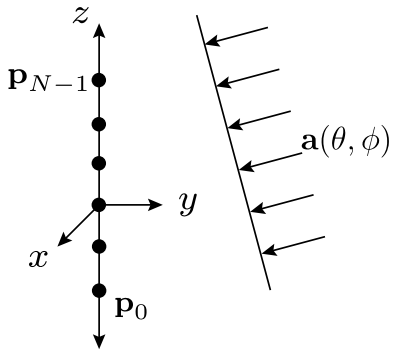


Figura 1.3: Ejemplo de arreglo lineal uniforme.

con $k_z = -\frac{2\pi}{\lambda}\cos\theta$. Es importante notar que el *array-manifold-vector* está sólo en función del ángulo θ, por lo que sólo se podrá filtrar el espacio según elevación, mas no se podrá filtrar según ángulo azimutal. Para filtrar en ambos ángulos se requiere un arreglo bidimensional, lo que se aborda más adelante en la sección 1.1.3, sobre arreglos planares.

Eligiendo filtros iguales a pesos complejos constantes $\mathbf{H}(\omega_c) = \mathbf{w}^* = [w_0^*, \ldots, w_{N-1}^*]$, la función de transferencia del procesador del arreglo queda como:

$$\begin{aligned}\Upsilon(\omega, k_z) &= \mathbf{w}^* \cdot \mathbf{v}(k_z) \\ &= \sum_{n=0}^{N-1} \mathbf{w}_n^* \cdot \mathrm{e}^{-j\left(n-\frac{N-1}{2}\right)k_z d},\end{aligned} \tag{1.15}$$

definiendo $\psi = -k_z d = \frac{2\pi}{\lambda} d\cos\theta$, se tiene:

$$\Upsilon(\psi) = \mathrm{e}^{-j\frac{N-1}{2}\psi} \sum_{n=0}^{N-1} \mathbf{w}_n^* \cdot \mathrm{e}^{jn\psi}, \tag{1.16}$$

Para un arreglo excitado de manera uniforme, se tiene que todos los pesos son iguales $w_n = \frac{1}{N}$, con lo que la función de transferencia se reduce a:

$$\Upsilon(\psi) = \frac{1}{N}\frac{\sin\left(N\frac{\psi}{2}\right)}{\sin\left(\frac{\psi}{2}\right)}. \tag{1.17}$$

1.1.2. Giro de la dirección de máxima respuesta

El problema de girar la dirección de máxima respuesta (MRA, por su nombre en inglés *maximum response angle*) de un procesador de arreglo se puede resolver considerando que se desea una máxima respuesta para $\mathbf{k} = \mathbf{k}_T$, arbitrario. Esto equivale a rotar el vector de señales por medio de la siguiente matriz diagonal de rotación:

$$I(\mathbf{k}_T) = \begin{bmatrix} \mathrm{e}^{j\mathbf{k}_T\cdot\mathbf{p}_0} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathrm{e}^{j\mathbf{k}_T\cdot\mathbf{p}_{N-1}} \end{bmatrix} \tag{1.18}$$

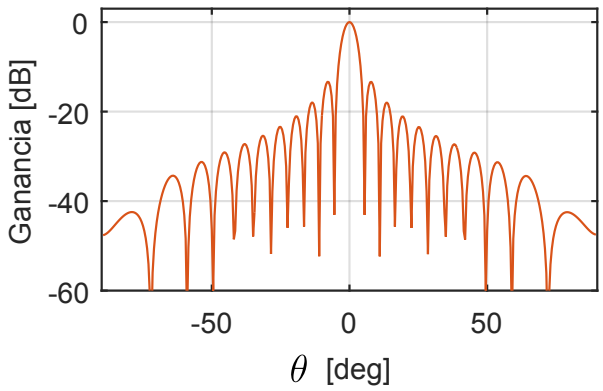


Figura 1.4: Ejemplo de función de transferencia para un arreglo de 21 elementos, $\lambda = 125$ mm y $d = \lambda/2$.

con lo que el vector de señales y la función de transferencia del procesador del arreglo cambian a:

$$\mathbf{f}(t, \mathbf{p}) = \mathrm{e}^{j\omega t}\mathbf{v}(\mathbf{k} - \mathbf{k}_T), \tag{1.19a}$$

$$\Upsilon(\omega, \mathbf{k}|\mathbf{k}_T) = \Upsilon(\omega, \mathbf{k} - \mathbf{k}_T), \tag{1.19b}$$

así, basta escoger una función de transferencia cuyos pesos correspondan al *array-manifold-vector* evaluado en la dirección $\mathbf{k}_T$, de manera que estos compensen la rotación. En el caso de un arreglo lineal uniforme, eso significa que:

$$\begin{aligned} \mathbf{w} &= \frac{1}{N}\mathbf{v}^*(\mathbf{k}_T), \\ \implies B(\mathbf{k}|\mathbf{k}_T) &= \frac{1}{N}\mathbf{v}^*(\mathbf{k}_T) \cdot \mathbf{v}(\mathbf{k}), \end{aligned} \tag{1.20}$$

es decir, el patrón de radiación, que depende de $\mathbf{k}$ alcanza su máximo cuando el producto entre *array-manifold-vectors* se maximiza, lo cual ocurre sólo para $\mathbf{k} = \mathbf{k}_T$

1.1.3. Arreglos planares

Los arreglos planares son aquellos arreglos en que sus elementos se ubican sobre un plano, teniendo dos grados de libertad para disrtibuir los elementos. Esto permite que los procesadores de arreglos también puedan filtrar el espacio con dos grados de libertad, que generalmente corresponden a ángulos azimutales y de elevación.

Un arreglo planar rectangular uniforme tiene la particularidad de separar sus elementos sobre una grilla rectagular de separación d entre dos filas o columnas adyacentes. Este caso se puede abordar como una extensión del arreglo lineal uniforme, ya que se puede procesar primeramente todas las filas, de manera individual, lo que genera un conjunto de señales distribuidas en el eje de las columnas, que también es un arreglo lineal. Esto es válido sólo

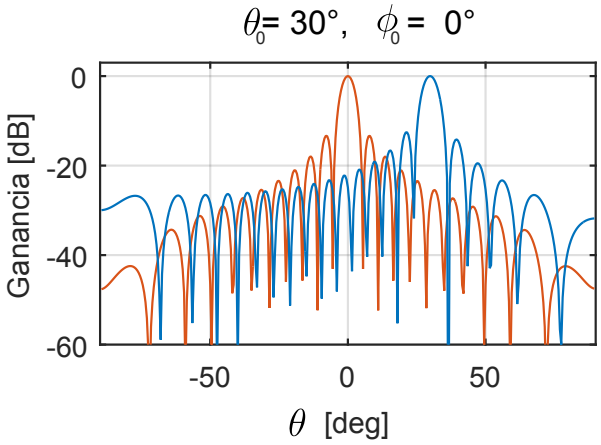


Figura 1.5: Patrón de radiación girado del arreglo. La curva naranja corresponde a la misma de la figura 1.4; la azul corresponde al patrón girado 30° en θ.

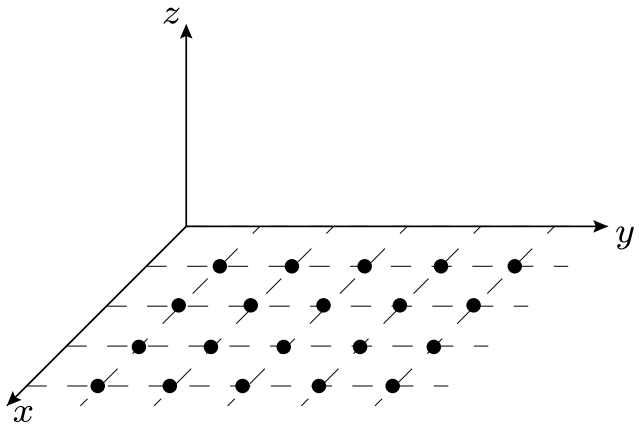


Figura 1.6: Arreglo planar rectangular uniforme.

si el procesador es LTI, pues la linealidad permite sumar en distinto orden los términos del arreglo.

1.2. Antenas

1.2.1. Conceptos y parámetros básicos

Las antenas son un tipo de elemento radiante, ampliamente usadas en la ingeniería de microondas, ya que permiten transmitir o recibir información por medio de campos electromagnéticos que se propagan en el espacio.

Directividad

Uno de los parámetros más importantes de una antena corresponde a la *directividad*, que da cuenta de la dirección (o direcciones) en que se transmiten los campos electromagnéticos. La directividad depende fuertemente de la geometría de la antena. Se habla de una antena *isotrópica* cuando esta no distingue ninguna dirección preferencial, por lo que irradia con la misma intensidad en todas las direcciones. Cuando una antena tiene una o más direcciones de preferencia se dice que la antena tiene *lóbulos*. Según la magnitud relativa de estos lóbulos se habla de lóbulos principales, laterales u otros, según el caso.

Alternativamente al patrón de radiación, se puede caracterizar la directividad espacial de la antena usando la *función de dispersión de punto* (PSF, *point spread function*), que corresponde a la respuesta al impulso espacial de un sistema de generación de imágenes. Dicho de otro modo, es cómo reacciona el sistema al observar en todas las direcciones una cierta región del espacio, considerando una fuente puntual en algún lugar del campo de visión.

Campo cercano y lejano

La distinción entre campo cercano y lejano obedece a que en el primero existen principalmente componentes reactivas del campo, mientras que en el segundo priman las componentes activas, es decir, la transmisión de potencia ocurre en el campo lejano. Es por esto que para caracterizar adecuadamente una antena se debe estar fuera del campo cercano, el cual está delimitado por una distancia tal que:

$$\ell_{\text{near field}} \leq \frac{2D^2}{\lambda}, \tag{1.21}$$

donde D corresponde a la dimensión más larga del elemento. Para el caso de un arreglo planar como el de la figura 1.6, la distancia más larga corresponde a la diagonal entre dos elementos de esquinas opuestas. Esta desigualdad se cumple para antenas *electricamente grandes*, es decir, $D \geq \lambda$.

Parámetros de dispersión de una antena

Por otra parte, las antenas actúan como interfaz entre un medio abierto y un circuito de microondas. En este sentido, una antena también es un dispositivo de microondas, cuyos

parámetros de dispersión relevantes son:

- S_{11}: Razón de potencia reflejada al circuito
- S_{21}: Razón de potencia transmitida desde el circuito hacia el medio

Para una antena que no cuenta con elementos resistivos y sus materiales dieléctricos tienen bajas pérdidas ($\tan\delta$), se puede asumir que transmite toda la potencia que no es reflejada en la conexión al circuito:

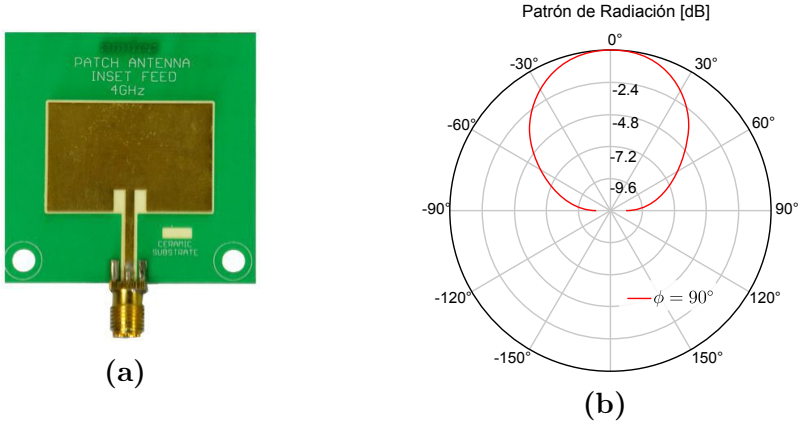
$$|S_{21}|^2 = 1 - |S_{11}|^2, \tag{1.22}$$

por lo tanto, se puede conocer el comportamiento de dicha antena tan sólo con medir las reflexiones en el puerto de entrada. Así, una buena antena debe tener bajas reflexiones en la banda de frecuencias en que se quiere operar.

1.2.2. Antenas tipo parche

Las antenas de tipo parche corresponden a elementos radiantes compuestos por dos conductores planos. Generalmente, uno de los planos corresponde a la referencia (tierra), mientras que el otro tiene una geometría que permite una resonacia entre ambos conductores cuando se los excita con una frecuencia determinada. Actualmente son ampliamente utilizados por su facilidad de fabricación con tecnologías de circuito impreso.

Dado que generalmente el parche es más pequeño que el plano de tierra, éste actúa como reflector, lo que permite que el patrón de radiación sea direccional, limitado a un solo hemisferio. Esta cualidad las hace ser apropiadas para aplicaciones en las que se requiera cubrir regiones más acotadas del espacio.



(a) **(b)**

Figura 1.7: Ejemplo de antena tipo parche (a) y patrón de radiación (b).

En cuanto al ancho de banda, se sabe que si la geometría del parche es simple (por ej.: rectángulos), el ancho de banda fraccional será inferior al 5 % [1]. Otras geometrías más complejas pueden aumentar el ancho de banda.

Estas antenas pueden tener dos tipos de alimentación: micro-cinta y por sonda coaxial. Generalmente se usa la primera por la facilidad de combinar señales de varias antenas en la misma placa en la que se fabrican los elementos, pero esto tiene la desventaja de no poder trabajar con cada señal de manera independiente.

1.3. Mezcladores como convertidores hacia abajo

Un mezclador es un dispositivo de tres puertos que realiza un proceso denominado *conversión de frecuencia*, que consiste en trasladar espectros en el dominio de la frecuencia. Esto se logra con dispositivos no-lineales, como por ejemplo diodos, que permiten multiplicar dos señales en el tiempo, lo cual se traduce en un desplazamiento en el dominio de la frecuencia cuando al menos una de las dos señales corresponde a un tono puro sinusoidal:

$$\begin{aligned} v_{RF} &= \cos(\omega_{RF}\ t + \phi_{RF}) \\ v_{LO} &= \cos(\omega_{LO}\ t + \phi_{LO}) \\ v_{IF} &= v_{RF} \cdot v_{LO} \\ &= \frac{K}{2}\cos\left((\omega_{RF} - \omega_{LO})t + \phi_{RF} - \phi_{LO}\right) \\ &\quad + \frac{K}{2}\cos\left((\omega_{RF} + \omega_{LO})t + \phi_{RF} + \phi_{LO}\right) \end{aligned} \tag{1.23}$$

Generalmente, la respuesta en frecuencia de los dispotivos posteriores filtra las altas frecuencias, operando sólo con la componente de baja frecuencia de la IF, lo que resulta en una *conversión hacia abajo* de la frecuencia. Este proceso está ampliamente presente en los demoduladores de señales en sistemas de telecomunicación, así como también en receptores radio-astronómicos.

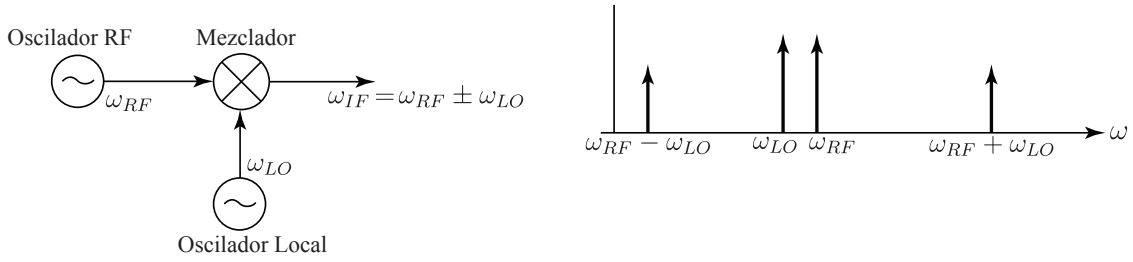
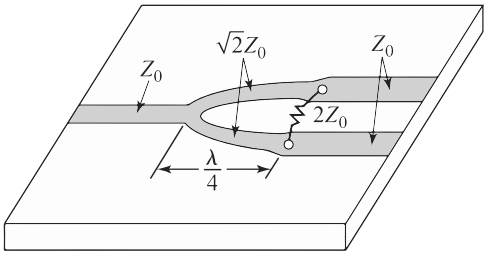


Figura 1.8: Mezclador y conversión de frecuencia.

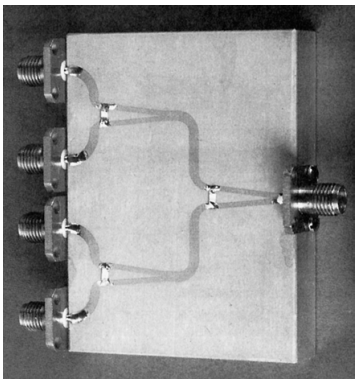
1.4. Divisores de potencia de Wilkinson

En muchas ocasiones es necesario poder distribuir una señal a varios dispositivos, lo que se lleva a cabo con divisores de potencia. Generalmente consisten de líneas de transmisión cuya geometría logra guíar el campo electromagnético por más de un camino, repartiendo la

potencia de entrada entre todas las salidas. Una implementación particular es el divisor de Wilkinson, que tienen la carácterística de ser fácil de fabricar en circuitos impreso, requiere sólo una resistencia de 100 Ω para mejorar la aislación entre los puertos de salida y, por construcción, la diferencia de fase de las señales de salida es cero.



(a)



(b)

Figura 1.9: Divisor de Wilkinson [2]. (a) Modelo del divisor. (b) Divisor de dos etapas fabricado

1.5. Procesamiento digital en el dominio de la frecuencia

En el procesamiento de señales es muy común realizar análisis en el dominio de la frecuencia a través de herramientas como la transformada de Fourier, la cual permite obtener información de la señal que no es tan sencilla obtener en el dominio temporal. En este trabajo son de fundamental importancia las técnicas en el dominio de la frecuencia aplicadas a series de tiempo discretas.

1.5.1. Transformada de Fourier Discreta

La transformada de Fourier tiene su extensión en dominios discretos. Para una señal, debidamente discretizada y cuantizada, $x[n]$, su transformada de Fourier discreta (DFT) está dada por:

$$X[k] = \sum_{n=0}^{N-1} x[n] \mathrm{e}^{j2\pi kN}, \tag{1.24}$$

Este operador conserva varias de las propiedades de su simil continuo, tales como la linealidad. Sin embargo, la resolución espectral que se puede obtener está limitada por dos parámetros: la frecuencia de muestreo y la cantidad de muestras de $x[n]$. Si se tienen N muestras a una frecuencia de muestreo f_s, el ancho de banda y la resolución espectral estarán

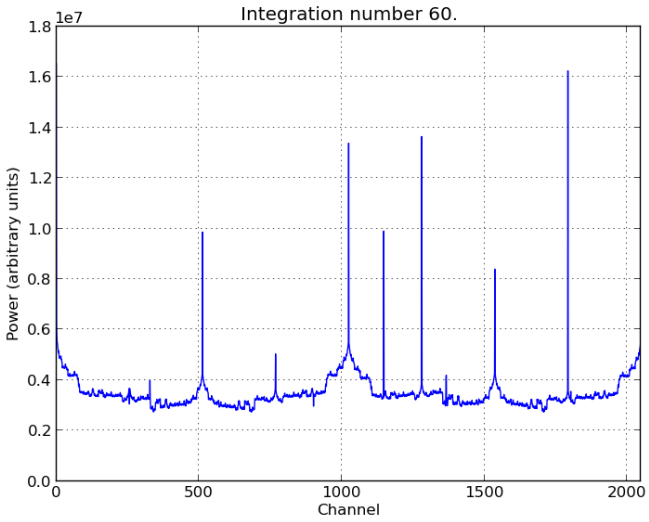


Figura 1.10: Ejemplo de un espectro de 2048 canales [3], de 0 a 400 MHz.

dados por:

$$BW = \frac{f_s}{2}, \tag{1.25a}$$

$$R = \frac{BW}{N/2}, \tag{1.25b}$$

es decir, para N muestras de una señal real, se podrán obtener los primeros $\{X[k]\}_{k=0,\ldots,N/2}$ canales, que corresponden a la parte positiva del espectro.

Un aspecto importante de la DFT es que está contenida en el dominio complejo, lo que posibilita obtener información de la fase de una señal, referenciada a la ventana de datos:

$$X[k] = \mathfrak{Re}\{X[k]\} + j\mathfrak{Im}\{X[k]\} = |X[k]| \cdot \underbrace{\mathrm{e}^{j\phi}}_{\text{fase}}. \tag{1.26}$$

1.5.2. Espectrometría

La espectrometría consiste en analizar la información en frecuencia de una señal, lo que se hace comúnmente con la transformada de Fourier. En radioastronomía resulta de gran importancia, debido a que es una de las técnicas más utilizadas para entender las características de la fuente luminosa que se está observando.

Un espectrómetro es cualquier instrumento que permita visualizar la distribución de potencias en cada uno de los canales que se pueden observar. La figura 1.10 muestra un ejemplo de la información obtenida por un espectrómetro. Generalmente se requiere integrar varias ventanas de un mismo espectro para mejorar la razón de señal a ruido (SNR).

1.6. Estado del arte

1.6.1. Técnicas modernas de *beamforming*

Los dos enfoques más tradicionales son en el dominio del tiempo y la frecuencia. El primero utiliza filtros y respuestas al impulso, mientras que el segundo calcula la transformada de Fourier de las señales y las opera en el dominio de la frecuencia. El dominio del tiempo es preferido en aplicaciones como radares y comunicaciones, mientras que el dominio de la frecuencia se prefiere en espectrometría (radio astronomía).

Cualquiera de las dos implementaciones puede ser analógica, digital o híbrida. La tendencia es trasladar la mayor parte del procesamiento al mundo digital, pero como implica un enorme uso de recursos computacionales, los dispositivos híbridos (por ejmplo [4]) presentan un buen *trade-off* entre rendimiento y costo, entendido como tamaño, precio y consumo de energía.

Indiscutiblemente, las plataformas que obtiene el *throughput* más alto son los *Field Programmable Gate Arrays* (FPGA), los que han sido ampliamente utilizados en la implementación de (DBF) [5, 6, 7, 8, 9].

Un enfoque novedoso [10] consiste en modelar ambos problemas como *multi-dimensionales*, para luego usar herramientas como la transformada de Laplace multi-dimensional. Este enfoque ya cuenta con una publicación reciente [8], que además está implementado en el mismo hardware que se usa en este trabajo.

1.6.2. Parámetros de desempeño

Ancho de Banda

Uno de los parámetros importantes corresponde al ancho de banda resultante, que suele ser acotado para *smart antennas* (cerca del 1 %). Si bien los elementos del arreglo y la electrónica RF limitan el ancho de banda en primera instancia, los efectos del procesamiento también han sido abordados, obteniendo aplicaciones con buen ancho de banda [11, 7, 9].

Steerability

Corresponde a la capacidad de rotar el haz sintetizado y la velocidad a la que se puede efectuar esta tarea. La formación de haces estáticos no presenta mayores dificultades, mientras que uno de los desafíos actuales consiste en poder barrer la mayor parte del espacio posible, en el menor tiempo, de modo que se pueda rastrear terminales y modificar el enlace en tiempo real. Para el caso de las telecomunicaciones, existen implementaciones recientes y satisfactorias en estaciones base, mas en terminales aún falta camino que recorrer [7, 12].

Cantidad de elementos del arreglo

La cantidad y distribución de los elementos del arreglo determina fuertemente las capacidades para formar haces. Los arreglos lineales sólo pueden discriminar franjas del espacio, ya que solo hay un grado de libertad para las fases, mientras que un arreglo planar posee dos grados de libertad para las fases y permite discriminar en una dirección definida, por lo que el proyecto contempla utilizar uno de este tipo. El gran inconveniente es que la cantidad de recursos escala considerablemente, principalmente porque aumenta notablemente el número de elementos. Al respecto existen avances tanto para arreglos lineales [8], como para planares [13, 14].

Capítulo 2

Experimento y *hardware* desarrollado

2.1. Descripción general del experimento

El esquema general del experimento consisten en una antena tipo parche, emitiendo un tono puro de 5.81 GHz (RF), frente a un arreglo de 16 antenas tipo parche idénticas, conectadas a tarjetas electrónicas de *down-conversion* para llevar cada señal del arreglo a 10 MHz (IF). Esta electrónica incorpora internamente etapas de amplificación de alta ganancia programable, lo que posibilita tener la fuente emisora a decenas de metros de distancia. Luego de la conversión hacia abajo, cada una de las 16 señales IF se conecta a una tarjeta de convertidores A/D, adherida a una ROACH2. Los datos generados por la ROACH2 se pueden adquirir con un computador conectado a la red local. Las figuras 2.1 y 2.2 describen el experimento completo y el montaje de los componentes de *hardware*.

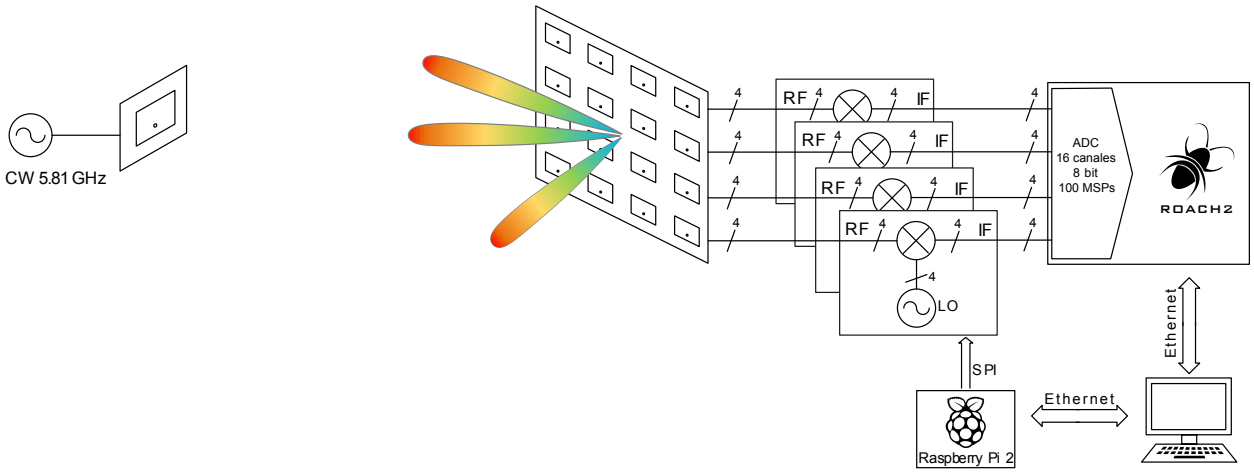


Figura 2.1: Esquema general del experimento.

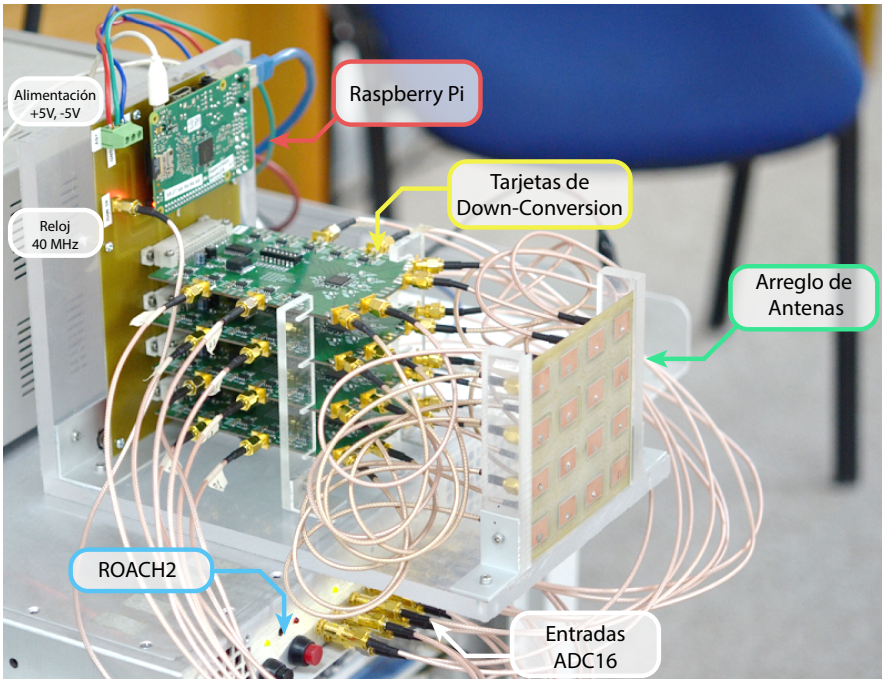


Figura 2.2: Montaje completo del *hardware* de recepción.

2.1.1. Coherencia de fase en el sistema

Dado que la coherencia de fase a través del sistema es de extrema importancia para poder realizar la síntesis de haces de manera adecuada, es necesario tener en cuenta un proceso de calibración que permita anular todos los desfases producidos por el hardware.

En particular, no es posible controlar el desfase producido por los down-convertidores. Un ejercicio sencillo que comprueba esto es que al reconfigurar la electrónica, sin cambiar sus parámetros, algunos convertidores se desfasan en 180°, lo que representa al menos un problema de ambigüedad de fase. Si bien comparten una referencia para el LO, no se puede asegurar que todos los integrados enclavan en la misma fase, ni mucho menos que enclaven al mismo tiempo. Además, la misma fabricación de las tarjetas considera pistas de RF que no son del mismo largo.

Más adelande en la cadena, tampoco se puede asegurar que los integrados convertidores A/D se enciendan y comiencen a muestrear al mismo tiempo. Afortunadamente, se puede calibrar las interfaces *Serializer-Deserializer* (SerDes) que transmiten la información a la ROACH2, de modo que las entradas sean coherentes. Así, sólo es necesario compensar, digitalmente, los errores de fase introducidos por las tarjetas de down-conversión, como se aborda en la sección 3.1.2.

2.1.2. Pruebas del sistema

La primera prueba al *hardware* consiste en emular condiciones ideales de recepción, lo que se logra al inyectar un tono de RF en cada una de las entradas de las tarjetas de *down-conversion*. Para esto se utilizan varias etapas de divisores de potencia coherentes, de modo de dividir un tono en 16 señales de igual amplitud y sin desfase relativo.

Con esta prueba se aislan los efectos producidos por la fabricación del arreglo de antenas, las múltiples reflexiones de la habitación y la intromisión de señales ajenas al experimento (puntos de acceso WiFi, entre otros). Así mismo, se puede caracterizar la ganancia y el desfase de cada uno de los dieciseis canales.

Posteriormente, la segunda prueba agrega el arreglo para poder caracterizar el sistema de forma completa. Dado que en esta situación el sistema es susceptible a la contaminación electromagnética del ambiente, la caracterización debe realizarse en una pieza vacía y de gran tamaño.

En las siguientes secciones se detallan las consideraciones de diseño de cada parte del hardware, su fabricación y la caracterización del desempeño alcanzado por cada una de sus partes. El anexo A contiene las instrucciones para operar el *hardware*.

2.2. Arreglo de Antenas

El arreglo de antenas se compone de 16 antenas tipo parche, idénticas, en una distribución uniformemente espaciada de 4×4 elementos. La elección del tipo de antenas está dada, principalmente, por la direccionalidad (120°) y la facilidad de construcción con tecnologías de circuito impreso disponibles en el laboratorio. Cada elemento está diseñado para funcionar a una frecuencia de 5,81 GHz, con un ancho de banda de $\approx$ 200 MHz a -10 dB, lo que equivale a un ancho de banda fraccional cercano al 3 %. La separación entre elementos es de media longitud de onda, lo que equivale a $d = \lambda/2 = 25{,}86$ mm. La cantidad de elementos se ha definido considerando las limitantes de *hardware* disponible (convertidores A/D de 16 entradas), pero con el objetivo de obtener el mejor rendimiento posible en términos de resolución espacial.

El diseño de las antenas se ha realizado en el software *Ansys HFSS*, considerando un sustrato de FR-4 ($\varepsilon_r \approx 4{,}4$) de 1,53 mm de espesor, con capas de cobre de 35 μm en ambas caras y considerando una alimentación de tipo *probe-fed* coaxial compatible con el estándar *SMA*. La geometría del arreglo se muestra en la figura 2.3; la tabla 2.1 muestra las dimensiones.

Las figuras 2.4 y muestran los parámetros simulados y medidos. Los datos medidos corresponden a uno de los 16 elementos, ya que el arreglo se fabricó en una impresora mecánica, cuya tolerancia es de 50 μm, lo que genera discrepancias en las dimensiones de los elementos. Estas diferencias se muestran en la tabla 2.2, donde la media de la frecuencia de resonancia es $\bar{f_0} = 5{,}824$ GHz.

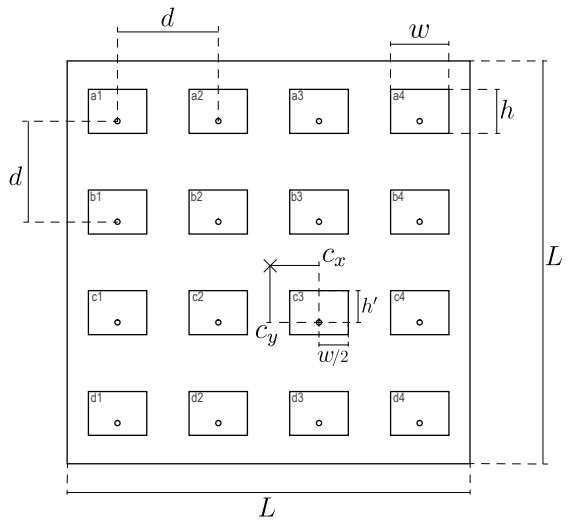


Figura 2.3: Diseño del arreglo de 16 antenas tipo parche. Cada elemento tiene un identificador `a1`,...,`d4`.

Variable		Dimensión [mm]
d	$(\lambda/2)$	25,86
w		15,00
h		11,30
L	$(4d)$	103,44
c_x	$(d/2)$	12,93
c_y	$(c_x + 2{,}5\text{mm})$	15,43
h'	$(h/2 + 2{,}5\text{mm})$	8,15

Tabla 2.1: Medidas relevantes del arreglo.

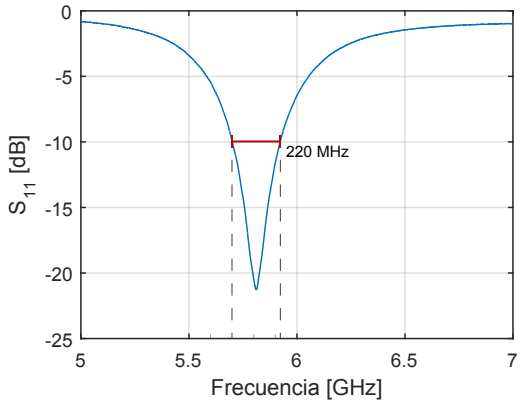


Figura 2.4: Parámetro S_{11} medido de una de las antenas del arreglo.

	A	B	C	D
1	5,822	5,827	5,822	5,835
2	5,840	5,845	5,791	5,870
3	5,822	5,837	5,822	5,848
4	5,802	5,777	5,804	5,832

Tabla 2.2: Frecuencias de resonancia, en GHz, de cada elemento del arreglo.

2.3. Electrónica de Down-Conversión

La *down-conversion* se realiza con cuatro integrados *MAX2851*, pensados para sistemas *MIMO* en la banda ISM de 5 GHz. Cada integrado cuenta con la capacidad de recibir hasta 5 señales RF en la banda de 4.8 GHz a 5.9 GHz y convertirlas en frecuencia base a canales de hasta 40 MHz de ancho de banda, según cómo se configuren sus registros, accesibles con interfaz *SPI*. Además, cada canal RF posee una ganancia de hasta 68 dB, lo que aumenta la sensibilidad del sistema, lo que permite ver señales que se encuentran a decenas de metros de distancia. Una funcionalidad adicional de los integrados es que pueden *up-convertir* una señal de 20 MHz de ancho de banda para transmitirla en la banda ISM de 5 GHz. No obstante, esta funcionalidad no es utilizada en este trabajo.

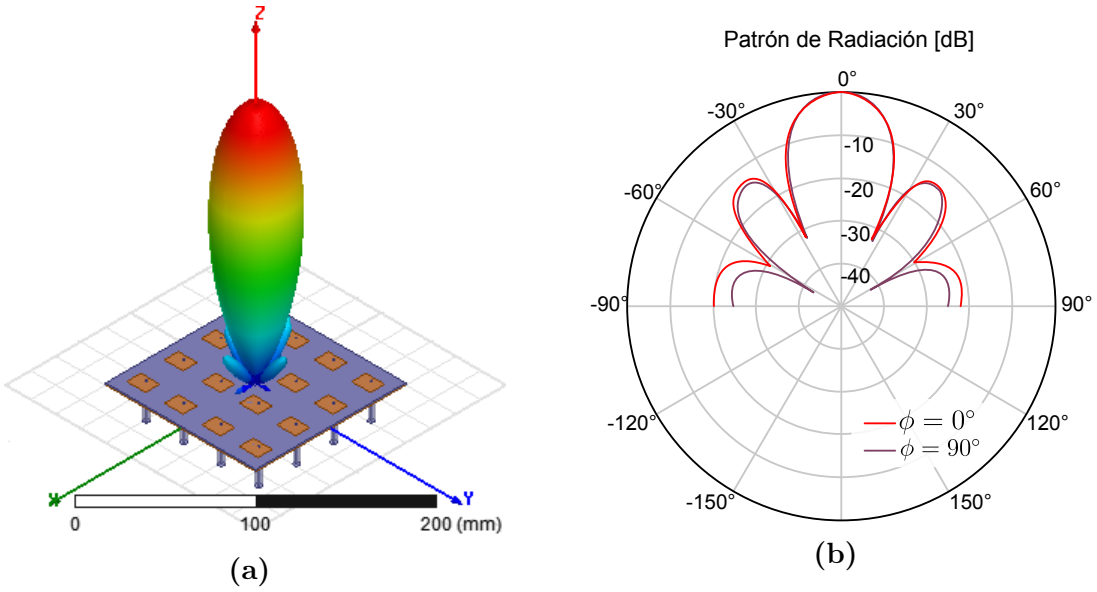


Figura 2.5: Patrón de radiación del arreglo simulado. (a) Vista tridimensional del modelo en *HFSS*. (b) Cortes en planos azimutales ($\phi = 0°$, $\phi = 90°$).

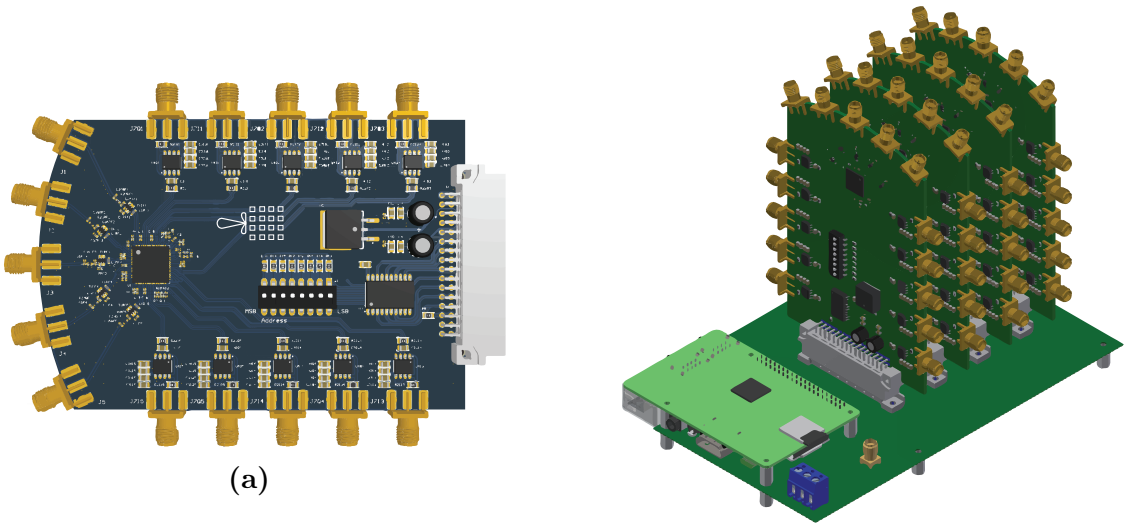
2.3.1. Implementación de tarjetas de *down-conversion* en base al integrado *MAX2851*

El diseño y la fabricación de las tarjetas han sido realizados por Diego Benavente, basándose en el kit de evaluación del fabricante[15]. El anexo G presenta los esquemáticos y gerbers del diseño. Cada integrado está montado en una placa de circuito impreso que cuenta con 5 canales de entradas RF y 5 canales de salida IF en cuadratura (I y Q; 10 puertos de salida en total). Sin embargo, se han escogido solo cuatro puertos RF para las entradas y cuatro puertos I para las salidas. Las tarjetas también incluyen una lógica combinacional que permite agregar una dirección física a cada placa por medio de un *DIP-switch*, de modo que se pueda usar un solo canal SPI y dos líneas de selección de chip para configurarlas con una Raspberry Pi.

Las comunicaciones y las líneas de alimentación de las placas se realizan con una placa de circuito impreso de distribución, cuyas conexiones se muestran en la figura 2.7. Esta placa incluye cuatro conectores *DIN 41612* de 32 pines (J4,J5,J5,J7), un *header* de 40 (2x20) pines para interfacear la *Raspberry Pi* (J7), un conector SMA para un reloj de referencia sinusoidal de 40 MHz (J2) y un cabezal para la alimentación de corriente continua de dos canales: +5V y -5V (J1). Nótese que en los conectores *DIN 41612* cada señal usa pines redundantes; esto se usa para facilitar el ruteo de la PCB de distribución.

2.3.2. *Software* de control

La configuración de los integrados de down-conversión consiste en escribir adecuadamente los registros `main` (32 en total) por medio de la interfaz SPI, según el modo en que se quiera operar. Para esto se ha implementado una clase en Python que facilita la comunicación con



(a) (b)

Figura 2.6: Tarjetas de down-conversion. (a) Diseño de una tarjeta de down-conversion. (b) Cuatro tarjetas montadas en la PCB de distribución, junto a una Raspberry Pi

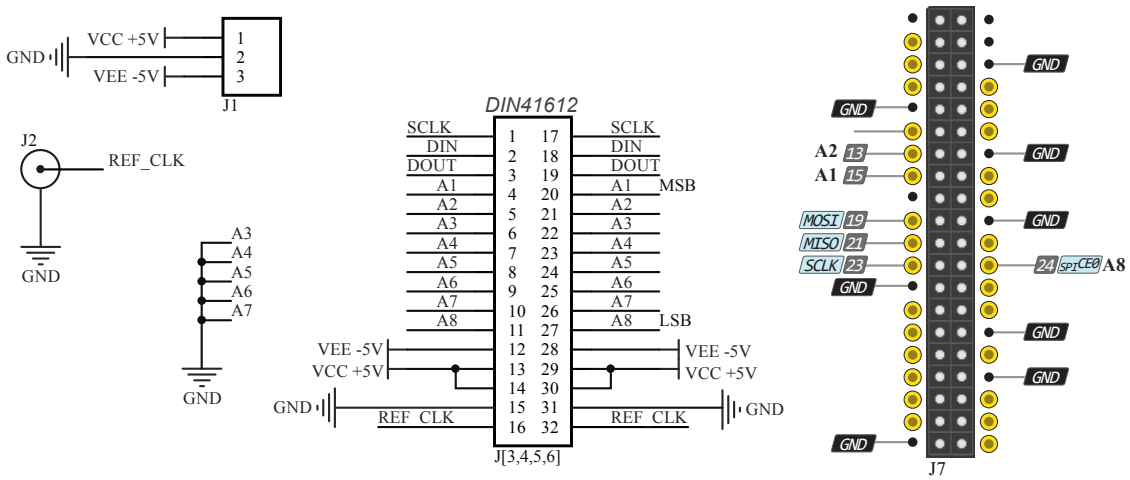


Figura 2.7: Conexiones de la placa de distribución.

las tarjetas de down-conversion, controlando un bus SPI y dos pines GPIO de la Raspberry Pi de la placa de distribución. El código completo de la clase se adjunta en el anexo D.2. La clase contiene dos métodos importantes para la inicialización:

- `Mixer.init_all`: inicialización de las cuatro tarjetas, de manera secuencial, escribiendo el valor por defecto de cada registro `main`, considerando el modo de recepción (*Reveice [Rx] Mode*), según lo indica la hoja de datos [16] (pag. 25).
- `Mixer.calibrate_all`: aplica las configuraciones de ganancias encontradas para disminuir la variación de amplitud entre todos los canales. La siguiente sección contiene más detalles sobre la calibración de amplitudes.

Para diferenciar entre las cuatro tarjetas, la clase `Mixer` maneja internamente dos pines GPIO que seleccionan la *dirección física* de la tarjeta que se quiere configurar (A,B,C,D), haciendo uso del metodo `_select_board`. Una vez lista la configuración y calibración es

necesario liberar el control de los pines de dirección física, para lo cual se implementa el método `Mixer.clean_gpio`.

Para configurar la frecuencia del oscilador local de cada chip - lo cual es necesario para seleccionar la banda de frecuencia de operación - se debe escribir los registros `N` y `F`. El fabricante provee una hoja de cálculo que agiliza el cálculo de los valores de estos registros según las ecuaciones de sus osciladores.

$$f_{LO} = \frac{f_{ref}}{R} \cdot \frac{4}{2} \cdot \left(\mathtt{N} + \frac{\mathtt{F}}{2^{20}} \right) \tag{2.1}$$

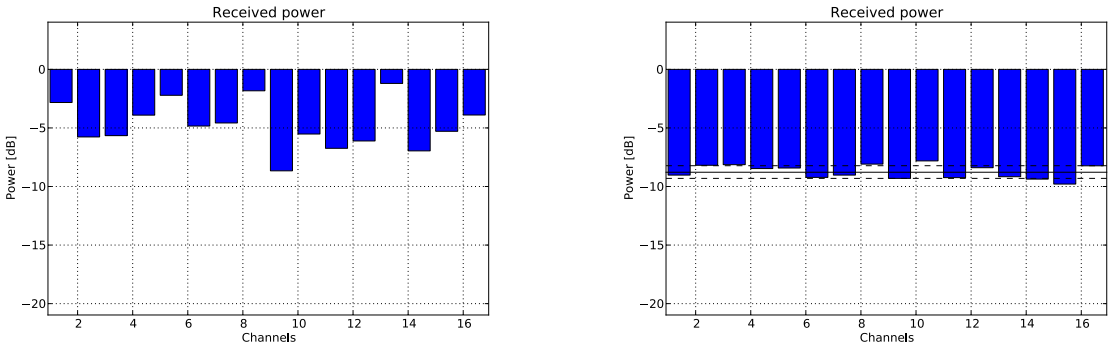
donde $f_{LO} = 5{,}81$ GHz, $f_{ref} = 40$ MHz, $R = 1$. Por lo tanto, los valores de `N` y `F` son

$$\mathtt{N} = (0100\ 1000)_2$$
$$\mathtt{F} = (1000\ 0000\ 0000\ 0000\ 0000)_2$$

2.3.3. Calibración de ganancias

Dado que las tarjetas de *down-conversion* tienen ganancia analógica programable, se hace una calibración gruesa (pasos de 2 dB) de las amplitudes que entregan los mezcladores. Esta tarea podría hacerse en la etapa digital del hardware, con la desventaja de perder resolución (bits menos significativos), por lo que vale la pena sacrificar un poco de sensibilidad del instrumento a cambio.

Utilizando los valores de ganancia máxima (por defecto) se han medido consistentemente las potencias que muestra la figura 2.8a, por medio de los integradores de potencia que posee el modelo del diseño digital (ver sección 3.2.2). La calibración consiste en dejar todos los canales cerca de una potencia que minimice la desviación entre ellos. Para esto se ha implementado el método `set_gain` en la clase `Mixer`, que toma como argumentos el canal a modificar (`a1`,...,`d4`), la ganancia de la etapa RF (`lna_gain`=7,...,0) y la ganancia IF (`vga_gain`=15,...,0). En el anexo se encuentra la tabla D.1, que muestra los valores de `vga_gain` que logran la calibración de la figura 2.8b, manteniendo la ganancia RF al máximo.



(a) **(b)**

Figura 2.8: Efecto de la calibración de ganancias en tarjetas de *down-conversion*. (a) Antes de la calibración. (b) Después de la calibración.

2.3.4. Consumo energético

Al encender la alimentación los consumos de corriente en el canal de +5 V es de 0,33 A, mientras que en el canal de -5 V, el consumo es de 0,21 A. Cuando se configuran los registros para operar en recepción, el consumo de la fuente de +5V sube a 1,8 A. El consumo de la fuente de -5V se mantiene constante.

2.4. Otros componentes y alternativas de bajo costo

2.4.1. Down-convertidores no programables

Dado que el experimento depende fuertemente de la etapa de *down-conversion* y que la implementación de las tarjetas anteriores es compleja, se ha realizado un diseño completamente analógico, no programable y de bajo costo como solución de respaldo. Este comprende una etapa de amplificación RF, un mezclador pasivo y una etapa de amplificación IF, con una ganancia acumulada de 70 dB, lo que permite tener una sensibilidad similar a la de las tarjetas basadas en el chip *MAX2851*.

La *down-conversion* se realiza con un mezclador doble-balanceado *HMC218BMS8GE*, que opera entre 3,5 GHz y 8 GHz y pérdidas por conversión de −7 dB. La amplificación de RF se realiza con dos integrados *SKY65404-31*, cuya ganancia es de hasta 13 dB en la banda de 4,9 GHz a 5,8 GHz, con una figura de ruido de 1,0 dB. Una de sus ventajas es que sólo requieren una alimentación de 5 V , además de contar con un pin que permite apagarlo electrónicamente. La amplificación de IF se realiza con dos amplificadores monolíticos *MAR-8ASM+*, de hasta 31,5 dB, que requiere pocos componentes pasivos para polarizarlo. La figura 2.9 muestra el diseño para la *down-conversion* de una señal. Dado que se necesita replicar

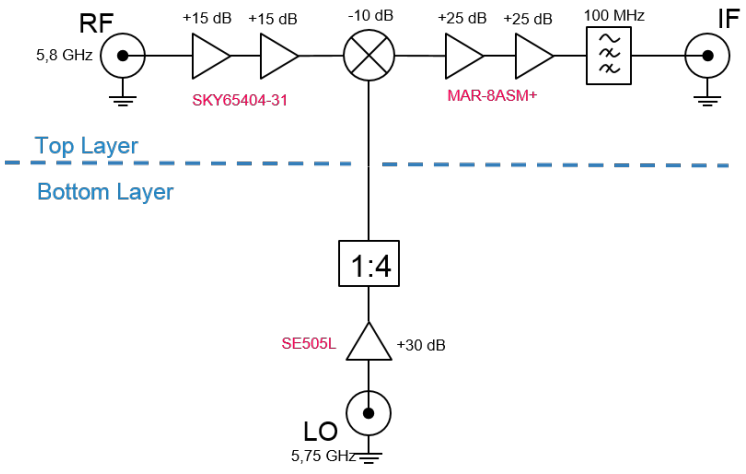


Figura 2.9: Alternativa analógica.

este circuito para las dieciseis señales y se necesita repartir un oscilador local coherente entre todas las placas. Para esto se ha considerado fabricar cinco placas según lo siguiente:

- 4 placas alternativas de down conversión de tres capas que contienen el circuito de la figura 2.9 en la capa superior (*top*), un plano de tierra en la intermedia y en la inferior (*bottom*), un divisor de potencia de Wilkinson para distribuir el LO (una entrada y cuatro salidas) y la alimentación de los cuatro circuitos de la capa superior.
- Una placa de distribución y amplificación, a partir de un divisor Wilkinson y un amplificador de alta potencia *SE5005-L*

Debido al éxito obtenido con las tarjetas basadas en el integrado *MAX2851*, este diseño no se integró, sin embargo, ha permitido evidenciar la importancia y dificultad de la integración para sistemas MIMO.

2.4.2. Divisor de potencia 1:4

Para realizar calibración de fase y amplitud de todo el sistema se necesita alimentar todas las entradas RF del sistema con igual amplitud y de manera *coherente*, esto es, que no haya desfase relativo entre ellas. Una forma de lograr esto es con divisores de Wilkinson. La figura 2.10a muestra el diseño de un divisor, en guía de onda coplanar (CPW, *co-planar waveguide*), de dos etapas, lo que permite dividir la potencia de entrada en cuatro salidas, utilizando tres divisores

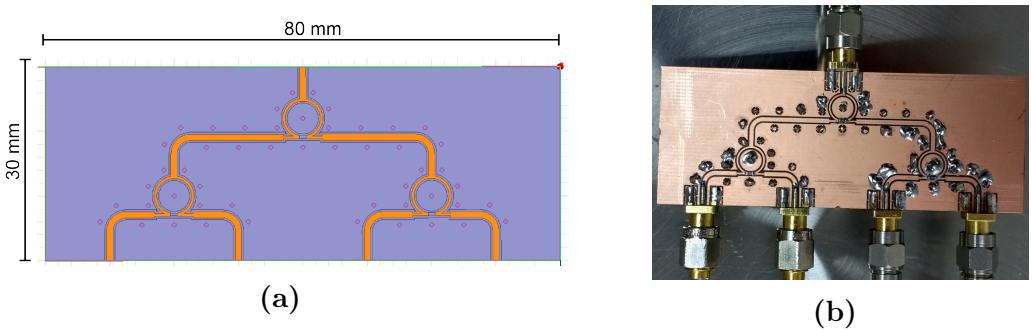


Figura 2.10: Divisor de potencia de dos etapas. (a) Diseño. (b) Divisor fabricado, sometido a pruebas en VNA.

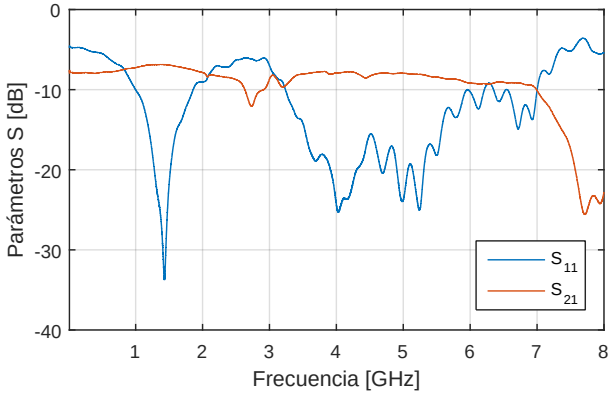


Figura 2.11: Parámetros de dispersión del divisor de dos etapas.

2.5. Electrónica digital

2.5.1. ROACH2

El grueso del procesamiento digital se lleva a cabo con una plataforma *Reconfigurable Open Architecture Computing Hardware* ver. 2 (ROACH2) [17], desarrollada por el grupo *Collaboration for Astronomy Signal Processing and Electronics Research* (CASPER). La ROACH2 se basa en una matriz de puertas programables (FPGA, por su nombre en inglés *Field Programmable Gate Array*), de la serie Virtex-6 SX475T de Xilinx [18]. Junto a esta FPGA se incluye un micro-computador PowerPC 440EPx, para acceder a funciones de control, dos interfaces ZDOK para conectar convertidores A/D de alta velocidad y otros periféricos.

CASPER dispone de librerías en lenguajes de descripción de hardware VHDL y Verilog, además de la opción de usar la interfaz del software Simulink para hacer diseño digital con bloques (de manera gráfica), que en su conjunto extienden las librerías de Xilinx para diseño digital. Los bloques disponibles son principalmente transformadas de Fourier discretas, filtros de respuesta finita, interfaces para memorias RAM, multiplicaciones y otras operaciones básicas optimizadas. CASPER también dispone de las librerías `corr` y `katcp`, escritas en Python, que permiten interactuar con plataformas ROACH2 para adquirir datos, cargar información a las plataformas o cambiar parámetros de los modelos implementados.

La ROACH2 utilizada en este trabajo cuenta con un sintetizador *Valon Synthesizer 5007* de dos canales independientes, programables desde un computador por interfaz USB, con capacidad de generar tonos puros para ser utilizados como relojes de referencia. En este trabajo se ha configurado para generar un reloj de 200 MHz, a una potencia de 6 dBm (nivel 4). La figura 2.12 describe la ROACH2 utilizada en este trabajo.

2.5.2. Convertidor A/D de 16 canales

Tal como se muestra en la figura 2.12, se ha conectado un convertidor A/D a la ROACH en el puerto ZDOK0. Este corresponde a una tarjeta `adc16x250-8 rev. 2`, desarrollada por CASPER, basada en cuatro integrados *HMCAD1511* [19], capaz de trabajar en distintos modos de operación, según cuántas señales se quiera digitalizar y el ancho de banda de estas. La tabla 2.3 resume los modos de operación de la tarjeta y sus anchos de banda asociados. El integrado tiene un rango mucho más amplio, pero debido a las limitaciones del *mixed-mode clock manager* de la FPGA no es posible operar fuera de los rangos de la tabla 2.3.

	Número de entradas			Reloj de referencia
	16	8	4	[MHz]
Tasa de muestreo [MSPs]	67,5 − 120	135 − 240	270 − 480	135 − 240
	135 − 240	270 − 480	540 − 960	270 − 480

Tabla 2.3: Modos de operación `adc16x250-8` [20].

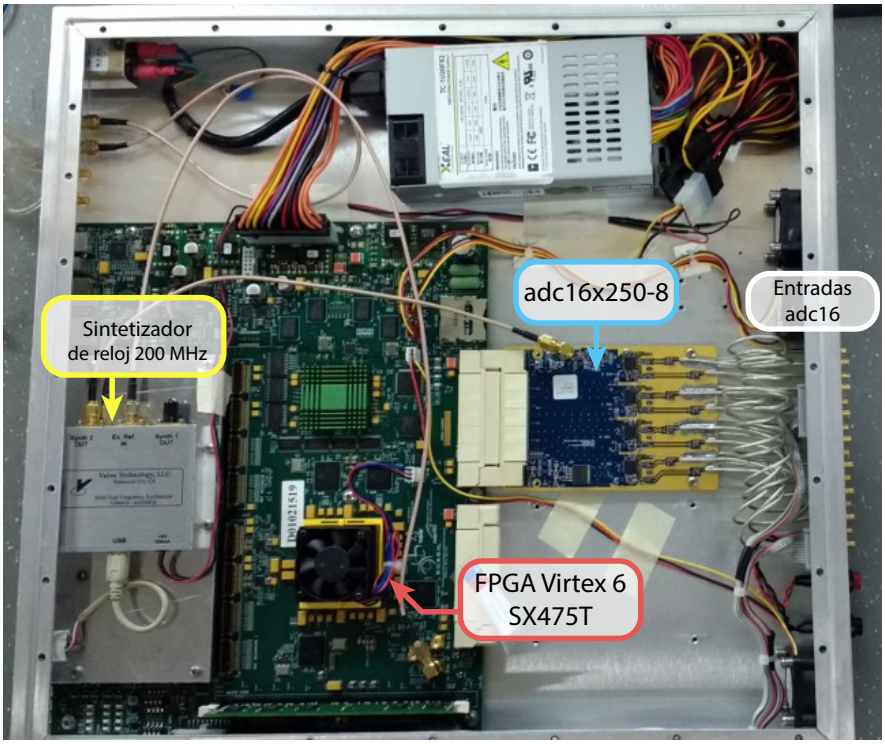


Figura 2.12: ROACH 2 utilizada.

Nótese que un mismo rango de reloj de referencia sirve para los tres modos de operación. Esto es debido a que el *HMCAD1511* cuenta con un *digital clock manager* (DCM) para dividir el reloj y/o administrar la adquisición de varios canales con *interleaving*. No se puede operar en dos modos simultaneamente.

La conversión es realizada en 8 bits con representación en punto fijo con signo 8.7[1] y la máxima potencia de entrada es de +10 dBm. Los datos convertidos son entregados a la ROACH2 por medio de interfaces SerDes de alta velocidad .

Un problema de esta tarjeta es la *ambigüedad de partida*, que provoca que los datos entregados por dos chips no estén necesariamente sincronizados, ya que no se puede asegurar que ambos inicien en el mismo canto de reloj (la diferencia es siempre un múltiplo del periodo de muestreo). Para obtener coherencia entre las señales se deben calibrar los bloques SerDes, lo cual está implementado en una rutina escrita en el lenguaje de programación Ruby, escrita por David Macmahon[2] en el archivo `adc16_init.rb`. Esta rutina también coordina una secuencia de inicialización de los chips, ya que se deben encender por software, además de configurar el modo (número de entradas) en que se debe operar el integrado. Adicionalmente a la rutina de inicialización existen otras, también en Ruby, que permiten chequear que el dispositivo está funcionando, por medio de gráficos de la señal adquirida e histogramas.

Para usar el `adc16x250-8` basta su *yellow block* `adc16-250` en Simulink.

[1]La notación en punto fijo `M.N` describe el largo de la palabra, `M`, y el bit en que se encuentra el punto decimal, `N`.

[2]Disponible en el repositorio `https://github.com/FranciscoCasado/calan-mbf/tree/master/utils/adc16_test/src/bin`

Capítulo 3

Diseño del sintetizador e implementación digital

3.1. Arquitectura del sintetizador de haces

La arquitectura del sistema digital consta de cuatro etapas: adquisición de datos, transformación al dominio de la frecuencia, calibración de fase y procesador del arreglo (ver figura 3.1). Entre cada una de las etapas existen *sondas* (`probes`) que permiten leer la información que está siendo procesada, así como también se dispone de *acciones* (`actions`), bloques que reciben información desde el usuario para variar parámetros importantes del sistema, tales como fasores de calibración y fasores de síntesis de haces. En el anexo F se adjunta la vista principal del esquemático en Simulink de todo el sistema.

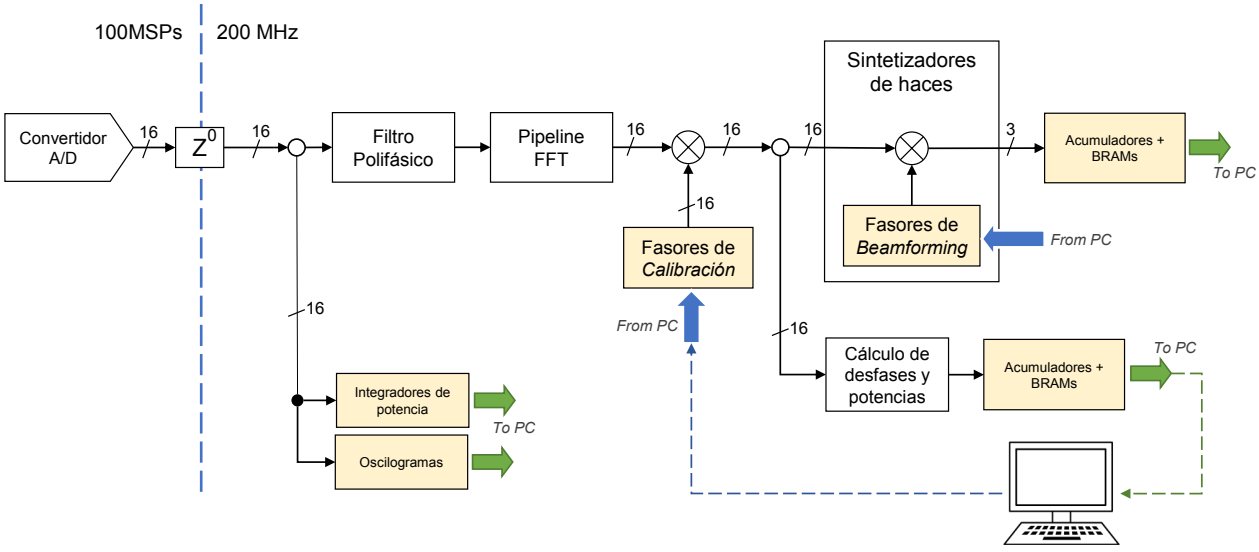


Figura 3.1: Arquitectura del sistema.

Se ha optado por una técnica de síntesis en el dominio de la frecuencia como la descrita por J. Porcello [7], ya que producir cambios de fase en el dominio de la frecuencia implica sólo una multiplicación compleja por cada señal y es de tanta resolución como bits tenga la representación de ella, a diferencia de las técnicas temporales, en que calibrar fase requiere filtros más complejos de implementar. Por otra parte, los desarrollos de CASPER aprovechan implementaciones con *pipelining* de la transformada rápida de Fourier (FFT), para aprovechar el máximo throughput de la FPGA.

3.1.1. Conversión A/D y transformación al dominio de la frecuencia

Tal como se muestra en la figura 3.1, cada señal *down-convertida* es digitalizada por el convertidor `adc16x250-8` a una frecuencia de muestreo correspondiente a la mitad de la frecuencia del reloj del sistema[1], lo que hace que en el sistema se manifieste un retentor de orden cero (*zero-order hold*). Posteriormente, la señal es filtrada con un filtro polifásico, con el objetivo de mejorar el rechazo fuera de la banda y luego se calcula su FFT, considerando un tamaño de 256 muestras/canales.

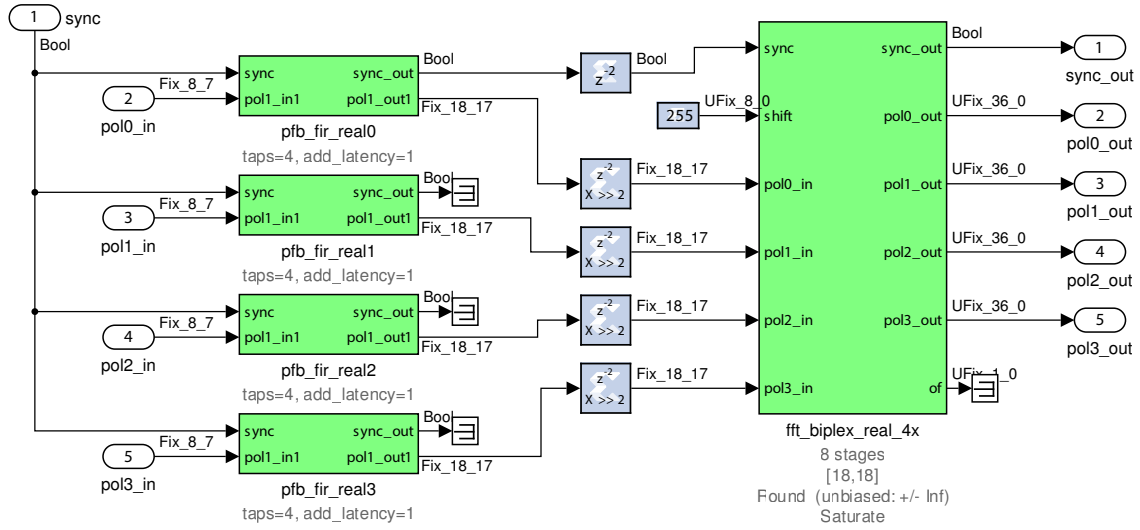


Figura 3.2: Transformada rápida de Fourier y filtro polifásico implementados. Se muestran sólo cuatro señales, ya que la implementación completa usa 16 `pfb_fir_real` y 4 `fft_biplex_real_4x`, separadas en cuatro bloques.

El filtro polifásico está implementado con bloques `pfb_fir_real`, que reciben señales reales de 8.7 bits en punto fijo. Se procesan cuatro ventanas y se ajusta el tamaño de las señales resultantes a 18.17 bits. Inmediatamente a la salida existen bloques `shift` que dividen el resultado por 4 para mantener las señales dentro del circulo unitario (módulo menor a 1), lo que previene que se produzcan *overflows* en las siguientes etapas.

La FFT está implementada con bloques `fft_biplex_real_4x`, que reciben entradas reales de 18.17 bits con signo. La salida corresponde a señales complejas 36.0[2] bits. El hecho de que

[1]Referirse a la tabla 2.3, considerando el modo de 16 entradas

[2]Una señal compleja está compuesta por dos señales reales de 18.17 bits

la transformada sea *biplex* permite ahorrar recursos. También se ha considerado hacer `shift` en cada etapa interna de la FFT, de modo de prevenir *overflows*.

Es importante destacar que las FFTs están implementadas con *pipelining*, por lo que el resultado del cómputo se entrega de manera secuencial, esto es, en cada ciclo de reloj se entrega un nuevo canal del espectro asociado a la ventana muestreada. Esta implementación es la habitual en las librerías de CASPER, ya que evita que los recursos dela FPGA estén ociosos y disminuye el número de *DSP-slices* a utilizar.

Esto también tiene la ventaja de poder sincronizar la salida de los espectros con la lectura de una memoria RAM, cuyo contenido sea la función de transferencia de algún filtro, de modo que en cada ciclo de reloj se multiplica un canal del espectro por un ponderador. El resultado es una implementación sencilla de filtros en el dominio de la frecuencia.

El efecto de retener la muestra por un ciclo de reloj se traduce en que por cada ventana de la FFT se obtendrán cuatro zonas de Nyquist, en lugar de dos (ver figura 3.3). Es decir, la cantidad de canales útiles de la FFT se reducen a un cuarto del total de muestras.

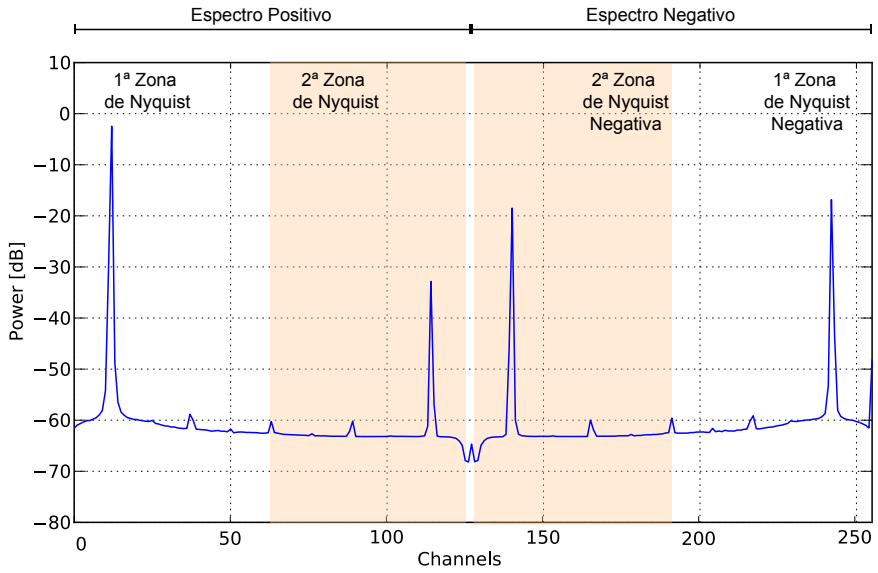


Figura 3.3: Efecto de la retención de muestras en los espectros. La figura muestra un espectro de 256 canales (parte negativa incluída), desde 0 hasta 100 MHz, ante una entrada sinusoidal de 10 MHz.

Por lo tanto, la cantidad de canales útiles de la FFT son sólo 64, en lugar de 128, como en una FFT que reciben entradas puramente reales. Sin embargo, la resolución espectral, de acuerdo a la ecuación (1.25b) es igual a:

$$R = \frac{200 \text{ MHz}/4}{N/4} = \frac{50 \text{ MHz}}{64} = 0{,}78125 \text{ MHz} \tag{3.1}$$

3.1.2. Calibración de fases

Como ya se ha mencionado antes, la coherencia en fase es condición necesaria para realizar la síntesis de haces de manera adecuada. Para lograr la coherencia se ha agregado un multiplicador a la salida de cada FFT, de modo de multiplicar cada señal por un fasor constante, configurable por el usuario.

Teniendo en cuenta la ecuación (1.26), se puede calcular el desfase entre dos señales haciendo el producto conjugado entre ellas:

$$\begin{aligned} \hat{A} &= A\mathrm{e}^{j\phi_A}, \\ \hat{B} &= B\mathrm{e}^{j\phi_B}, \\ \implies \hat{A}\cdot\hat{B}^* &= AB \underbrace{\mathrm{e}^{j(\phi_A-\phi_B)}}_{\text{diferencia de fase}} \end{aligned} \tag{3.2}$$

Si a cada señal $\hat{B}$ se la multiplica por la diferencia de fase de la ecuación (3.2), entonces quedará con la misma fase que la señal $\hat{A}$:

$$\hat{B}_{\text{calibrated}} = \hat{B}\cdot\mathrm{e}^{j(\phi_A-\phi_B)} = B\mathrm{e}^{j\phi_A} \tag{3.3}$$

Dentro de la FPGA se han implementado estas multiplicaciones, cuyos factores corresponden al espectro de la señal `a1` y el espectro conjugado de alguna otra señal. Esto se repite para todas las señales, de modo que cada uno de los diesciseis productos exhibe directamente la diferencia de fase entre cada señal y la señal `a1`, para todos sus canales espectrales. Esta operación es equivalente al sistema de la figura 3.4.

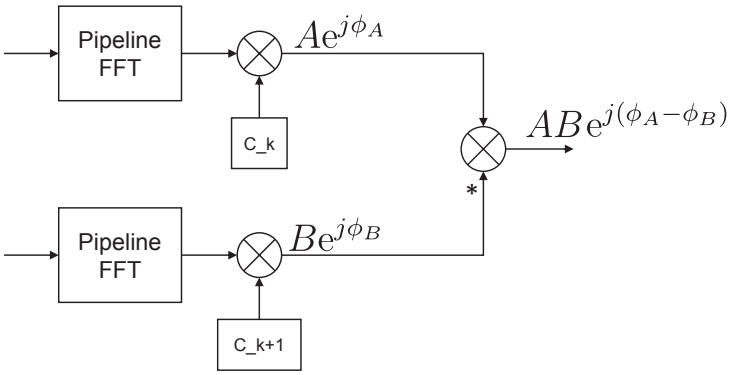


Figura 3.4: Esquema del cálculo de desfases.

Los productos resultantes son acumulados y el resultado de cada acumulación es almacenado en bloques `bram`. Los bloques responsables de estas operaciones son las sondas `cal_probe`[3]. Una vez extraídos estos datos, se procesan con una rutina en Python que normaliza las diferencias de fase obtenidas para: 1) mantener una amplitud igual a la de la señal `a1` y 2)

[3]Explicada en detalle en la sección 3.2.1

mantener todos los fasores dentro del círculo unitario[4], de modo de evitar *overflows*. Así, cada coeficiente de calibración se calcula como:

$$\hat{C}_i = \frac{\hat{X}_{\mathtt{a1}} \cdot \hat{X}^*_{\mathtt{i}}}{|\hat{X}_{\mathtt{a1}} \cdot \hat{X}^*_{\mathtt{i}}|} = \mathrm{e}^{j(\phi_{X_{\mathtt{a1}}} - \phi_{X_{\mathtt{i}}})} \tag{3.4}$$

La ventaja de esta forma de calibración es que puede corregir desfases hasta la mínima resolución angular del sistema, a diferencia de otras técnicas con colas de sincronización de muestras. Sin embargo, tiene el problema de no poder corregir más de $\pm 180°$, ya que la periodicidad de la multiplicación compleja evita agregar (o quitar) más fase. Esto no significa un gran problema para hacer *imaging*, pero sí para demodulación de comunicaciones moduladas por fase (PSK), entre otras. El máximo potencial se logra usando ambas soluciones.

Dado que el ancho de banda fraccional es menor al 1 % (40 MHz << 5,8 GHz), se puede asumir que la calibración para el canal de 10 MHz también es válida para los demás canales inferiores a 40 MHz [11].

3.1.3. Procesador del arreglo

El procesador del arreglo se compone de tres `beamformers` que replican la estructura presentada en la figura 1.1. Se multiplica cada señal por un fasor unitario, para luego sumarlas todas. Dado de que la suma aumenta el tamaño de la señal en cuatro bits (18.17→ 22.17), se agrega un `shift` de cuatro bits que mantiene la representación de 18.17 bits. Esto también tiene el efecto de normalizar la salida de cada `beamformer` a la amplitud de las entradas. Todas las multiplicaciones están implementadas con `dsp48e` y cada una requiere dos, ya que se trata de una multiplicación compleja. La suma de las señales se realiza con dos bloques `adder_tree` de CASPER, uno para la parte real y otro para la parte imaginaria.

Esta implementación tiene la ventaja de realizar síntesis de haces usando cualquiera de las técnicas en el dominio de la frecuencia presentes en la literatura [21], ya que los fasores son independientes entre ellos y totalmente ajustables a la necesidad del usuario. Así mismo, la posibilidad de asignarle direcciones permite tener múltiples sintetizadores de haces independientes.

Al igual que en la calibración, los fasores se almacenan en un banco de fasores y se escriben por software, a través de una lógica que permite discriminar entre cada uno de los `beamformer` según la dirección con la que se configuren al momento hacer el diseño en Simulink. La sección 3.2.4 describe la lógica en detalle.

La figura 3.5 muestra un esquema del bloque implementado.

[4]Módulo inferior a 1. En representación 18.17, esto corresponde a $\left|\mathrm{e}^{j\phi}\right| \leq 2^{17} - 1$ en la notación entera sin signo.

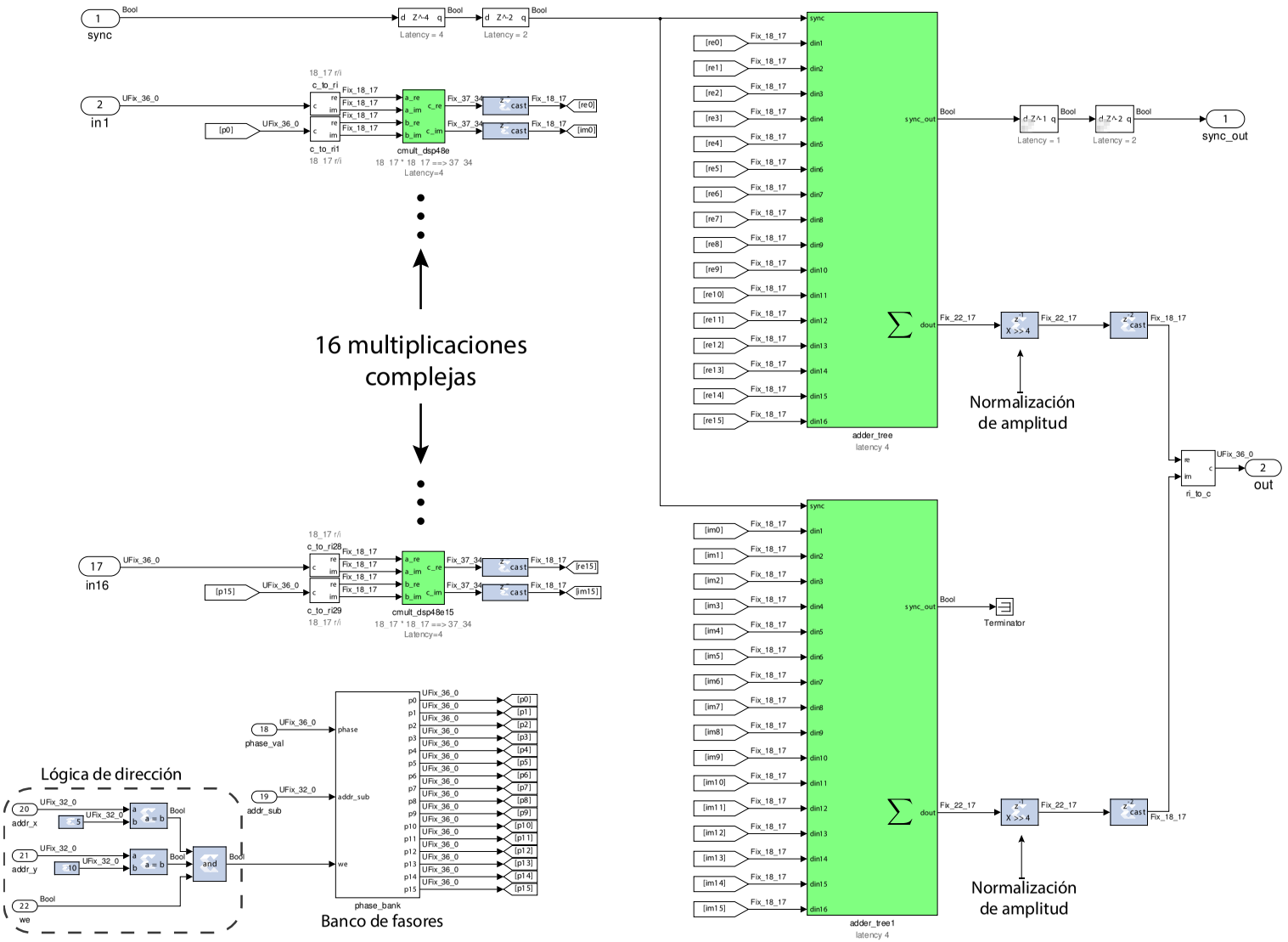


Figura 3.5: Bloque `beamformer` para sintetizar haces.

3.2. Instrumentos: adquisición de datos y controles

Se ha mencionado en la sección 3.1 la existencia de *sondas* y *acciones*, que corresponden a bloques con los que el usuario puede interactuar, según se quiera adquirir datos desde el sistema ó se quiera modificar sus parámetros. Dentro de las sondas existen espectrómetros, integradores de potencia y oscilogramas. Las acciones están relacionadas con registros que pueden ser escritos por el usario.

Para acceder a las sondas y acciones se han implementado clases en Python, agrupadas dentro del paquete `mbf`. Este paquete se ha pensado para que el usuario pueda interactuar con el sistema tanto con rutinas, como con sesiones en iPython.

Dentro del paquete `mbf` también se encuentran dos rutinas importantes:

- `main.py`: Habiendo ejecutado las rutinas en Ruby para inicialización y calibración de los convertidores A/D, la rutina `main.py` inicializa todos los registros configurables (ver sección 3.2.4), realiza la calibración de fase del sistema y grafica en tiempo real la información obtenida con las sondas, usando las clases dentro el paquete `mbf.displays`.
- `measure_psf.py`: Después de haber calibrado la fase con la rutina `main.py`, se barre una zona visible del espacio, definida por el usuario, y se mide la potencia en cada dirección, generando un mapa de intensidad de potencia recibida. Esta rutina se usa para obtener las curvas PSF del capítulo 4.

El paquete se encuentra disponible en el repositorio `github.com/FranciscoCasado/calan-mbf`. A continuación se describen en detalle el diseño digital de las sondas y acciones implementadas.

3.2.1. Espectrómetros

En el sistema existen dos grupos de espectrómetros: los de calibración de fase (sondas `cal_probe`) y los de lectura de los `beamformer` (sondas `bf_probe`).

Las sondas `cal_probe` contienen las multiplicaciones conjugadas entre espectros, mencionadas en la sección 3.1.2, y bloques que calculan la potencia de cada canal de cada espectro, aprovechando el *pipelining* de las FFT, que entrega cada canal en un ciclo de reloj. Las figuras 3.6 y 3.7 muestran la implementación para una señal. En total se tienen dieciseis de cada una, distribuidas en cuatro bloques `cal_probe`.

La última etapa en ambas figuras corresponde a re-cuantizar los datos a una representación de 64.17 bits. Esta operación se realiza para luego acumular los espectros en acumuladores vectoriales `simple_bram_vacc` de 64×256 bits. Cuando el usuario pide una nueva toma de datos, se inicia una nueva acumulación. Al terminar la acumulación, los datos se traspasan a memorias compartidas `bram`, a las cuales se puede acceder con los métodos de lectura de la librería `katcp`. La figura 3.8 muestra la implementación de los acumuladores y memorias compartidas.

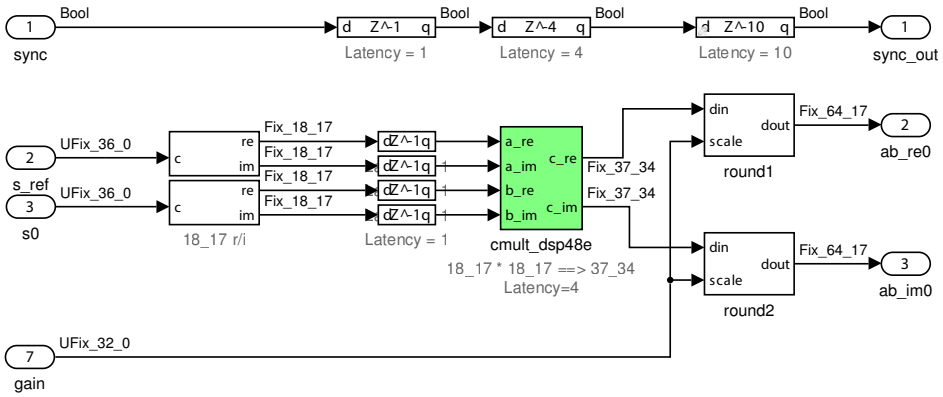


Figura 3.6: Multiplicador conjugado implementado para el cálculo de desfases para una señal.

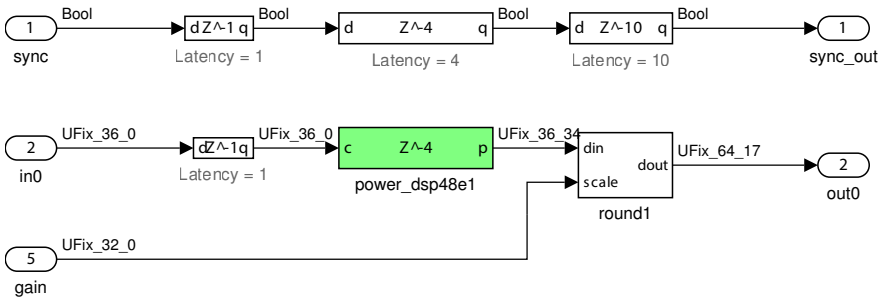


Figura 3.7: Cálculo de potencia implementado para una señal.

Las sondas `bf_probe` contienen sólo el cálculo de potencia y los bloques de re-cuantización, acumulación y traspaso a memoria.

En ambos casos, la sincronización de la toma de datos es realizada por los bloques `[cal,bf]_acc_control`, los que se encargan de iniciar las acumulaciones sólo cuando el usuario escribe un `1` en los registros `[cal,bf]_new_acc` y no antes de que el pulso de sincronización `sync` marque el inicio de una nueva ventana.

Para aprovechar de mejor manera la RAM de la FPGA, cada bloque `bram` tiene un largo de 256 palabras y 128 bits de ancho, por lo que en cada palabra se guarda el $k-$ésimo canal de dos espectros. Posteriormente, el software se encarga de separar estos datos con una rutina de *interleaving*. Para desplegar los espectros en pantalla se ha desarrollado la clase `mbf.displays.Spectra` en Python, que usa las clases `mbf.probes.CalSpectrum` y `mbf.probes.BfSpectrum` para leer las sondas de calibración y síntesis de haces.

3.2.2. Integradores de potencia

A cada señal convertida por el `adc16x250-8` se le calcula la potencia promedio usando integradores de potencia, los cuales acumulan el cuadrado de 1024 muestras en el tiempo. Una vez que se termina una acumulación, se guarda el dato de cada señal en registros, que luego el usuario puede leer con los métodos de la librería `katcp`. La figura 3.9 muestra el diseño del integrador para una señal. Adicionalmente, existe un registro que guarda el valor

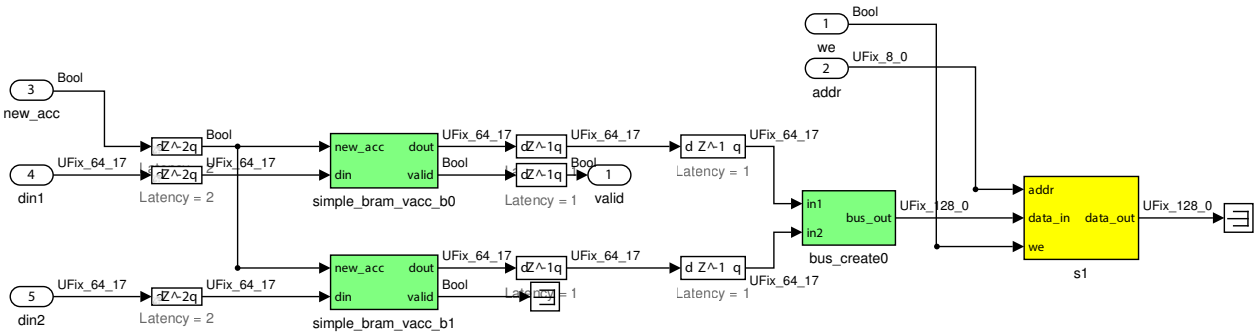


Figura 3.8: Implementación de acumuladores y memorias compartidas para dos señales.

de la salida `pow0` cuando el usuario escribe un pulso en el registro `hold_data`. Estos registros extra, más la señal de control `hold_data` tienen el objetivo de obtener lecturas de la misma ventana de tiempo para todas las señales.

Este bloque de integración se ha añadido en el diseño dada su simpleza y gran utilidad para comprobar de manera rápida - y en tiempo real - el correcto funcionamiento de todo el *hardware* analógico. La calibración de potencia de las tarjetas de *down-conversion* (sección 2.3.3) se basa en los datos proporcionados por estos integradores. Estos datos se despliegan en pantalla usando la clase `mbf.displays.Powers`.

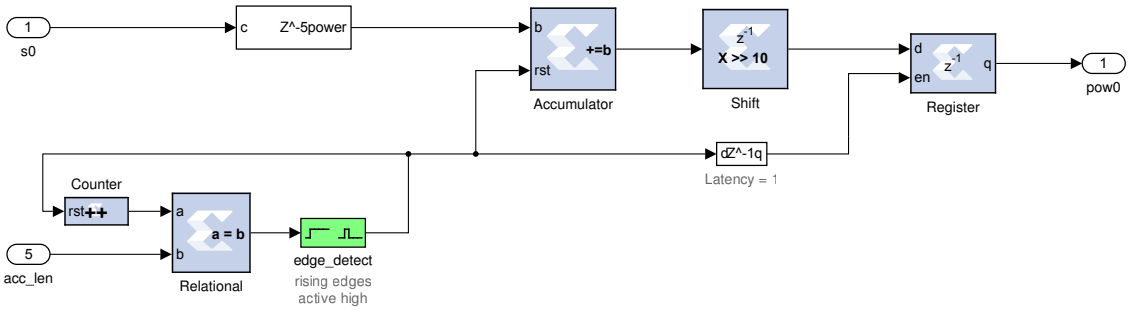


Figura 3.9: Implementación de los integradores de potencia.

3.2.3. Oscilogramas en la entrada

Los oscilogramas a la entrada están implementados con un bloque `snapshot`, con activación manual, que guarda las primeras 1024 muestras de cada señal que entra al sistema, a partir del momento en que se recibe la instrucción de `trigger` por parte del usuario. Para asegurar que todos los oscilogramas se toman al mismo momento, se concatenan todas las señales del `adc16x250-8` con el bloque `bus_create`, en una gran palabra de $8 \times 16 = 128$ bits, que corresponde al ancho máximo de palabra soportado por las memorias. Así, en cada ciclo de reloj se guarda la muestra de cada señal en una línea de memoria. En resumen, los oscilogramas son asíncronos, pero coherentes.

En la lectura se tiene una rutina de *interleaving* que separa la información de cada señal y

que luego puede ser desplegada en pantalla para verificar la forma de las ondas visualmente, usando la clase `mbf.displays.LiveChannels`, como lo muestra la figura 3.11.

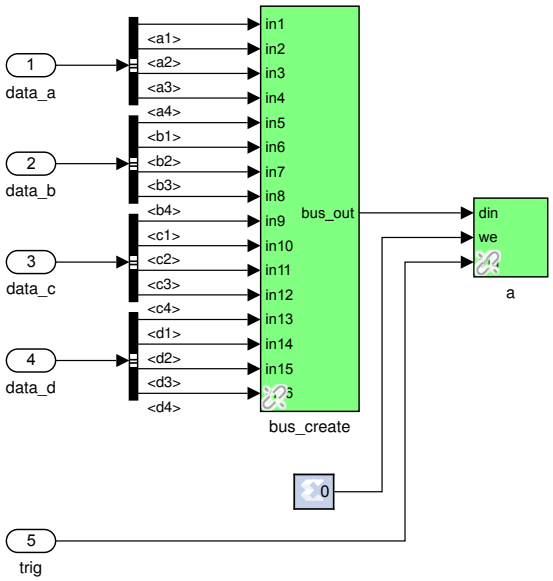


Figura 3.10: Implementación de oscilogramas.

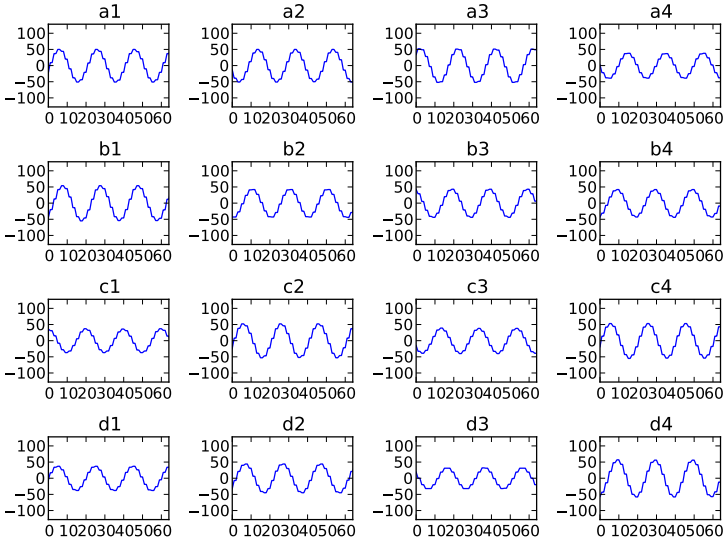


Figura 3.11: Visualización de oscilogramas.

3.2.4. Registros accesibles por el usuario

Registros de configuración de parámetros

Tanto la configuración de fasores y ganancias de cuantización son realizadas por registros que se escriben usando los métodos de escritura `fpga.write_int` de la líbrería `katcp`. La tabla 3.1 lista los registros relacionados con los parámetros que el usuario puede variar.

Registro	Descripción	Configuración recomendada
`cal_gain`	Ganancia de cuantización para las sondas de calibración	2^{17}
`cal_acc_len`	Largo de acumulación en las sondas de calibración	2^{12}
`cal_phase_addr`	Dirección del fasor de calibración que se quiere escribir	-
`cal_phase_re`	Parte real del fasor a escribir	2^{16}
`cal_phase_im`	Parte imaginaria del fasor a escribir	2^{16}
`bf_gain`	Ganancia de cuantización para las sondas de *beamforming*	2^{17}
`bf_acc_len`	Largo de acumulación en las sondas de *beamforming*	2^{12}
`bf_addr_x`	Dirección `x` del bloque `beamformer` al que se quiere cambiar un fasor	-
`bf_addr_y`	Dirección `y` del bloque `beamformer` al que se quiere cambiar un fasor	-
`bf_addr_sub`	Dirección del fasor, dentro del bloque `beamformer`, se quiere escribir	-
`bf_phase_re`	Parte real del fasor a escribir	2^{16}
`bf_phase_im`	Parte imaginaria del fasor a escribir	2^{16}

Tabla 3.1: Registros configurables por el usuario.

Los valores recomendados para los registros de fasores `[cal,bf]_phase_[re,im]` (ver tabla 3.1) corresponden a los valores de inicialización, ya que luego de la calibración estos son completamente diferentes. La magnitud de los valores obedece a que son cercanos al máximo número en representación 18.17 bits, pero son lo suficientemente pequeños para evitar *overflows*, a cambio de un sacrifico leve de ganancia.

Los valores recomendados para largos de acumulación se basan en la experimentación con el modelo. Disminuir este parámetro tiene el efecto de integrar durante menos tiempo, pero con un SNR más bajo.

Los valores recomendados de ganancia de cuantización corresponden al mínimo necesario para no perder bits menos significativos en la recuantización dentro de las sondas. La pérdida de bits afecta directamente la medición del piso de ruido.

Registros para gatillar adquisición de datos y aplicar cambios de parámetros

Tanto la adquisición de datos como la modificación de parámetros se realiza escribiendo un pulso en registros que tienen interpretación booleana dentro del sistema. Para accionarlos se debe usar el método `fpga.write_int` de la líbrería `katcp`, escribiendo un 0 seguido de un `1`, de modo de producir un flanco de subida. Los oscilogramas (`snapshots`) son la única excepción a este sistema de accionamiento, ya que la librería `katcp` implementa su propia activación con el método `fpga.snapshot_get`.

Registro	Descripción
`hold_data`	Retención del valor de potencia integrada en los integradores de potencia
`cal_new_acc`	Inicio de una nueva acumulación en las sondas de calibración
`bf_new_acc`	Inicio de una nueva acumulación en las sondas de *beamforming*
`cal_cnt_rst`	Reinicio de contadores de acumulación y sincronización de acumuladores de sondas de calibración con ventanas de las FFT
`bf_cnt_rst`	Reinicio de contadores de acumulación y sincronización de acumuladores de sondas de *beamforming* con ventanas de las FFT
`cal_phase_we`	Escritura del banco de fasores de calibración
`bf_phase_we`	Escritura del banco de fasores de un bloque `beamformer`

Tabla 3.2: Registros para gatillar adquisición de datos y aplicar cambios de parámetros.

Es de suma importancia escribir un pulso en los registros `cal_cnt_rst` y `bf_cnt_rst`, de lo contrario los espectros medidos en cualquiera de los dos tipos de sondas estarán desfasados, es decir, el primer canal en el gráfico no será necesariamente el primero de la FFT (DC). Se recomienda reiniciar los contadores justo después de haber programado la FPGA y calibrado los bloques SerDes de los convertidores A/D.

Configuración de fasores

La configuración de los fasores es levemente más compleja, ya que se deben escribir los valores de manera secuencial. El valor de cada fasor se aloja dentro de un *banco de fasores*, compuesto por registros y una lógica combinacional que de-multiplexa la señal que sobreescribe los valores. La figura 3.12 muestra el circuito responsable de la configuración de fasores: El banco de fasores de calibración contiene 16 unidades como la que muestra la figura 3.12. Los bancos de fasores de los bloques `beamformer` tienen una lógica levemente más grande, ya que manejan más de una dirección. La secuencia de configuración de un fasor es la siguiente:

1. Escribir los registros de dirección `address`,

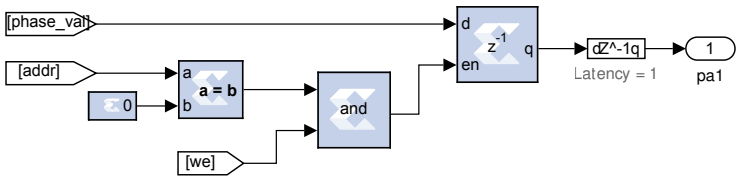


Figura 3.12: Lógica para almacenar un fasor, dentro del banco de fasores.

2. Escribir los registros `phase_re` y `phase_im`,
3. Escribir un pulso en el registro `phase_we`.

De esta manera, se requiere de cinco escrituras de registros para cada fasor de calibración de fase, haciendo un total de 80 escrituras. La configuración de bloques `beamfomer` requiere dos escrituras adicionales por cada fasor (tres direcciones en lugar de una), lo que hace un total de 112 escrituras. Una mejor solución a este problema es abordada en el capítulo 4.

3.3. Cálculo de fasores de síntesis de haces

El tipo de sintetizador de haz implemetado corresponde a un *dealy-and-sum beamformer*, por lo que se calcula la función de transferencia del procesador descrita por la ecuación (1.20). Para esto, se calcula el *array-manifold vector* en la dirección (θ, ϕ) deseada:

$$\mathbf{v}^* \left(\mathbf{k}_T = \mathbf{k}|_{(\theta,\phi)} \right) = \begin{bmatrix} \mathrm{e}^{-j\mathbf{k}_T \cdot \mathbf{p}_0} \\ \mathrm{e}^{-j\mathbf{k}_T \cdot \mathbf{p}_1} \\ \vdots \\ \mathrm{e}^{-j\mathbf{k}_T \cdot \mathbf{p}_{N-1}} \end{bmatrix}. \tag{3.5}$$

Dado que la distribución de los elementos del arreglo es planar uniforme, la parametrización de las posiciones está dada por:

$$\mathbf{p}_{ij} = \frac{\lambda}{2} \cdot \begin{bmatrix} i - \frac{3}{2} \\ 0 \\ j - \frac{3}{2} \end{bmatrix}, \qquad i, j = 0, \ldots, 3. \tag{3.6}$$

El sistema de coordenadas usado para el cálculo corresponde a situar θ como ángulo de elevación y ϕ como ángulo azitmutal. La dirección normal al plano del arreglo corresponde a los ángulos $(\theta = 0°, \phi = 0°)$. La figura 3.13 muestra el sistema de coordenadas empleado.

Tal como se ha presentado en la sección anterior, el bloque `beamformer` requiere de la escritura de varios registros para poder variar sus fasores. Es por esto que se ha implementado la clase `mbf.actions.Beamformer`, que realiza tanto el cálculo, como la secuencia de escritura de registros. Para inicializarla se debe ingresar el objeto `fpga` de la librería `katcp` y las direcciones `x` e `y` del bloque. Luego, para girar el haz de dicho `beamformer` basta ejecutar el método `steer_beam`, ingresando la dirección (θ, ϕ) a la que se quiere apuntar.

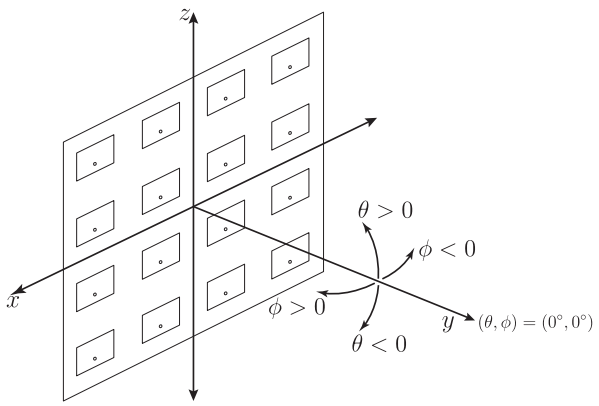


Figura 3.13: Sistema de coordenadas del arreglo.

3.4. Resumen de recursos utilizados

A continuación se muestra un resumen de los recursos utilizados para la síntesis del modelo. Este resumen es un extracto del reporte de compilación; el anexo E contiene el reporte completo:

```
Slice Logic Utilization:
  Number of Slice Registers:                88,565 out of 595,200   14%

  Number of Slice LUTs:                     58,170 out of 297,600   19%
    Number used as logic:                   27,541 out of 297,600    9%
    Number used as Memory:                  12,122 out of 122,240    9%
    Number used exclusively as route-thrus: 18,507


Slice Logic Distribution:
  Number of occupied Slices:                21,403 out of  74,400   28%


Specific Feature Utilization:
  Number of RAMB36E1/FIFO36E1s:               163 out of   1,064   15%
  Number of RAMB18E1/FIFO18E1s:               172 out of   2,128    8%
  Number of DSP48E1s:                         755 out of   2,016   37%


Average Fanout of Non-Clock Nets:                1.85
```

El resumen de recursos utilizados muestra que la cantidad de *DSP-slices* es notoriamente mayor. La cantidad de *slices* logicos es suficientemente baja como para implementar aritmética (sumas y multiplicaciones) de manera comportacional (*behavioural*). En la sección 4.2 se presenta una discusión sobre mejoras, optimizaciones y potencialidades del sistema.

Capítulo 4

Resultados y Discusión

4.1. Caracterización del sistema

En esta sección se presentan los resultados de las dos pruebas realizadas para caracterizar el sistema. Como ya se ha mencionado en la sección 2.1, la primera prueba permite ver el comportamiento del sistema sin considerar el arreglo, de modo de aislar efectos del ambiente y verificar el funcionamiento de la electrónica. Al agregar el arreglo, en la segunda prueba, se notan los efectos externos de inmediato.

4.1.1. Comportamiento del sistema alimentado con señal sintética

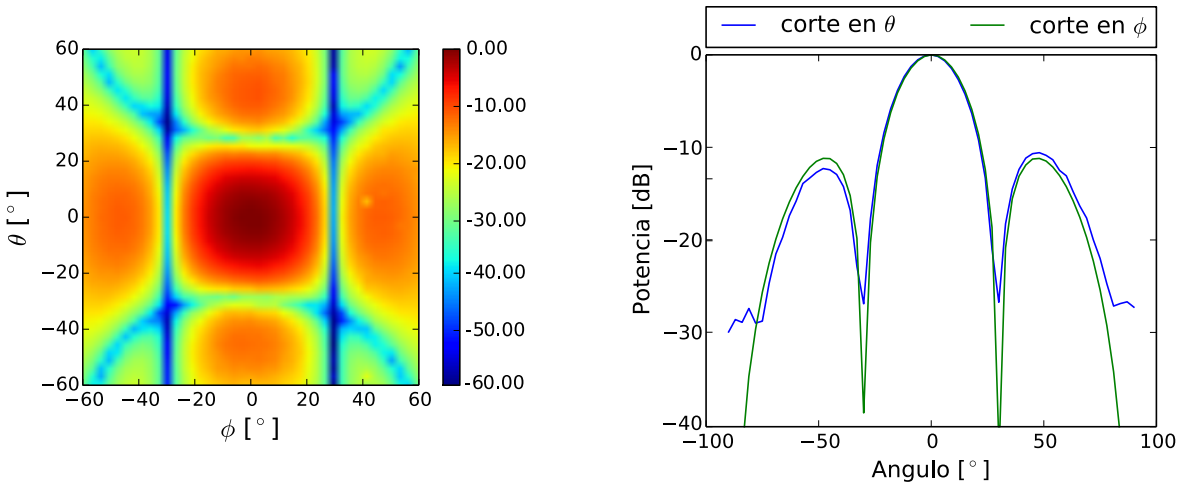
Respuesta espacial del sistema

Para esta prueba se han inyectados tonos coherentes en las entradas RF de las tarjetas de *down-conversion*[1]. Los resultados más importantes que se obtienen de esta prueba corresponden a la curva PSF, que representa la respuesta al impulso espacial del sistema, mostrada en la figura 4.1.

Para obtener esta imagen se ha barrido la región $\theta, \phi \in [-90°, 90°]$ del espacio, apuntando uno de los haces del sintetizador con pasos de 3°, cubriendo un total de 61×61 puntos. Debido a la alta tasa de *overhead* en la comunicación entre el computador y la ROACH2, necesaria para girar los haces y adquirir las mediciones, esta imagen toma cerca de 35 minutos en completarse.

La PSF muestra claramente un HPBW de 20°, simétrico, y lóbulos laterales en ambas direcciones (θ y ϕ) cuyos máximos son un orden de magnitud inferior al máximo del lóbulo principal.

[1]Para inyectar 16 tonos se ha usado un generador y 5 divisores de potencia (4 etapas en total) como los de la sección 2.4.2.



(a) (b)

Figura 4.1: PSF del sistema en alimentado con una señal sintética.

Esta información corrobora no solo que el sistema funciona y barre el espacio de manera adecuada, sino también que la calibración de fase en base a un frente de onda monocromático, normal a la superficie del plano, es suficiente para llevar a cabo la síntesis de haces en direcciones arbitrarias.

Sensibilidad

En la figura 4.2 se presenta uno de los espectrómetros de un `beamformer`, en unidades relativas. En ella se aprecia que el tono está cerca de 57 dB sobre el piso de ruido. Al aumentar la potencia del generador del tono en 10 dB se evidencia la saturación de la electrónica de *down-conversion*, lo que permite concluir que el rango dinámico del sistema es del orden de 63 dB.

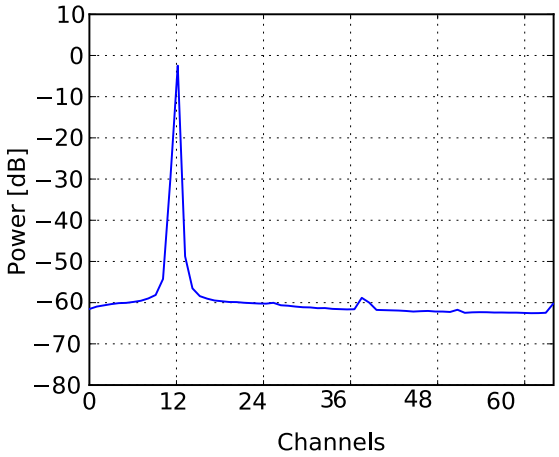


Figura 4.2: Espectro de un `beamformer` alimentado con una señal sintética de 10 MHz. El espectro abarca 64 canales, desde 0 a 50 MHz.

No se debe olvidar que la ganancia analógica del sistema, dada por las tarjetas de *down-conversion* se ha configurado y calibrado para estar en el máximo posible. Por lo tanto, este número también representa la máxima sensibilidad del instrumento, en términos de las pérdidas de espacio libre que es capaz de compensar para una o más fuentes que emiten una potencia dada, a una distancia dada.

Ahora bien, si sequiere medir fuentes de alta potencia a poca distancia, se puede bajar la ganancia de las tarjetas de *down-conversion*. Teniendo en cuenta la tabla de calibración D.1, el canal con la menor ganancia es el `d1`, al cual se le pueden bajar otros 62 dB, combinando los valores posibles de `lna_gain` y `vga_gain`. Esto agrega otros 62 al rango dinámico del sistema.

4.1.2. Comportamiento del sistema con arreglo montado

Respuesta espacial

Replicando la prueba de la sección anterior, se instala el experiemnto de la figura 2.1 en una sala vacía[2], situando el *hardware* de recepción y la fuente puntual en dos extremos opuestos. La distancia entre ambas partes es de 6 metros. Originalmente se alimentó la fuente emisora con ún tono de -20 dBm, pero al estar en un ambiente sin condiciones ideales, fue posible ver comunicaciones en la banda de operación, situación que se muestra en la figura 4.3. Entonces, para que la medición estuviese varios órdenes de magnitud por sobre estas interferencias se aumentó la potencia hasta $+10$ dBm.

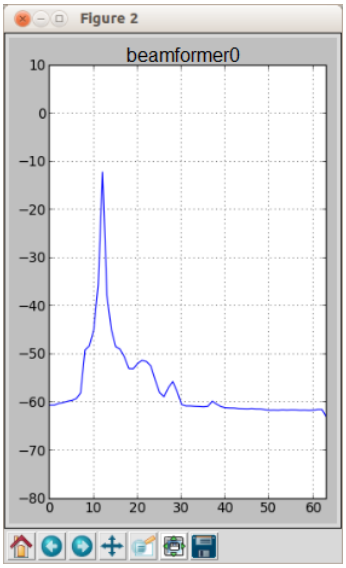


Figura 4.3: Espectro medido en la sala de pruebas.

Esta imagen corresponde a un pantallazo de un espectro de un haz, apuntando en dirección $(\theta, \phi) = (0°, 0°)$, mas en tiempo real se puede ver que: 1) la comunicación es intermitente y

[2]Sala Seminario del Departamento de Astronomía

2) la información ocupa sólo la banda de paso de las tarjetas de *down-conversion* (40 MHz).

Los gráficos de la figura 4.4 muestran las PSF del sistema para distintas posiciones de la fuente y los gráficos de la figura 4.5 muestran los cortes en θ y ϕ de la PSF, centrados en sus respectivos máximos. Todas las imágenes barren conos de visión de $\pm 60°$, tanto para θ como para ϕ, a pasos de $3°$, generando una matriz de 41×41 puntos. Cada imagen toma cerca de 14 minutos en completarse.

Las distintas posiciones de la fuente corresponden a desplazamientos de entre $\approx \pm 2$ metros en la horizontal y $\pm 1{,}3$ metros en altura, equivalentes a $\approx \pm 18°$ en ϕ y $\approx \pm 12°$ en θ. En las figuras 4.4 las filas corresponden a las distintas alturas de la fuente dentro de la sala (cercana al piso, a media altura y cercana al techo), mientras que las columnas corresponden al desplazamiento a lo largo de la sala.

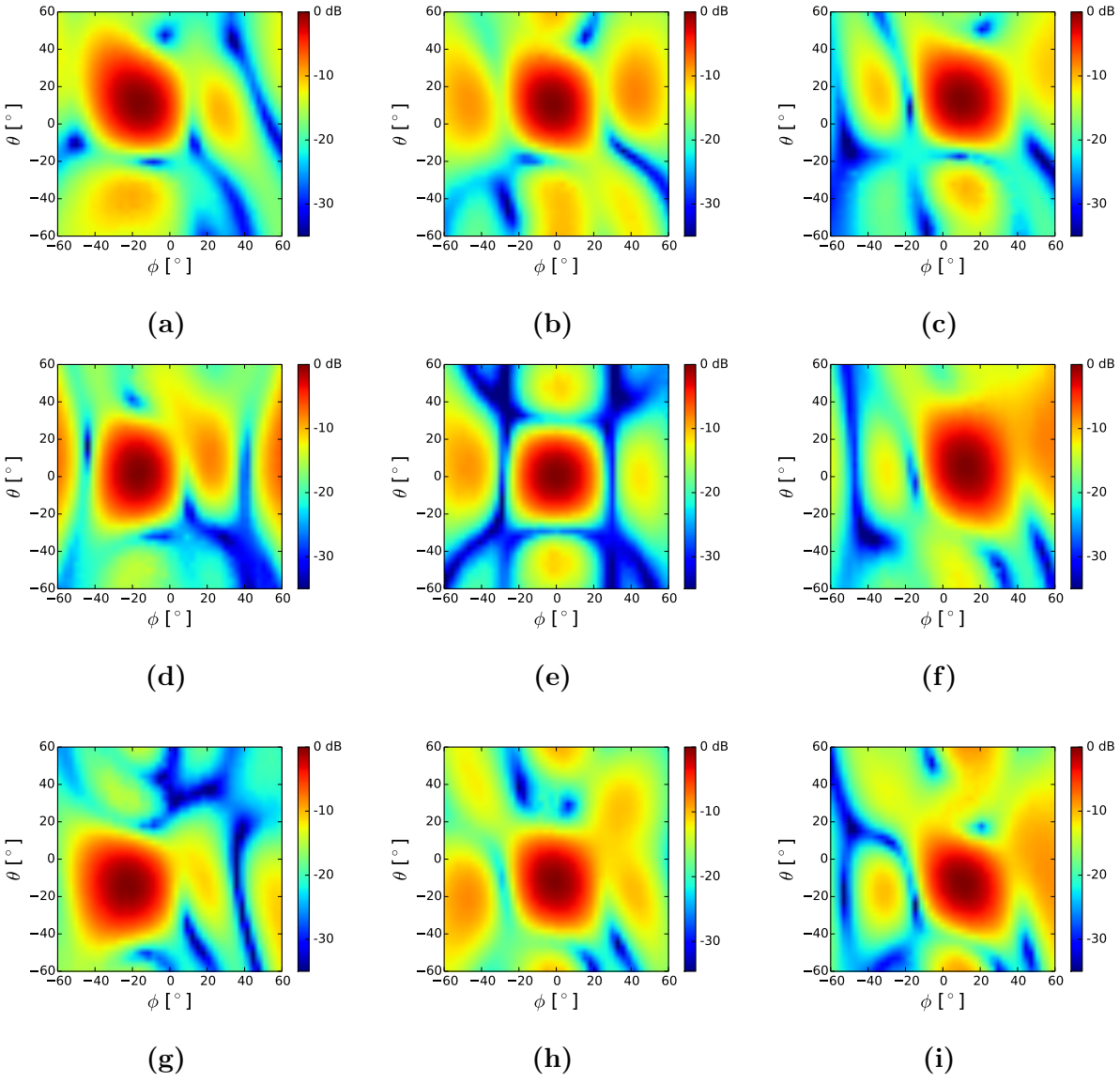


Figura 4.4: PSF para fuente en distintas posiciones.

La calibración de fase del sistema se ha realizado considerando la situación de la figuras 4.4e y 4.5e, que corresponde a tener la fuente justo en frente del arreglo. Si se compara esta situación con la figura 4.1a se puede notar la distorsión del patrón por efecto de condiciones externas. El anexo B muestra los cortes de la 4.5 comparando cada corte en θ con su respectivo corte en ϕ, de modo de graficar la excentricidad de cada patrón.

En todos los demás casos se nota una pérdida de simetría y distorsión evidente de los lóbulos laterales, principalmente cuando se mira en direcciones de ϕ positivo. Esto se puede atribuir, principalmente, a efectos de la sala en la medición. Sin embargo, uno de los resultados más importantes es el hecho de que se puede calibrar con una fuente puntual suficientemente alejada del arreglo y que la calibración es consistente, lo que se comprueba al cambiar la fuente emisora de lugar.

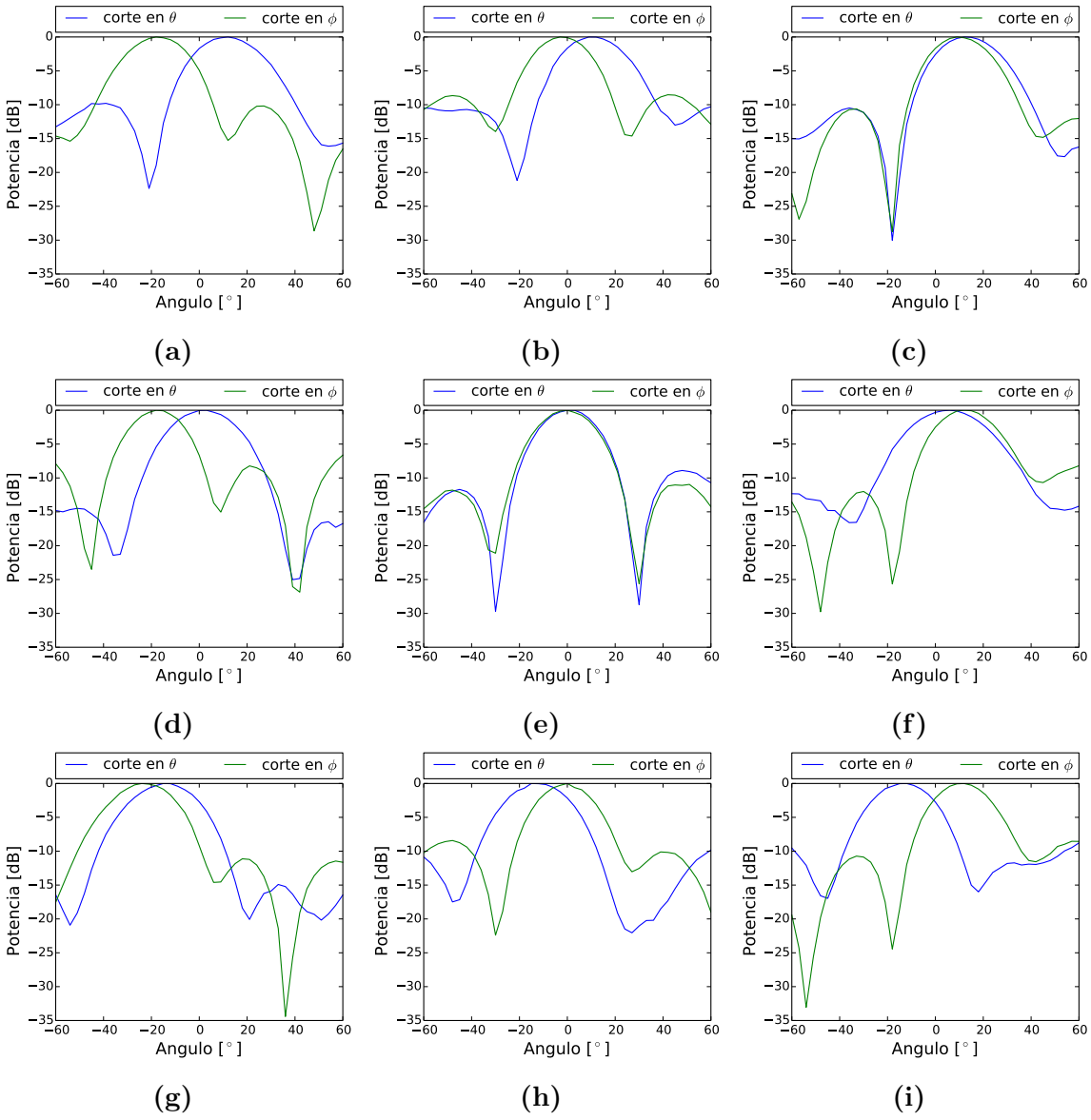


Figura 4.5: Cortes de planos θ y ϕ de las figuras 4.4a-4.4i. Cada par de cortes está centrado en máximo de su respectivo mapa de potencia.

4.2. Mejoras propuestas

4.2.1. Optimización de configuraciones

Ya se ha comentado que el *overhead* de la comunicación es demasiado grande como para leer y escribir datos constantemente. Para ambos casos, condensar más información en un paquete disminuye el *overhead*, así como también se ahorra tiempo si se escriben menos paquetes. Se propone entonces:

- En la escritura de fases se puede ahorrar tiempo si las direcciones son administradas por contadores y los valores de las fases son cargados previamente en memorias, de modo que configurar un `beamformer` tome sólo decenas de ciclos de reloj.
- En la lectura, se puede ser más selectivo en la cantidad de canales que se leen, en lugar de leer cada espectro por completo.
- Otra opción para el proceso de lectura es agregar una lógica que permita crear el mapa de potencia dentro de la FPGA, de modo de hacer sólo una lectura por cada imagen, lo que reduciría drásticamente el tiempo de adquisición de datos.

4.2.2. Añadir más `beamformer`

Actualmente la implementación contiene sólo tres bloques `beamformer`, sin embargo, agregar más de las mismas características requiere hacer *floorplanning* en el diseño, una técnica que permite restringir el espacio que ocupan los distintos bloques dentro de la FPGA, de modo de disminuir el largo del camino eléctrico que debe recorrer una señal. La implementación actual no logra compilar sin fallas de *timing closure* si se usa el compilador por defecto de CASPER, pero con ayuda del *software* SmartXplorer se ha logrado compilar con un *worst-case slack* de 0,004 ns. Este tiempo es más bien ajustado y se podría mejorar haciendo *floorplanning*. También se ha intentado compilar un modelo con 4 bloques `beamformer`, pero el puntaje de *timing closure* es sobre 10000, lo que está totalmente fuera de poder ser optimizado con SmartXplorer.

Otra forma de aumentar el número de sintetizadores, sacrificando desempeño, consiste en no acumular espectros, sino integrar toda la potencia en el dominio de la frecuencia y guardar el dato en un registro. Esta operación que no requiere acumuladores vectoriales, por lo que la cantidad de RAM usada en el proceso es nula.

4.3. Potencialidades del sistema

4.3.1. Potencialidades para *imaging*

Para utilizar este receptor como *multipixel*, se requeriría poder introducir la mayor cantidad de sintetizadores de haces posibles. Como ya se ha comentado, esto es posible si es que se renuncia a la posibilidad de muestrear espectros, lo que no es ningún problema si lo que se quiere es sólo medir potencia total recibida. Ahora bien, es necesario tener en cuenta la cantidad de recursos disponibles dentro de la FPGA.

Cada `beamformer` utiliza 32 `dsp48e1`, por lo que la cantidad máxima de sintetizadores que se podrían agregar es igual a 39. A esto hay que descontar la factibilidad poder hacer *Place & Route* de forma exitosa, lo que es menos probable cuando se usa sobre el 70 % de los recursos, incluso haciendo *floorplanning* [22]. En este sentido, una cantidad razonable sería intentar implementaciones de hasta 5×5, que mantienen el margen sugerido de uso de recursos. En caso de querer exigir más aún los recursos de la FPGA, siempre existe la posibilidad de usar lós *slices* lógicos para hacer implementaciones comportacionales (*behavioural*) de la multiplicación.

Existen en la literatura [23] implementaciones de aperturas RF múltiples que se aprovechan de aproximaciones de la transformada de Fourier para implementarla de manera espacial. Esta técnica permite obtener múltiples haces (3×3 si se usan 4×4 elementos receptores) apuntando en una dirección fija, sin usar multiplicaciones, ya que las aproximaciones sólo usan sumas y operacioines de cambios de signo. Si en cambio, se está dispuesto a barrer electrónicamente el espacio, se pueden implementar menos haces, pero con una lógica que vaya reconfigurando los fasores para apuntar en distintas direcciones de manera automática y sincronizada con las ventanas de integración. Para esto es necesario usar memorias donde se alojen los fasores, como se propone en la sección anterior

4.3.2. Potencialidades para comunicaciones

Un uso atractivo para comunicaciones consiste en la posibilidad de implementar decodificadores dentro de la FPGA, de modo de extraer la información de manera directa. Aprovechando que todas las señales son traspasadas al dominio de la frecuencia, lo más simple es implementar un decodificador FSK. Para esto es relevante tener en consideración dos cosas:

- El número de canales útiles determina el número de bits, en este caso 64, con una resolución espectral de 0,78125 MHz por canal,
- El tiempo de integración determina la velocidad de la comunicación, además del SNR, según lo siguiente:

$$\text{bitrate} = \frac{100 \text{ MSPs} \times \text{Canales útiles}}{\text{Tamaño FFT} \times \text{Largo de acumulación}} \tag{4.1}$$

  Usando el largo de acumulación recomendado en la tabla 3.1, se obtienen $\approx 6{,}1$ kbps, lo que es una comunicación lenta para los estándares actuales. Para mejorar la tasa de

transferencia de datos se puede bajar el largo de acumulación de 4096 a 32 o 16, lo que aumentaría el bitrate a cerca de 1,56 Mbps, a cambio de un aumento considerable del piso de ruido. En el caso extremo, si se usa un largo de acumulación de 1, entonces el bitrate puede llegar a ser de hasta 25 Mbps por cada haz sintetizado.

Si a este decodificador se le suma un algoritmo de rastreo de fuentes (*tracking*) sencillo, se podrá tener un enlace de recepción dinámico, que además tiene la ventaja de ser altamente directivo, lo que posibilita exclusividad del canal en varias circunstancias.

Conclusión

Se ha implementado una plataforma para realizar síntesis de tres haces directivos, de 20° de ancho y lóbulos laterales 10 dB por debajo del principal, capaz de operar en la banda ISM cercana a 5,81 GHz, con un ancho de banda de 40 MHz. El sistema alcanza un rango dinámico cercano a 63 dB.

La limitante principal para su uso como *imager* es el alto *overhead* de la comunicación, lo que se expresa en una tasa de refresco de imágenes de 1 cuadro por cada 14 minutos, para 41×41 puntos (2 puntos por segundo).

Se estima que, usando la técnica desarrollada en este trabajo, el sistema es capaz de sintetizar hasta 36 haces simultaneos, lo que es equivalente a muestrear el espacio a la mitad de la frecuencia espacial de Nyquist.

Los resultados experimentales muestran que es posible medir señales de comunicaciones en la banda ISM, teniendo una señal a ruido de al menos 10 dB incluso a través de muros, lo que hace posible el uso del sistema en ambientes incluso sin línea de vista asegurada.

Dado que las mejoras propuestas para mejorar el desempeño implican usar técnicas avanzadas de desarrollo, éstas se presentan como una oportunidad de generar más conocimiento útil para el laboratorio en el área de la planificación de diseño digitales, lo significaría un apovechamiento mucho mayor del *hardware* disponible.

Esta implementación prueba que la plataforma es totalmente apta para realizar desarrollo e investigación, abriendo un área de estudio dentro del Laboratorio de Ondas milímetricas. También se presenta como una oportunidad para realizar investigación en comunicaciones de sistemas de múltiples entradas y salidas a nivel de capa física.

Bibliografía

[1] C. Balanis, *Antenna Theory: Analysis and Design*, 4th ed. Wiley, 2016.

[2] David M. Pozar, *Microwave Engineering.* John Wiley & Sons, Inc., 2015.

[3] CASPER, "Wideband Spectrometer," accessed: 2017-10-12.

[4] G. Babur, G. O. Manokhin, A. A. Geltser, and A. A. Shibelgut, "Low-cost Digital Beamforming on Receive in Phased Array Radar," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 9251, no. 1, pp. 1–1, 2017.

[5] T. Salim, J. Devlin, and J. Whittington, "FPGA implementation of a phased array DBF using a latch method," in *International Conference on Networking and Communication, 2004. INCC 204.* IEEE, 2004, pp. 64–68. [Online]. Available: http://ieeexplore.ieee.org/document/1366578/

[6] S. Dikmese, A. Kavak, K. Kucuk, S. Sahin, A. Tangel, and H. Dincer, "Digital signal processor against field programmable gate array implementations of space–code correlator beamformer for smart antennas," *IET Microwaves, Antennas & Propagation*, vol. 4, no. 5, p. 593, 2010.

[7] J. C. Porcello, "Designing and Implementing Multibeam Smart Antennas for High Bandwidth VA V Communications using FPGAs fcarrier BWFrac," *Aerospace Conference, 2013 IEEE*, pp. 1–12, 2013.

[8] V. Seneviratne, A. Madanayake, and L. T. Bruton, "A 480MHz ROACH-2 FPGA realization of 2-phase 2-D IIR beam filters for digital RF apertures," *2nd International Moratuwa Engineering Research Conference, MERCon 2016*, pp. 120–125, 2016.

[9] N. Rajapaksha, A. Madanayake, and L. Bruton, "Systolic array architecture for steerable multibeam VHF wave-digital RF apertures," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 1, pp. 669–687, jan 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7073521/

[10] A. Madanayake, C. Wijenayake, D. G. Dansereau, T. K. Gunaratne, L. T. Bruton, and S. B. Williams, "Multidimensional (MD) circuits and systems for emerging applications including cognitive radio, radio astronomy, robot vision and imaging," *IEEE Circuits and Systems Magazine*, vol. 13, no. 1, pp. 10–43, 2013.

[11] W. Liu and S. Weiss, *Wideband Beamforming*, 4th ed. Wiley, 2014.

[12] Q. L. Yang, Y. L. Ban, K. Kang, C. Y. D. Sim, and G. Wu, "SIW Multibeam Array for 5G Mobile Devices," *IEEE Access*, vol. 4, pp. 2788–2796, 2016.

[13] Shinho Kim and Y. Wang, "Two-Dimensional Planar Array for Digital Beamforming and Direction-of-Arrival Estimations," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 7, pp. 3137–3144, sep 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4770203/

[14] M. Crocco and A. Trucco, "Design of Superdirective Planar Arrays With Sparse Aperiodic Layouts for Processing Broadband Signals via 3-D Beamforming," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 4, pp. 800–815, apr 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6732888/

[15] Maxim-Integrated, "Evaluation Kit for the MAX2851," accessed: 2017-06-22.

[16] Maxim Integrated, "5GHz, 5-Channel MIMO Receiver datasheet," accessed: 2017-06-22.

[17] CASPER, "ROACH 2 rev. 2," accessed: 2017-11-12.

[18] Xilinx Inc., "Virtex-6 family overview," accessed: 2017-11-12.

[19] Analog Devices, "Adc16x250-8," accessed: 2017-10-12.

[20] CASPER, "Adc16x250-8," accessed: 2017-06-22.

[21] H. L. Van Trees and John Wiley & Sons., *Detection, estimation, and modulation theory. Part 4, Optimum array processing.* Wiley-Interscience, 2002.

[22] CASPER, "Speed Optimization with PlanAhead," accessed: 2017-10-12.

[23] G. A. Hampson, R. de Wild, and A. B. Smolders, "Efficient Multi-Beaming for the Next Generation of Radio Telescopes," in *Perspectives on Radio Astronomy: Technologies for Large Antenna Arrays*, A. B. Smolders and M. P. van Haarlem, Eds., 2000, p. 265.

Anexo A

Instrucciones de uso

Para usar se debe seguir lo siguiente:

1. Verificar las conexiones de:
   - cada antena del arreglo y las tarjetas de down-conversión (apretar con llave de torque),
   - las tarjetas de down-conversión y las entradas del `adc16x250-8`,
   - la alimentación de todos los equipos,
2. Encender la ROACH2 y verificar que el sintetizador está encendido y configurado a 200 MHz y +6 dBm (nivel 4). Se recomienda hacer *flash* de la configuración para que no haya que reconfigurar el sintetizador,
3. Conectar una señal sinusoidal de 40 MHz, −3 dBm a la placa de distribución de down-conversión,
4. Encender la alimentación de las tarjetas de down conversión y conectar la alimentación USB de la Raspberry Pi,
5. Abrir una sesión por `ssh` en la Raspberry Pi y correr la rutina `init_mixers.py`,
6. Verificar que el archivo de programación `beamformer.bof` esté cargado en la ROACH2. Para esto se puede abrir una sesión en iPython, crear un objeto `fpga = corr.karcp_wrapper.FpgaClient('roach_ip')` y correr el método `fpga.listbof()`
7. Abrir una terminal y ejecutar la inicialización del `adc16x250-8`: `./adc16_init <beamformer.bof>`
8. (opcional) Correr la rutina de chequeo: `./adc16_check`
9. Ejecutar con Python la rutina `main.py`

Anexo B

Excentricidad de Patrones

Ver siguiente página

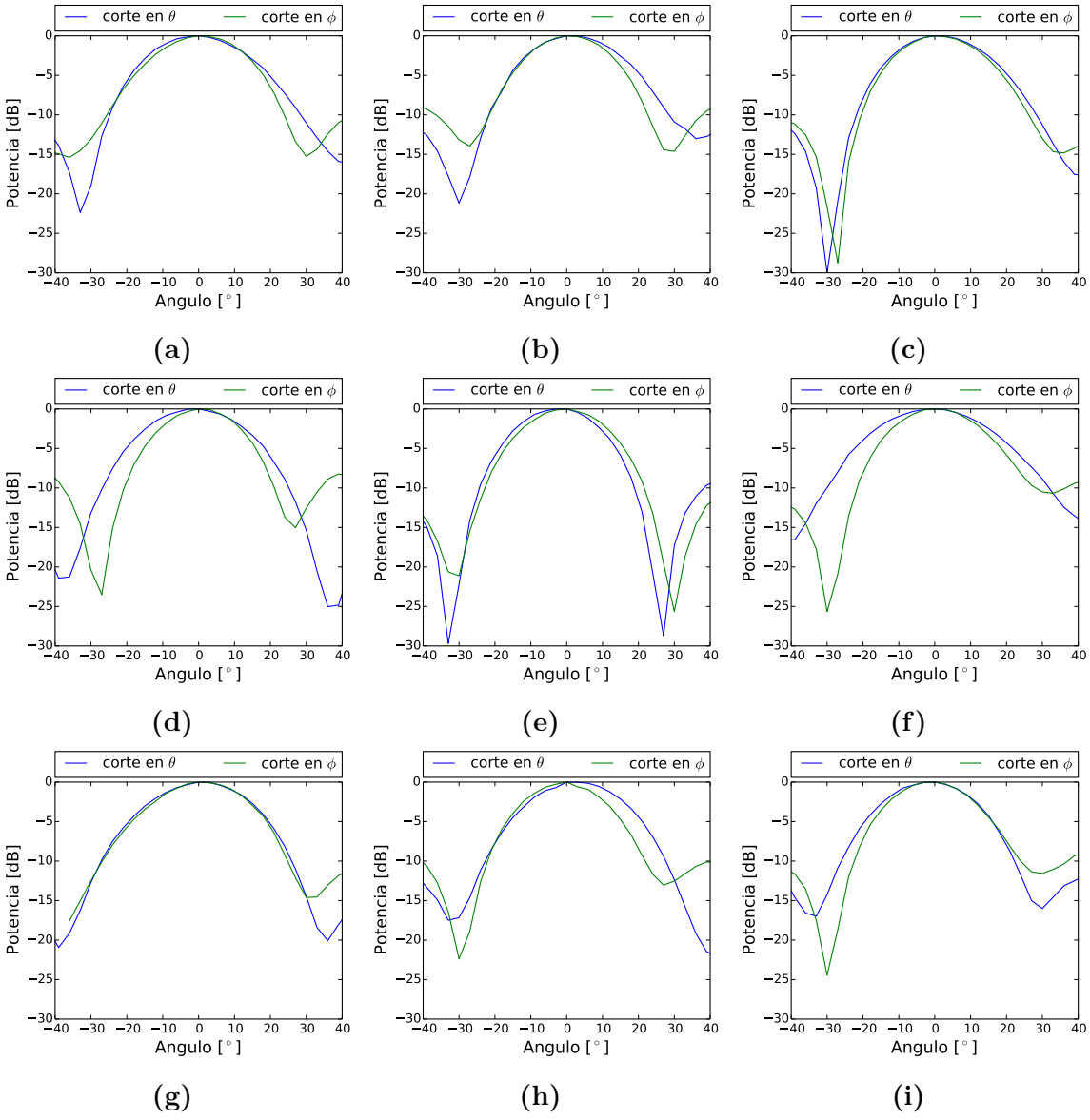


Figura B.1: Cortes de planos θ y ϕ de las figuras 4.5a-4.5i. Cada par de cortes se muestra centrado respecto de sí mismo para comparar la excentricidad de cada uno de los patrones.

Anexo C

Rutinas en Python

C.1. Rutina `main.py`

```
import logging
import sys
import time
import corr
import matplotlib.pyplot as plt
import mbf


bitstream = 'read_power_16chan_2017_Oct_30_1556.bof'
katcp_port = 7147


def exit_fail():
    print 'FAILURE DETECTED. Log entries:\n', lh.printMessages()
    try:
        fpga.stop()
    except:
        pass
    raise
    exit()


def exit_clean():
    try:
        fpga.stop()
    except:
        pass
    exit()


# Main parser
```

```
if __name__ == '__main__':

    from optparse import OptionParser
    p = OptionParser()
    p.set_usage('read_power_16chan.py [options]')
    p.set_description(__doc__)
    p.add_option("-i", '--roach_ip', dest='roach', type='str',
       default='192.168.1.13')
    p.add_option('-s', '--skip', dest='skip', action='store_false',
       default=True,
                help='Skip reprogramming the FPGA and configuring EQ
                   .')
    p.add_option('-b', '--bof', dest='boffile', type='str', default=
       bitstream,
                help='Specify the bof file to load')
    p.add_option('-p', '--power_bars', dest='power_bars', action='
       store_true', default=False,
                help='Skip reprogramming the FPGA and configuring EQ
                   .')
    p.add_option('-c', '--channels', dest='channels', action='
       store_true', default=False,
                help='Skip reprogramming the FPGA and configuring EQ
                   .')
    opts, args = p.parse_args(sys.argv[1:])


try:
    loggers = []
    lh = corr.log_handlers.DebugLogHandler()
    logger = logging.getLogger(opts.roach)
    logger.addHandler(lh)
    logger.setLevel(10)

    print('Connecting to server %s on port %i... ' % (opts.roach,
       katcp_port)),
    fpga = corr.katcp_wrapper.FpgaClient(opts.roach, katcp_port,
       timeout=10, logger=logger)
    time.sleep(1)

    if fpga.is_connected():
        print 'ok\n'
    else:
        print 'ERROR connecting to server %s on port %i.\n' % (opts.
           roach, katcp_port)
        exit_fail()

    print '------------------------'
    print 'Programming FPGA with %s...' % bitstream,
    if not opts.skip:
        fpga.progdev(bitstream)
```

```
    print 'done'
else:
    print 'Skipped.'

print 'Configuring Quantizers...',
fpga.write_int('cal_gain', 2**17)
fpga.write_int('cal_acc_len', 2**12)
fpga.write_int('bf_gain', 2**17)
fpga.write_int('bf_acc_len', 2**12)
print 'done'

print 'Resetting counters...',
fpga.write_int('cal_cnt_rst', 1)
fpga.write_int('cal_cnt_rst', 0)
fpga.write_int('bf_cnt_rst', 1)
fpga.write_int('bf_cnt_rst', 0)
print 'done'

print 'Setting Initial Phase Calibration...'
pcal = mbf.actions.PhaseCalibration(fpga)
pcal.init_phase()
pcal.calibrate()
print 'done'

print 'Setting Beamformers...',
bf0 = mbf.actions.Beamformer(fpga, 5, 10)
bf0.steer_beam(0, 0)
bf1 = mbf.actions.Beamformer(fpga, 6, 11)
bf1.steer_beam(0, 0)
bf2 = mbf.actions.Beamformer(fpga, 7, 12)
bf2.steer_beam(0, 0)
print 'done'

if opts.power_bars & (not opts.channels):
    powers = mbf.displays.Powers(fpga, plt.figure())
elif (not opts.power_bars) & opts.channels:

    channels = (fpga, plt.figure())

else:
    # powers = mbf.displays.Powers(mbf.probes.PowerIntegrator(
       fpga), plt.figure())
    # channels = mbf.displays.LiveChannels(fpga, plt.figure())
    # spectra_real = mbf.displays.Spectra(fpga, plt.figure(),
       mode='real', numc=4)
    # spectra_imag = mbf.displays.Spectra(fpga, plt.figure(),
       mode='imag', numc=4)
    # spectra_pow = mbf.displays.Spectra(mbf.probes.
       CalSpectrometer(fpga, numc=4), plt.figure(), mode='pow',
       scale='dB')
```

```
        bf_spectra = mbf.displays.Spectra(mbf.probes.BfSpectrometer(
            fpga, numc=2), plt.figure(), mode='pow', scale='dB')

    plt.show()


except KeyboardInterrupt:
    exit_clean()

exit_clean()
```

C.2. Rutina measure_psf.py

```
import corr
import logging
import time
import matplotlib.pyplot as plt
import mbf
import numpy as np
import telegram
from time import strftime, localtime

loggers = []
lh = corr.log_handlers.DebugLogHandler()
logger = logging.getLogger('192.168.1.13')
logger.addHandler(lh)
logger.setLevel(10)

# bot = telegram.Bot('YOUR BOT TOKEN')
# chat_id = <your chat_id>

ip_addr = '192.168.1.13'
fpga = corr.katcp_wrapper.FpgaClient(ip_addr, 7147, timeout=10,
   logger=logger)
time.sleep(1)

'''
if fpga.is_connected():
    print 'ok\n'
else:
    print 'ERROR connecting to server %s on port %i.\n' % (ip_addr,
       7147)
'''

print 'Starting measure'
datetime = strftime("%Y-%m-%d %H:%M:%S", localtime())
# bot.send_message(chat_id=chat_id, text='Running new measurement
   ...'+datetime)
```

```
bf = mbf.actions.Beamformer(fpga, 6, 11)
bf.steer_beam(0, 0)

probe = mbf.probes.BfSpectrometer(fpga, 2)

# Define range
theta = range(-90, 91, 3)
phi = range(-90, 91, 3)

value = np.zeros([len(theta), len(phi)])

bf.steer_beam(0, 0)
time.sleep(1)


for i in range(len(theta)):
    print '['+str(theta[i])+']'
    for j in range(len(phi)):
        # Steer beam
        bf.steer_beam(theta[i], phi[j])
        time.sleep(0.001)

        # Read spectrometer & process data
        re, im, pow, acc_n = probe.read()
        time.sleep(0.001)
        data = (pow[1]/(2.0**17))
        data_dB = 10*np.log10(data)
        value[j][len(theta)-i-1] = data[12]   #


# Save results and notify
np.savez('pattern_'+datetime+'.npz', theta, phi, value)
# bot.send_message(chat_id=chat_id, text='Done!... sending files')

try:
    # Plot results
    X, Y = np.meshgrid(phi, theta)
    levels = np.linspace(np.amin(value), np.amax(value), 1000)
    cmap = plt.cm.get_cmap("jet")

    plt.figure(figsize=(5, 4))
    cp = plt.imshow(value, extent=[-90, 90, -90, 90])    # don't
       forget to set extent to the previously defined range
    plt.xlabel(r'$\phi$ [$^\circ$]', fontsize=16)
    plt.ylabel(r'$\theta$ [$^\circ$]', fontsize=16)
    plt.tight_layout()

    cbar = plt.colorbar(cp, ticks=[levels[0], levels[-1]])
```

```
        cbar.ax.set_yticklabels(["{0:.2f}".format(levels[0]), "{0:.2f}".
            format(levels[-1])])

        # Save figures
        plt.savefig('pattern_'+datetime+'.png')
        plt.savefig('pattern_'+datetime+'.pdf')

        # Send files via Telegram
        # img = open('pattern_'+datetime+'.png', 'rb')
        # doc = open('pattern_'+datetime+'.pdf', 'rb')
        # bot.send_photo(chat_id=chat_id, photo=img)
        # bot.send_document(chat_id=chat_id, document=doc)

        # Display
        plt.show()

    except FutureWarning:
        pass
```

Anexo D

Software de tarjetas de *down-conversion*

D.1. Tabla de calibración de `vga_gain`

	1	2	3	4
A	12 (-6)	14 (-2)	14 (-2)	13 (-4)
B	12 (-6)	13 (-4)	13 (-4)	12 (-6)
C	15 (-0)	14 (-2)	14 (-2)	14 (-2)
D	11 (-8)	14 (-2)	13 (-4)	13 (-4)

Tabla D.1: Valores de `vga_gain` (dB) para calibración.

D.2. Clase `Mixer`

```
import spidev
import RPi.GPIO as GPIO


_WRITE = 0
_READ = 1

_default = [[0, 0, 0x022], [0, 1, 0x0ff], [0, 2, 0x1a0], [0, 3, 0x000
   ], [0, 4, 0x31c],
            [0, 5, 0x000], [0, 6, 0x3ff], [0, 7, 0x024], [0, 8, 0x000
               ], [0, 9, 0x00f],
            [0, 10, 0x000], [0, 11, 0x060], [0, 13, 0x000], [0, 14, 0
               x360], [0, 15, 0x242],
            [0, 16, 0x380], [0, 17, 0x000], [0, 18, 0x080], [0, 19, 0
               x05f], [0, 20, 0x1ea],
            [0, 21, 0x0bf], [0, 22, 0x1b8], [0, 23, 0x065], [0, 24, 0
               x24f], [0, 25, 0x3a8],
```

```
            [0, 26, 0x015], [0, 27, 0x180], [0, 28, 0x063], [0, 29, 0
               x000], [0, 30, 0x000],
            [0, 31, 0x000], [1, 1, 0x000], [1, 2, 0x000], [1, 3, 0
               x000], [1, 4, 0x380],
            [1, 5, 0x000], [1, 6, 0x000], [1, 7, 0x000], [1, 8, 0x1aa
               ], [1, 9, 0x114],
            [1, 10, 0x354], [1, 11, 0x073], [1, 12, 0x000], [1, 13, 0
               x000], [1, 14, 0x000],
            [1, 15, 0x000], [1, 16, 0x000], [1, 17, 0x000], [1, 18, 0
               x000], [1, 19, 0x000],
            [1, 20, 0x000], [1, 21, 0x000], [1, 22, 0x000], [1, 23, 0
               x000], [1, 24, 0x0c4],
            [1, 25, 0x12b], [1, 26, 0x165], [1, 27, 0x000], [1, 28, 0
               x004], [1, 31, 0x000]]

_boards = {'a': 0, 'b': 1, 'c': 2, 'd': 3, }
_ports = {'1': 1, '2': 2, '3': 4, '4': 5, }

_set_local_addrs = [0, 0, 0x023]
_standbyMode = [0, 0, 0x026]
_rxMode = [0, 0, 0x02A]
_doutEn = [0, 14, 0x362]

_A1 = 15
_A2 = 13


def _bitmask(n):
    return int('1'*n, 2)


def _bitread(n, i):
    return int("{0:02b}".format(n)[-(i+1)])


def _message(rw, a, d):
    return (rw << 15) | ((a & _bitmask(5)) << 10) | (d & _bitmask(10)
       )


class Mixer:
    """docstring for Mixer"""
    def __init__(self):
        global _WRITE, _READ, _default, _set_local_addrs, _A1, _A2
        global _standbyMode, _rxMode, _doutEn, _ports, _boards

        # GPIO config
        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(_A1, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(_A2, GPIO.OUT, initial=GPIO.LOW)
```

```
        # SPI config
        bus = 0
        device = 0
        self.spi = spidev.SpiDev()
        self.spi.open(bus, device)
        self.spi.max_speed_hz = 1000000

        # LO parameters
        self._N = 0b01001000
        self._F = 0b10000000000000000000
        self._Flow = self._F & _bitmask(10)
        self._Fhigh = (self._F >> 10) & _bitmask(10)

    # Board actions
    def init_all(self):
        self.init_board(0)
        self.init_board(1)
        self.init_board(2)
        self.init_board(3)

    def init_board(self, board):
        self._select_board(board)
        for i in xrange(len(_default)):
            if not _default[i][0]:
                self.write_reg(_default[i])
                if _default[i+1][0]:
                    self.write_reg(_set_local_addrs)
                else:
                    self.write_reg(_default[i])
        self.write_reg(_standbyMode)
        self.write_reg(_doutEn)
        self.write_reg(15, (self.get_default(15) & ~_bitmask(7)) |
           self._N)
        self.write_reg(16, self._Fhigh)
        self.write_reg(17, self._Flow)
        self.write_reg(5, 1)
        self.write_reg(_rxMode)
        self.write_reg(5, 0)
        print('Done configuring '+str(board))

    def calibrate_all(self):
        self.set_gain('a1',7,12)
        self.set_gain('a2',7,14)
        self.set_gain('a3',7,14)
        self.set_gain('a4',7,13)
        self.set_gain('b1',7,12)
        self.set_gain('b2',7,13)
        self.set_gain('b3',7,13)
        self.set_gain('b4',7,12)
```

```
        self.set_gain('c1',7,15)
        self.set_gain('c2',7,14)
        self.set_gain('c3',7,14)
        self.set_gain('c4',7,14)
        self.set_gain('d1',7,11)
        self.set_gain('d2',7,14)
        self.set_gain('d3',7,13)
        self.set_gain('d4',7,13)

    def _select_board(self,board):
        assert(board < 4)
        GPIO.output(_A1, _bitread(board, 0))
        GPIO.output(_A2, _bitread(board, 1))

    def standby(self, board):
        if board == 'all':
            for i in range(4):
                self._select_board(i)
                self.write_reg(_standbyMode)
        else:
            self._select_board(board)
            self.write_reg(_standbyMode)

    def set_gain(self, channel, lna_gain=7, vga_gain=15):
        if len(channel) > 2:
            print("invalid channel")
            return
        try:
            board = _boards[channel[0]]
            port = _ports[channel[1]]
        except KeyError:
            print("invalid channel")
            return
        self._select_board(board)
        self.write_reg(6, (self.get_default(6) & _bitmask(5)) | 1 <<
           (port+4))
        self.write_reg(1, (self.get_default(1) & ~_bitmask(8)) |
           lna_gain << 5 | vga_gain)

    # SPI communication
    def write_reg(self, addrs, data=None):
        if data is None:
            self.write_reg(addrs[1], addrs[2])
            return
        m = _message(_WRITE, addrs, data)
        send = [m >> 8, m & _bitmask(8)]
        self.spi.xfer2(send)

    def read_reg(self, addrs):
        m = _message(_READ, addrs, 0)
```

```
        send = [m >> 8, m & _bitmask(8)]
        read = self.spi.xfer2(send)
        return (read[0] << 8) | (read[1] & _bitmask(8))

    def get_default(self, addrs):
        for i in _default:
            if i[1] == addrs:
                return i[2]
        return -1

    def clean_gpio(self):
        GPIO.cleanup()
```

D.3. Runtina de inicialización

```
from mixer import Mixer

m = Mixer()
m.init_all()
m.calibrate_all()
m.clean_gpio()
```

Anexo E

Reporte de utilización de recursos

```
Slice Logic Utilization:
  Number of Slice Registers:                88,565 out of 595,200   14%
    Number used as Flip Flops:              88,547
    Number used as Latches:                      2
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:               16
  Number of Slice LUTs:                     58,170 out of 297,600   19%
    Number used as logic:                   27,541 out of 297,600    9%
      Number using O6 output only:          19,214
      Number using O5 output only:           1,657
      Number using O5 and O6:                6,670
      Number used as ROM:                        0
    Number used as Memory:                  12,122 out of 122,240    9%
      Number used as Dual Port RAM:            160
        Number using O6 output only:            32
        Number using O5 output only:             0
        Number using O5 and O6:                128
      Number used as Single Port RAM:            0
      Number used as Shift Register:        11,962
        Number using O6 output only:         9,080
        Number using O5 output only:           288
        Number using O5 and O6:              2,594
    Number used exclusively as route-thrus: 18,507
      Number with same-slice register load: 11,939
      Number with same-slice carry load:     6,568
      Number with other load:                    0
Slice Logic Distribution:
  Number of occupied Slices:                21,403 out of  74,400   28%
  Number of LUT Flip Flop pairs used:       78,964
    Number with an unused Flip Flop:         9,662 out of  78,964   12%
    Number with an unused LUT:              20,794 out of  78,964   26%
```

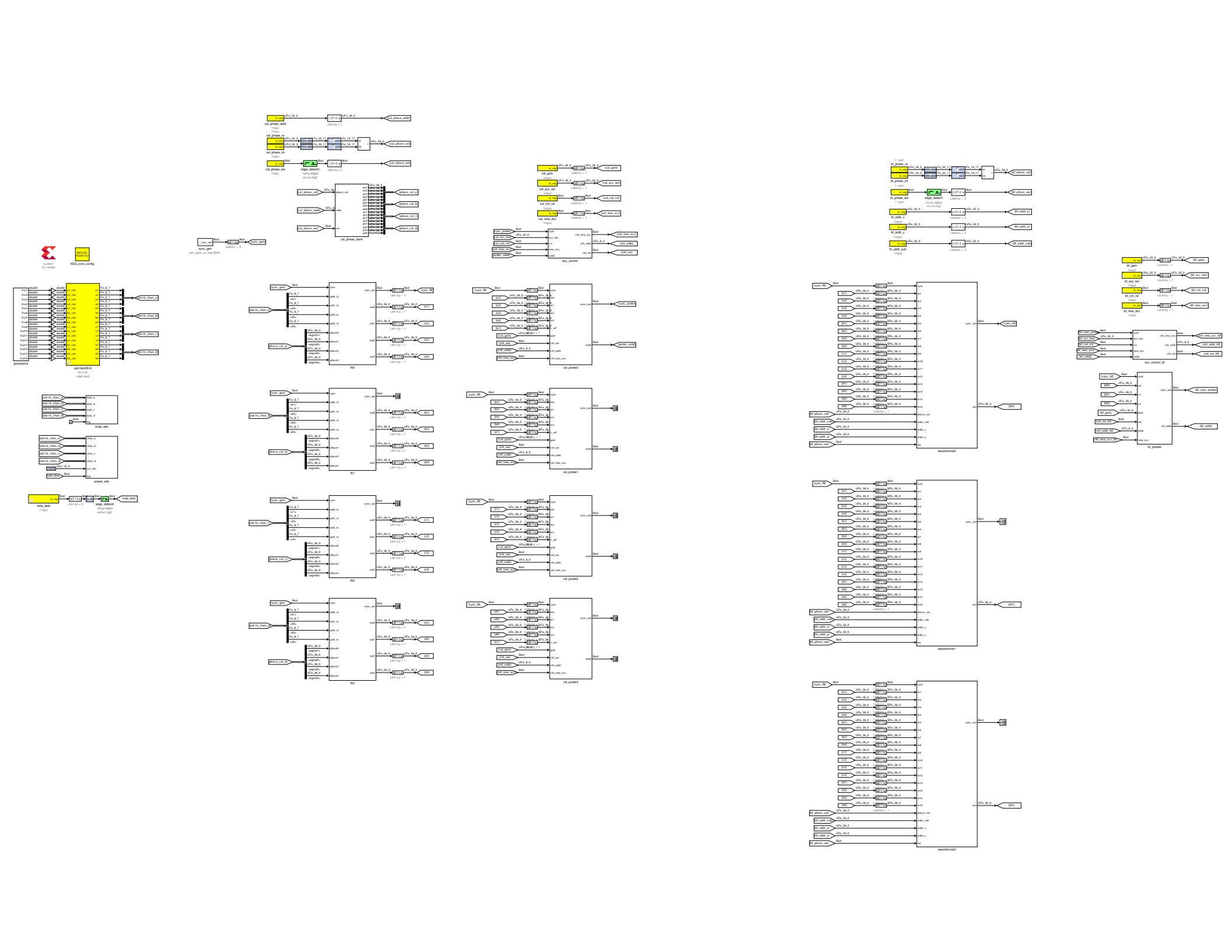
```
    Number of fully used LUT-FF pairs:      48,508 out of  78,964   61%
    Number of unique control sets:             475
    Number of slice register sites lost
      to control set restrictions:           1,599 out of 595,200    1%
...
IO Utilization:
  Number of bonded IOBs:                       142 out of     840   16%
    Number of LOCed IOBs:                      142 out of     142  100%
    IOB Flip Flops:                             95
Specific Feature Utilization:
  Number of RAMB36E1/FIFO36E1s:                163 out of   1,064   15%
    Number using RAMB36E1 only:                163
    Number using FIFO36E1 only:                  0
  Number of RAMB18E1/FIFO18E1s:                172 out of   2,128    8%
    Number using RAMB18E1 only:                172
    Number using FIFO18E1 only:                  0
  Number of BUFG/BUFGCTRLs:                      7 out of      32   21%
    Number used as BUFGs:                        7
    Number used as BUFGCTRLs:                    0
  Number of ILOGICE1/ISERDESE1s:               127 out of   1,080   11%
    Number used as ILOGICE1s:                   63
    Number used as ISERDESE1s:                  64
  Number of OLOGICE1/OSERDESE1s:                36 out of   1,080    3%
    Number used as OLOGICE1s:                   36
    Number used as OSERDESE1s:                   0
  Number of BSCANs:                              0 out of       4    0%
  Number of BUFHCEs:                             0 out of     216    0%
  Number of BUFIODQSs:                           0 out of     108    0%
  Number of BUFRs:                               0 out of      54    0%
  Number of CAPTUREs:                            0 out of       1    0%
  Number of DSP48E1s:                          755 out of   2,016   37%
  Number of EFUSE_USRs:                          0 out of       1    0%
  Number of FRAME_ECCs:                          0 out of       1    0%
  Number of GTXE1s:                              0 out of      36    0%
  Number of IBUFDS_GTXE1s:                       0 out of      18    0%
  Number of ICAPs:                               0 out of       2    0%
  Number of IDELAYCTRLs:                         2 out of      27    7%
  Number of IODELAYE1s:                         32 out of   1,080    2%
  Number of MMCM_ADVs:                           2 out of      18   11%
  Number of PCIE_2_0s:                           0 out of       2    0%
  Number of STARTUPs:                            1 out of       1  100%
  Number of SYSMONs:                             0 out of       1    0%
  Number of TEMAC_SINGLEs:                       0 out of       4    0%

Average Fanout of Non-Clock Nets:                1.85
```

Anexo F

Modelo completo en Simulink

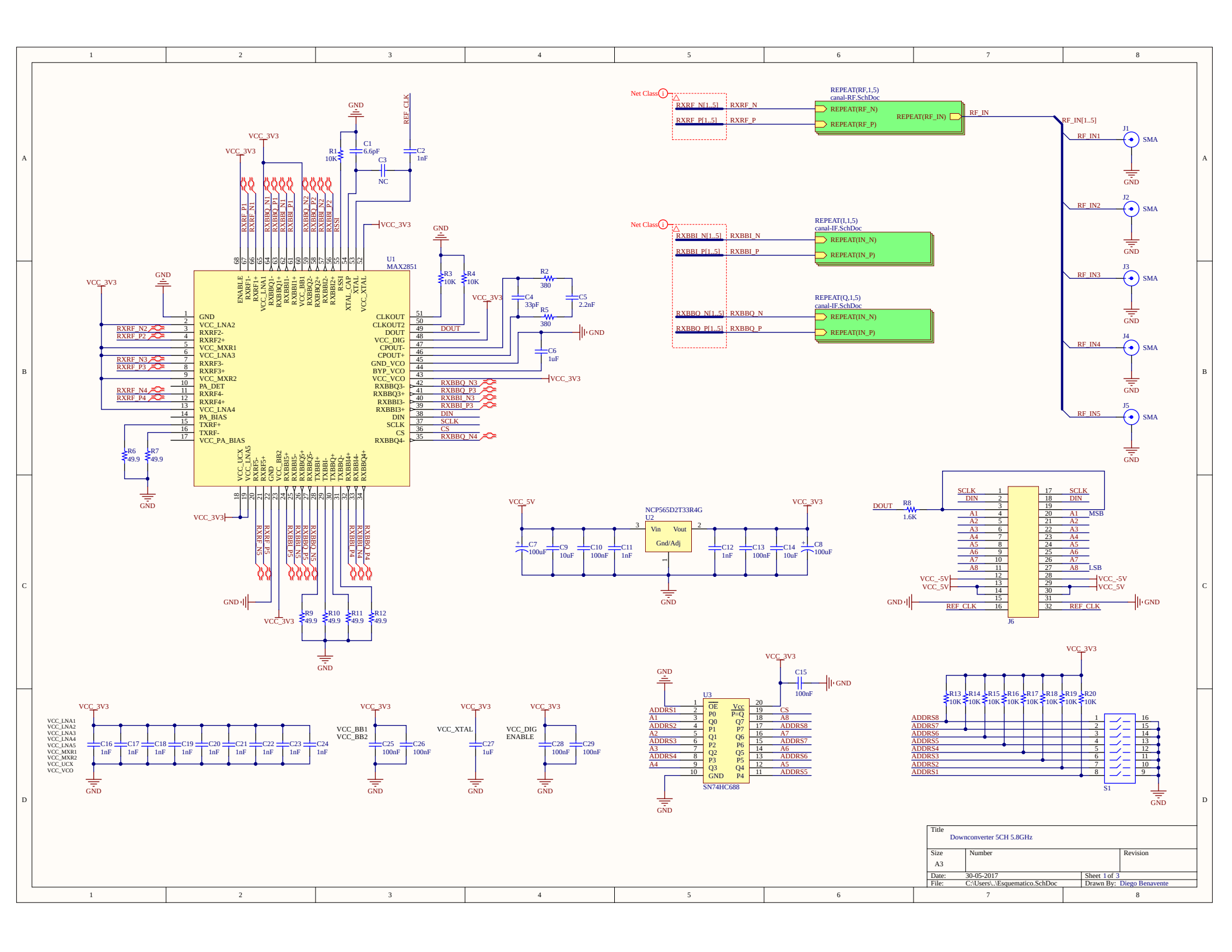
Ver siguiente página



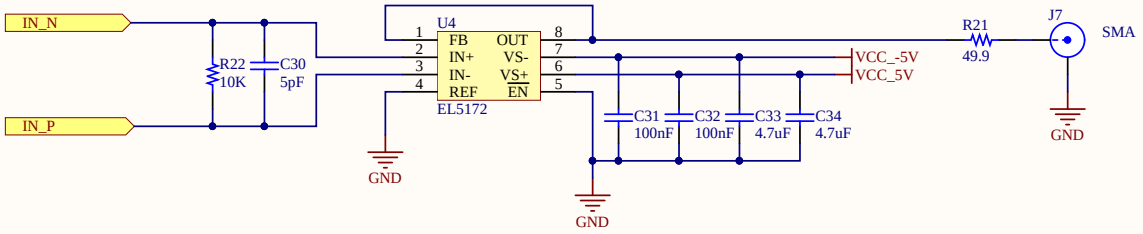
Anexo G

Esquemáticos y Gerbers de la etapa de *down-conversion*

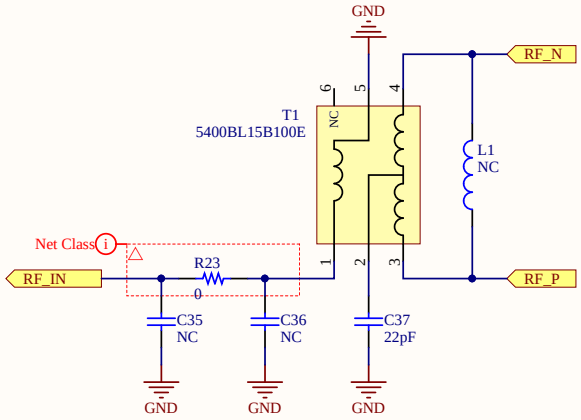
Ver siguiente página



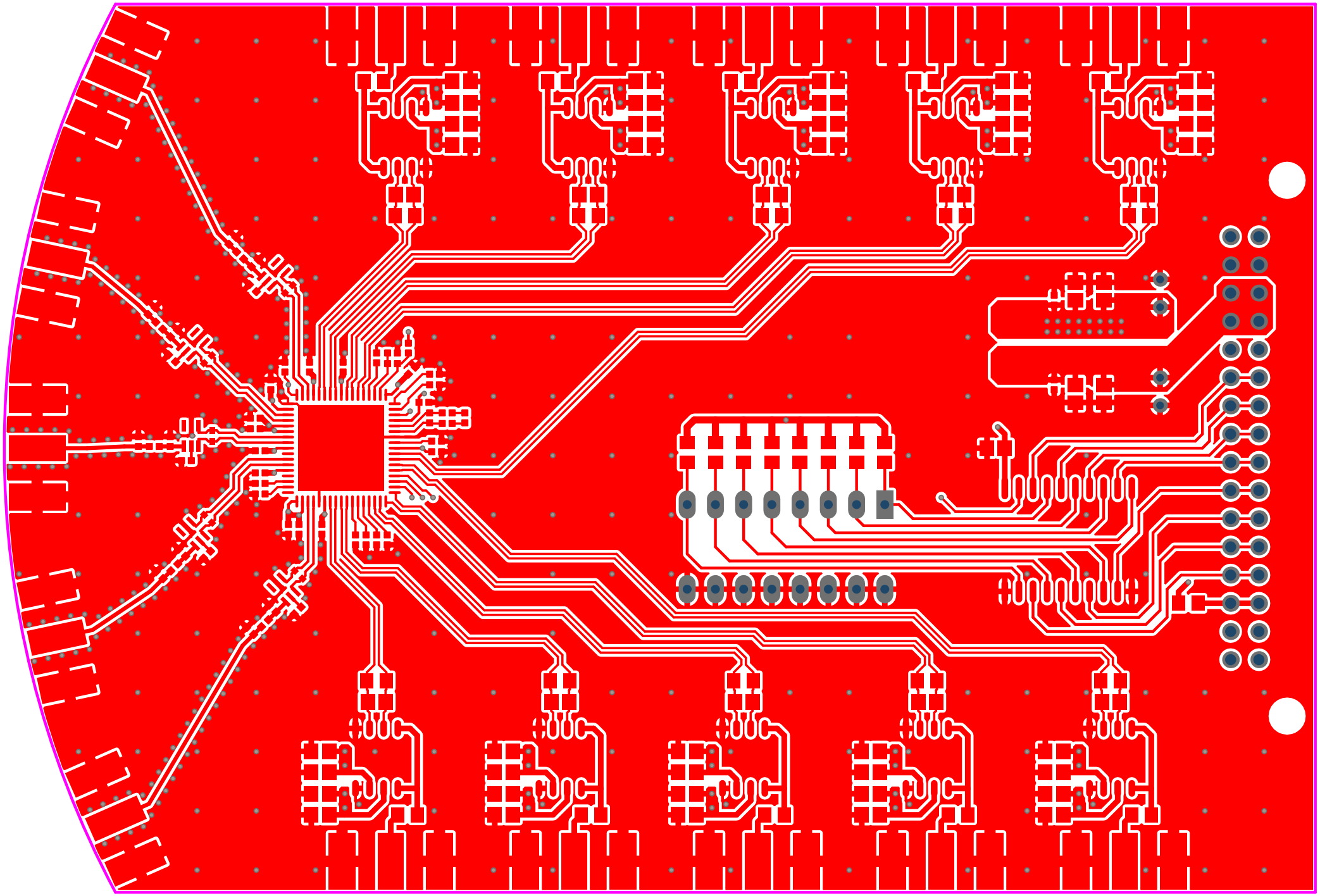
Title	Downconverter 5CH 5.8GHz	
Size A3	Number	Revision
Date: 30-05-2017		Sheet 1 of 3
File: C:\Users\..\Esquematico.SchDoc		Drawn By: Diego Benavente

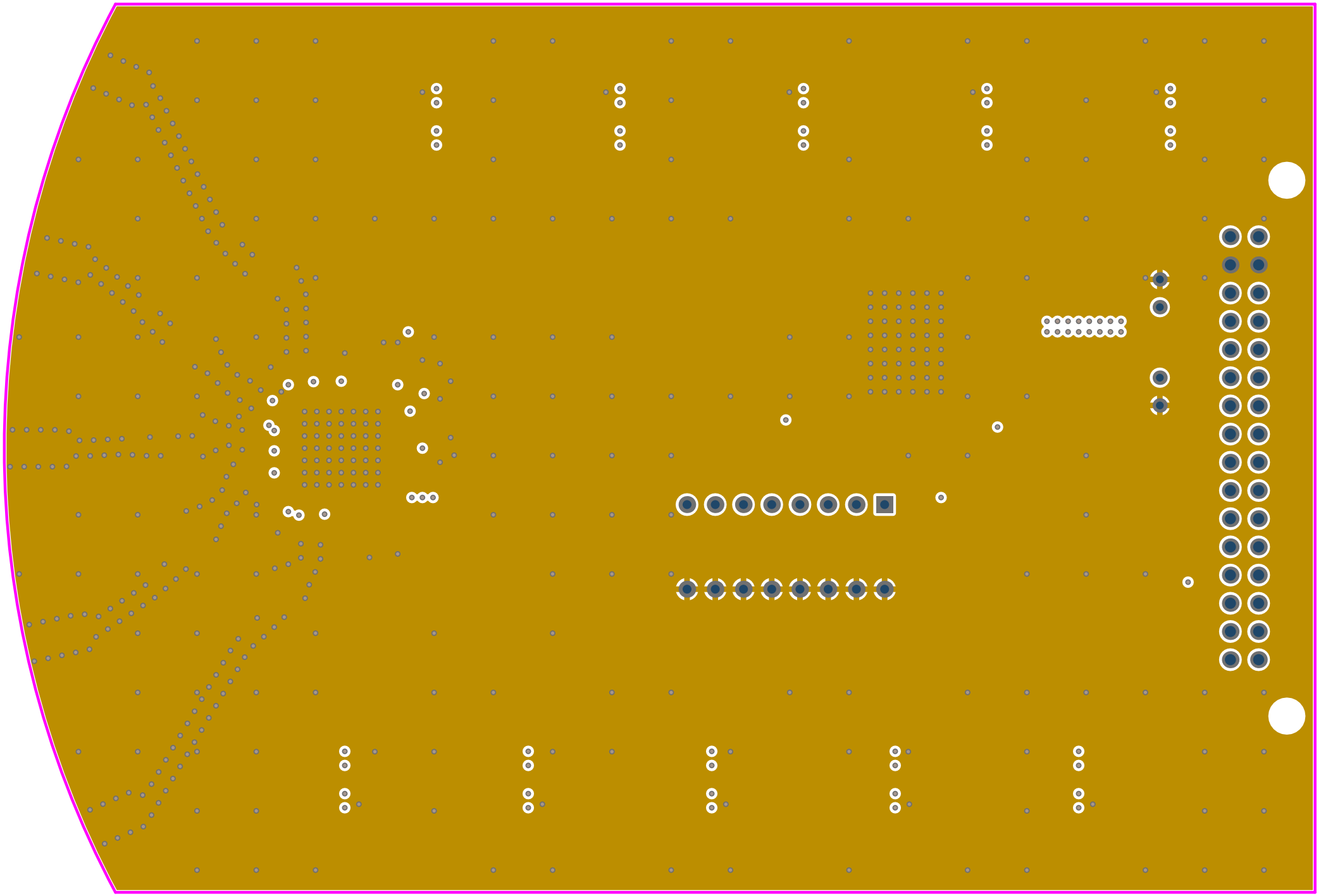


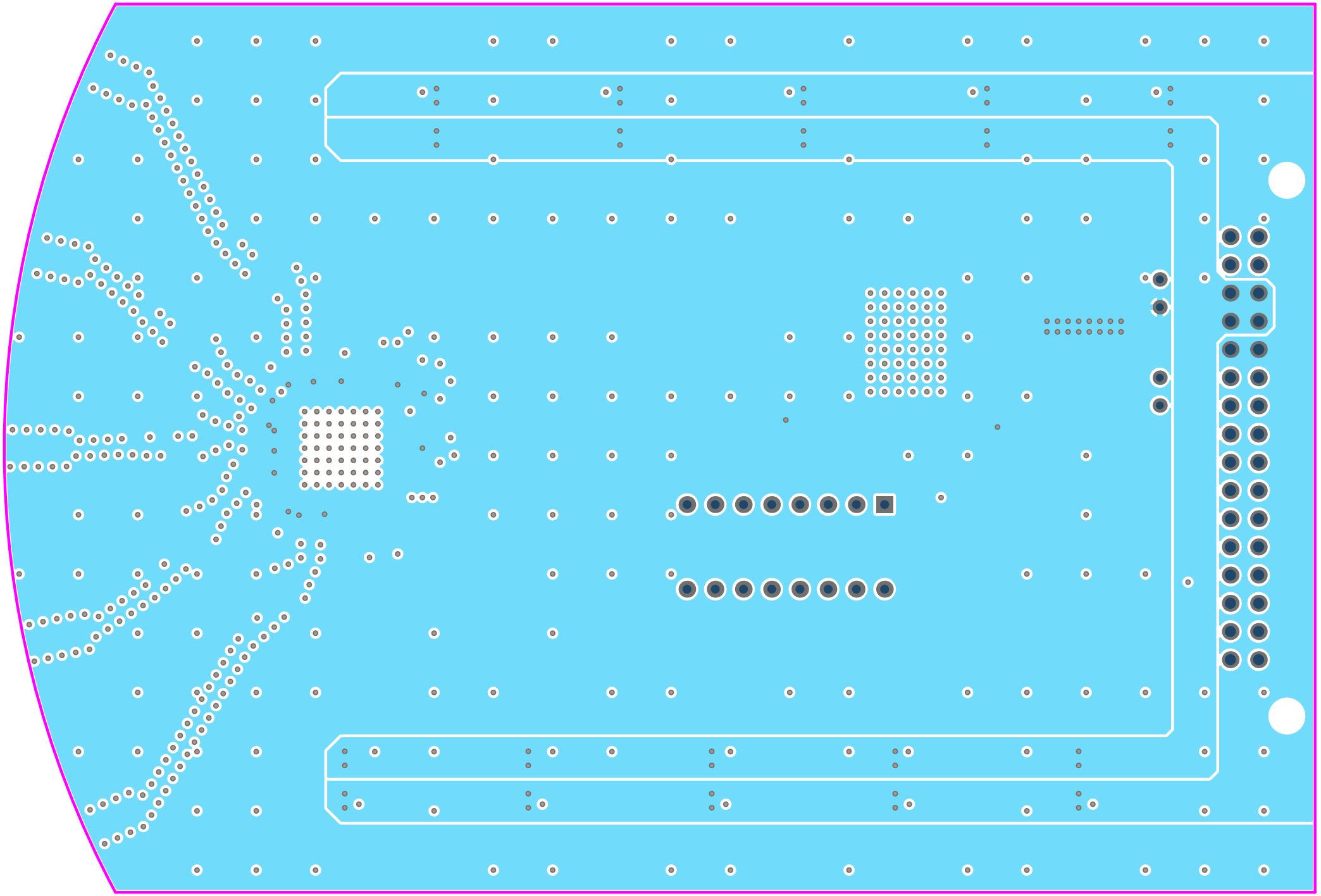
Title: Canal IF			
Size: Letter	Number:		Revision:
Date: 30-05-2017		Sheet 2 of 3	
File: C:\Users\..\canal-IF.SchDoc		Drawn By: Diego Benavente	

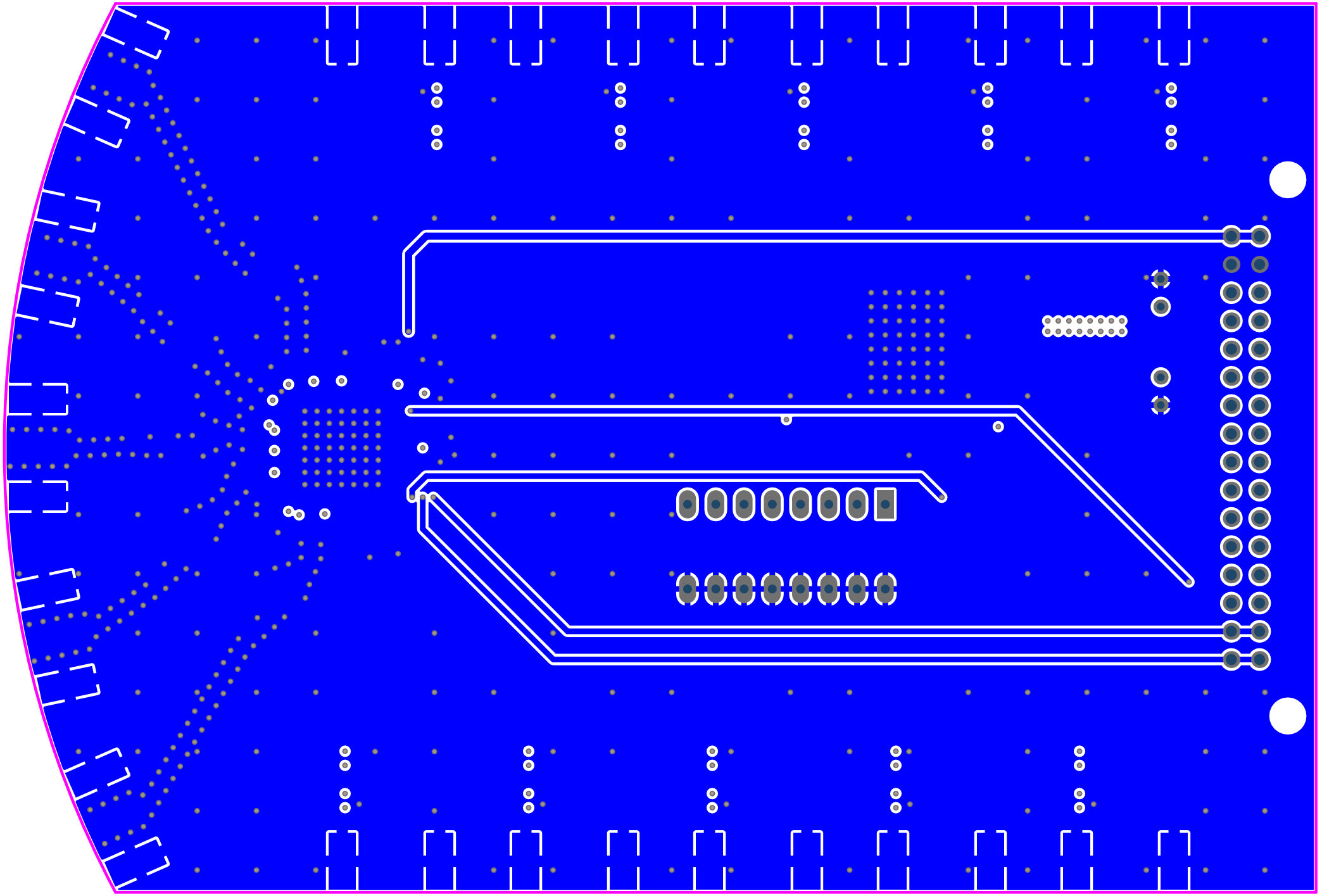


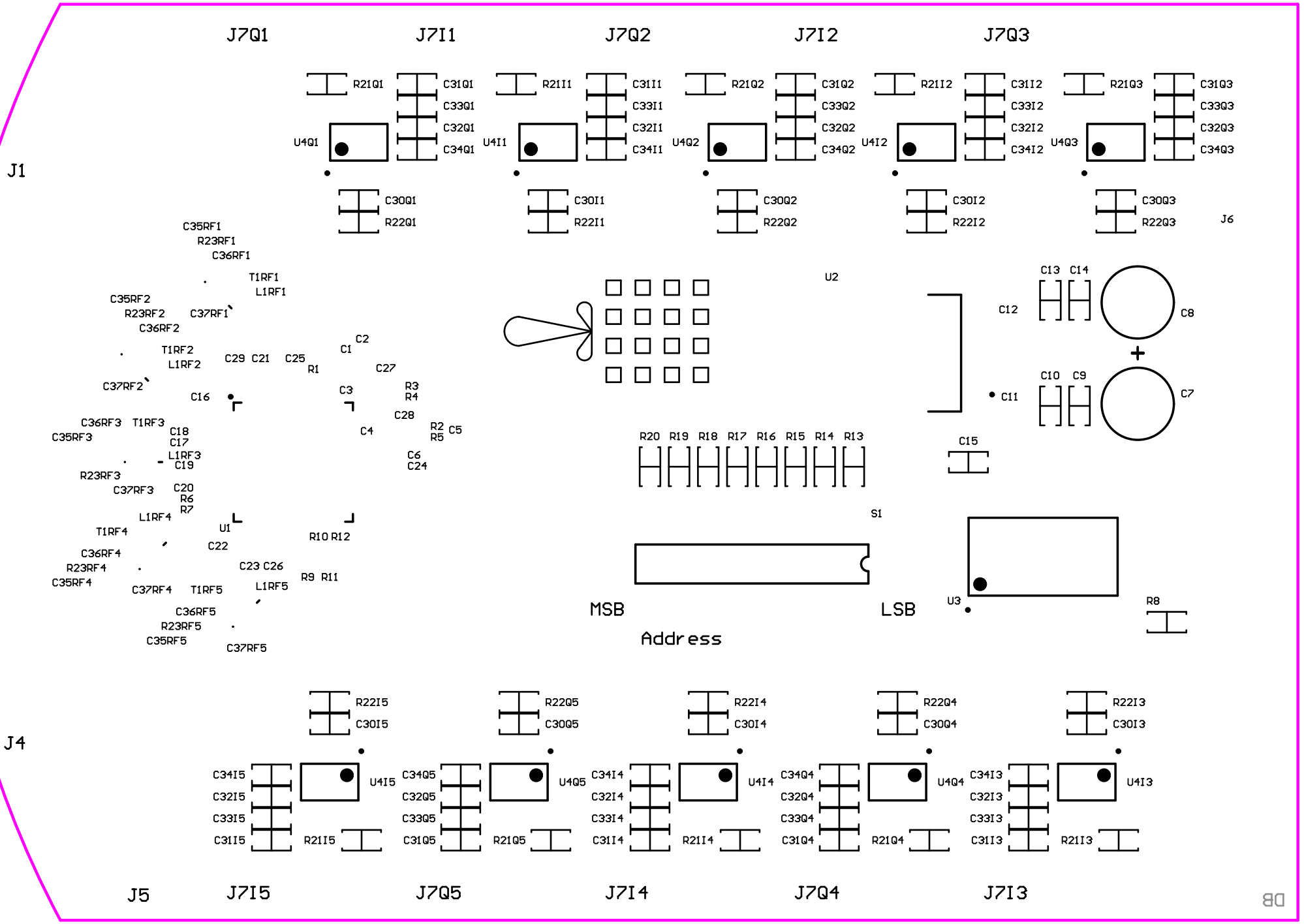
Title	Canal RF		
Size	Number		Revision
Letter			
Date:	30-05-2017	Sheet 3 of 3	
File:	C:\Users\..\canal-RF.SchDoc	Drawn By:	Diego Benavente

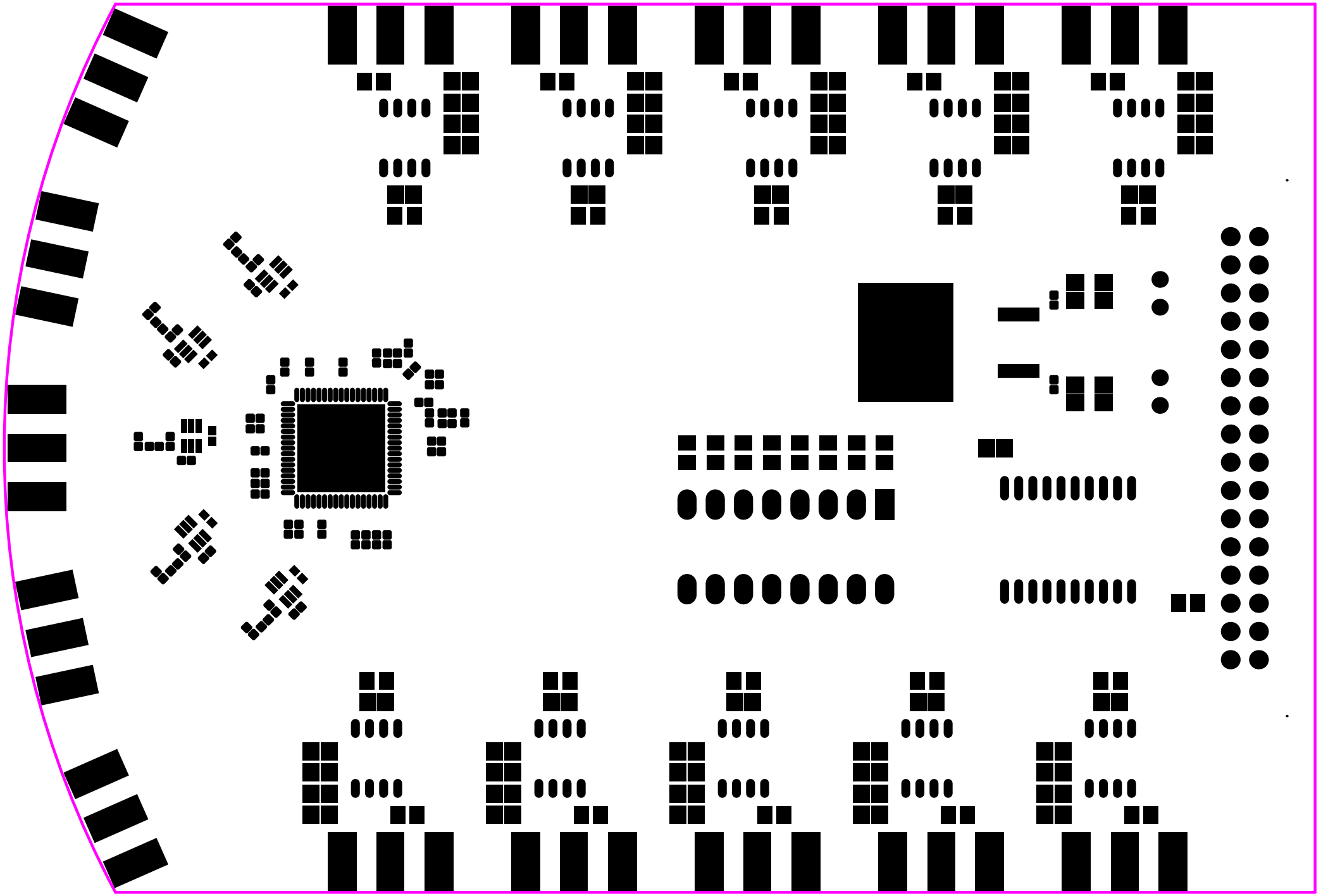


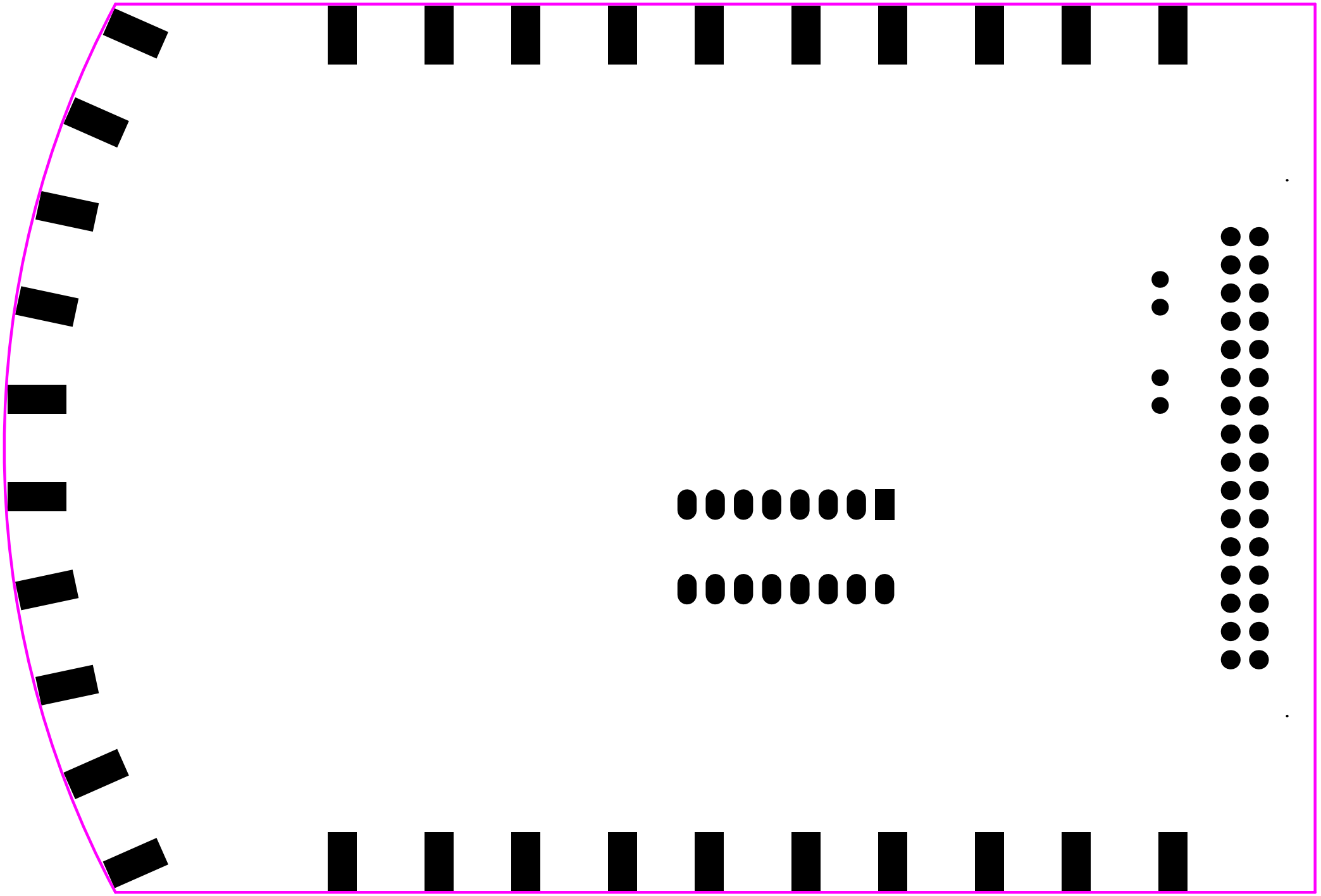


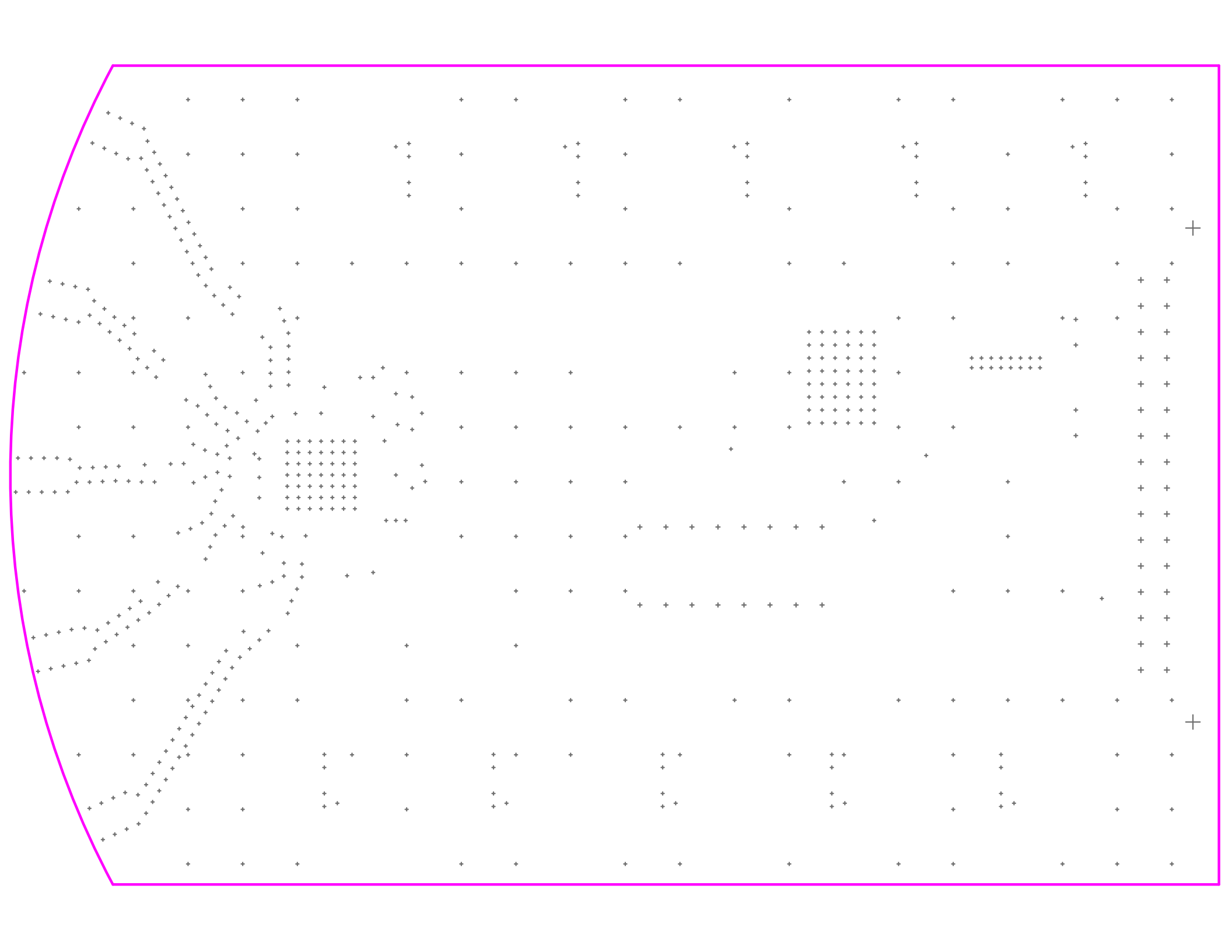


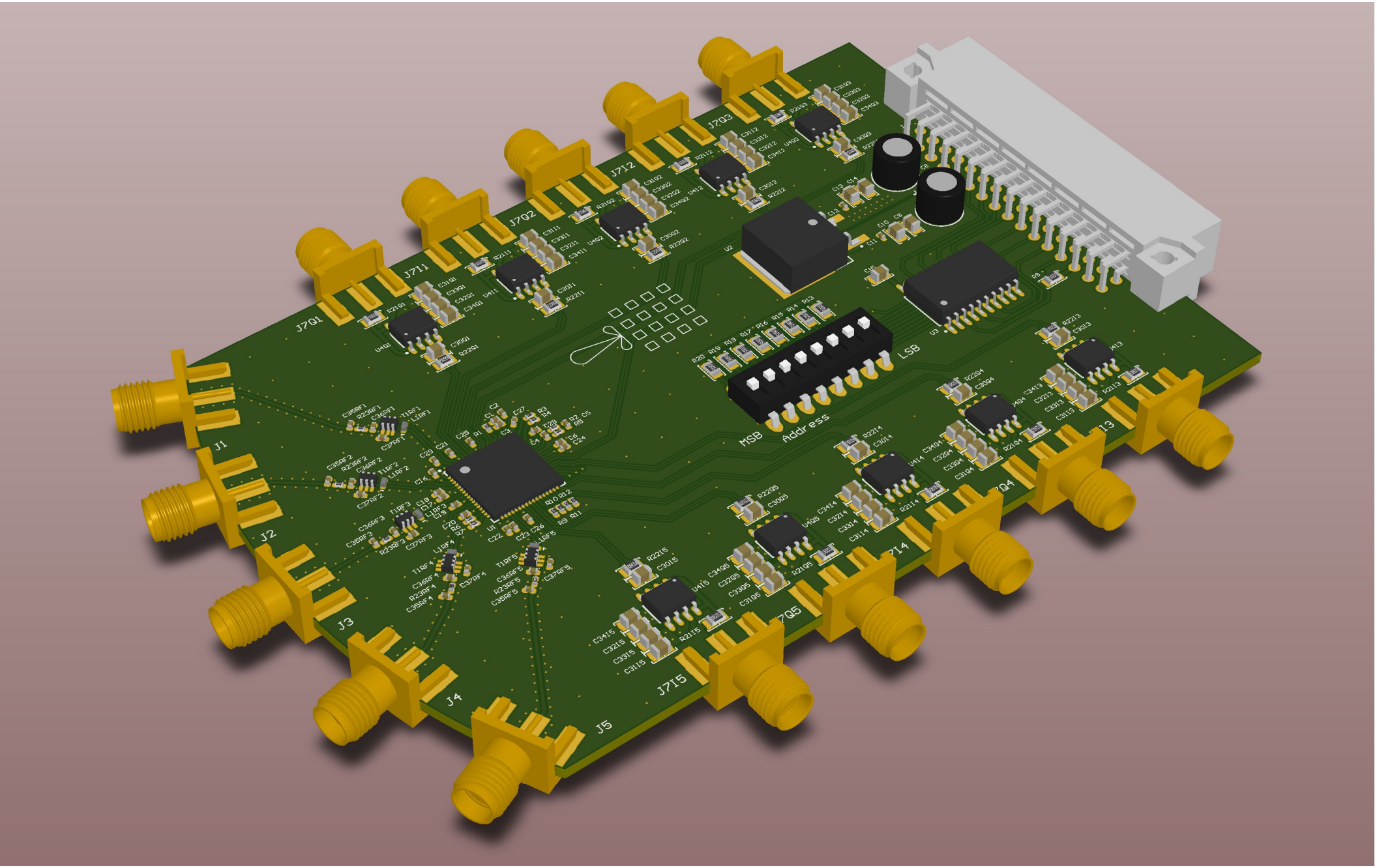



J7Q1
J7I1
J7Q2
J7I2
J7Q3
J1
J2
J3
J4
J5
J6
R21Q1
C31Q1
C33Q1
C32Q1
C34Q1
U4Q1
C30Q1
R22Q1
R21I1
C31I1
C33I1
C32I1
C34I1
U4I1
C30I1
R22I1
R21Q2
C31Q2
C33Q2
C32Q2
C34Q2
U4Q2
C30Q2
R22Q2
R21I2
C31I2
C33I2
C32I2
C34I2
U4I2
C30I2
R22I2
R21Q3
C31Q3
C33Q3
C32Q3
C34Q3
U4Q3
C30Q3
R22Q3
C35RF1
R23RF1
C36RF1
T1RF1
L1RF1
C35RF2
R23RF2
C37RF1
C36RF2
T1RF2
L1RF2
C29
C21
C25
C2
C1
R1
C27
C37RF2
C16
C3
R3
R4
C28
C36RF3
T1RF3
C35RF3
C18
C17
C4
R2
R5
C5
L1RF3
C19
C6
C24
R23RF3
C37RF3
C20
R6
R7
L1RF4
T1RF4
U1
C22
R10
R12
C36RF4
R23RF4
C35RF4
C23
C26
R9
R11
C37RF4
T1RF5
L1RF5
C36RF5
R23RF5
C35RF5
C37RF5
U2
C13
C14
C12
C8
C10
C9
C11
C7
C15
R20
R19
R18
R17
R16
R15
R14
R13
S1
MSB
LSB
Address
U3
R8
R22I5
C30I5
C34I5
C32I5
C33I5
C31I5
U4I5
R21I5
R22Q5
C30Q5
C34Q5
C32Q5
C33Q5
C31Q5
U4Q5
R21Q5
R22I4
C30I4
C34I4
C32I4
C33I4
C31I4
U4I4
R21I4
R22Q4
C30Q4
C34Q4
C32Q4
C33Q4
C31Q4
U4Q4
R21Q4
R22I3
C30I3
C34I3
C32I3
C33I3
C31I3
U4I3
R21I3
J7I5
J7Q5
J7I4
J7Q4
J7I3








J1
J2
J3
J4
J5
J7Q1
J7Q2
J7Q3
J7Q4
J7Q5
J7I1
J7I2
J7I3
J7I4
J7I5
LSB
MSB
Address