Universidad de la República
Facultad de Ingeniería

Implementación en un FPGA de la etapa de sincronismo de un receptor OFDM para recepción de señales de DTV del estándar ISDB-T

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Daniel Contrera, Florencia Ferrer

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Pablo Belzarena	Universidad de la República
Leonardo Etcheverry	Universidad de la República
Julio Perez	Universidad de la República

Tribunal

Federico La Rocca	Universidad de la República
Julio Perez	Universidad de la República
Leonardo Steinfeld	Universidad de la República

Montevideo,
miércoles 19 octubre, 2016

Implementación en un FPGA de la etapa de sincronismo de un receptor OFDM para recepción de señales de DTV del estándar ISDB-T, Daniel Contrera, Florencia Ferrer.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 199 páginas.
Compilada el miércoles 19 octubre, 2016.
http://iie.fing.edu.uy/

I have a dream...

Martin Luther King

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Agradecemos a nuestros familiares y amigos por apoyarnos en esta etapa y a lo largo de nuestras carreras. A nuestros tutores Julio, Pablo y Leonardo por su apoyo durante el transcurso del proyecto. A Federico, por colaborar también con este proyecto. Y a los docentes que tuvimos durante la carrera por contribuir con nuestra formación.

Esta página ha sido intencionalmente dejada en blanco.

A Celeste, Enrique, Laura, Romina y Belén
A Myriam, Rafael, Nati y Alan

Esta página ha sido intencionalmente dejada en blanco.

Resumen

En los últimos diez años, a nivel mundial, comenzó la traslación de la televisión analógica a la televisión digital también conocida como "el apagón analógico". Aquí en Uruguay, alrededor del año 2007 comenzaron a hacerse públicas las iniciativas de que el país adoptaría esta nueva tecnología. A fines del 2010 se había decidido qué estándar adoptaríamos: el estándar *Integrated Services Digital Broadcasting - Terrestial* (ISDB-T) al igual que Brasil.

Paralelamente, el avance de la tecnología en el área de las comunicaciones inalámbricas impulsaba cada vez más el uso de las radios definidas por *software* (SDRs, por su sigla en inglés). Esta tecnología permitiría a entusiastas tanto profesionales como *amateurs* del área de las comunicaciones por radio frecuencia, implementar múltiples sistemas de radio desde su PC (evitando el desarrollo del *hardware*) con tan sólo una placa que adquiriera los datos y los enviara por USB a la PC. A principios de siglo, aparecería una plataforma libre para el desarrollo de los SDR conocida como GNU Radio que luego crearía una comunidad internacional.

La coyuntura de estos dos mundos fue lo que originó el módulo en GNU Radio: `gr-isdbt`. Este es un sistema en código abierto de un receptor *full-seg* de Televisión Digital (DTV, por su sigla en inglés) del estándar ISDB-T, el cual fue desarrollado en el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería de la Universidad de la República por el grupo ARTES. El mismo permite realizar diferentes mediciones a lo largo de toda la cadena de procesamiento, evitando equipamiento costoso.

El presente documento describe la implementación en un FPGA (*Field Programmable Gate Array*, por su sigla en inglés) del primer bloque de la etapa de sincronización del receptor `gr-isdbt`: el bloque `OFDM Sym Acquisition` y a su vez, la implementación del protocolo de comunicaciones para el pasaje de los símbolos a la salida del mismo a la PC con el sistema `gr-isdbt` sin su primer bloque. El diseño en el FPGA se hizo en la placa bladeRF desarrollada por Nuand, un SDR de costo medio con grandes ventajas para este trabajo: código y esquemáticos disponibles para el usuario, comunidad activa, ancho de banda y frecuencia de muestreo definidos por la norma, soportados por el dispositivo, entre otros.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 1

Introducción

En el presente documento se describe el proyecto de grado *Implementación en un FPGA de la etapa de sincronismo de un receptor OFDM para recepción de señales de DTV del estándar ISDB-T*. El mismo consistió en la implementación en un FPGA (*field-programmable gate array*) del primer bloque de la etapa de sincronismo de un demodulador OFDM (*Orthogonal Frequency Division Multiplexing*, por su sigla en inglés) para un receptor de Televisión Digital del estándar ISDB-T [27] sobre un sistema de radio definido por *software* (SDR). Esto implicó también la definición e implementación de un protocolo de comunicaciones para el envío de símbolos OFDM del FPGA hacia la PC, en la cual corre el *software* del sistema y se realiza el resto del procesamiento.

El proyecto surge con el motivo de analizar las ventajas y las desventajas de la implementación parcial en *hardware* de un sistema SDR, particularmente de un receptor de DTV del estándar ISDB-T. Un punto a analizar específicamente corresponde al estudio de la posible reducción del procesamiento correspondiente a la PC.

Teniendo en cuenta que se trabajaría con el receptor de DTV, se tomó como base la implementación completa del sistema desarrollado sobre GNU Radio. Luego, se definió realizar la implementación de los primeros bloques de la cadena sobre *hardware*. En particular, se determinó utilizar una placa de *hardware* que disponga de una cantidad de recursos de lógica, memoria y procesamiento necesarios para desarrollar la implementación.

Por lo tanto, fue necesario el estudio e investigación de los sistemas sobre los que se trabajó y las herramientas necesarias para el desarrollo. Por otra parte, la implementación del protocolo de comunicaciones entre el bloque de *hardware* y el resto del sistema sobre la PC. Y finalmente la implementación del bloque sobre el FPGA.

Debido a esto, el proyecto se realizó en tres grandes etapas que se explican a continuación. Una primera de investigación en la cual se estudiaron los sistemas de radio definidos por *software*. En particular, la plataforma de desarrollo y procesamiento sobre *software* conocida como GNU Radio, el módulo `gr-isdbt`: un receptor *full-seg* de Televisión Digital del estándar ISDB-T implementado en GNU Radio. También se estudió el *hardware* de los sistemas de radio que son

responsables de la adquisición de las señales del aire y el envío a la PC para su procesamiento en GNU Radio o un programa similar. En particular, se profundizó en la placa bladeRF ya que fue la plataforma de *hardware* utilizada para el proyecto. Se estudió su funcionamiento enfatizando en el flujo de datos y en el sistema alojado en el FPGA de la placa bladeRF, al cual de ahora en más, llamaremos arquitectura del FPGA. Esto permitió definir de que forma se podría agregar una cadena de procesamiento sobre el FPGA y qué cambios serían necesarios para el envío de símbolos hacia la PC en lugar de muestras independientes.

La segunda etapa del proyecto consistió en la definición de un protocolo de comunicaciones para el manejo de símbolos entre el *hardware* y el *software* y los elementos necesarios para su implementación en ambas plataformas. En la bladeRF, se implementó un señalizador de símbolos OFDM dentro del FPGA, y sobre GNU Radio, un bloque para la lectura de los mismos.

Por último para la tercera etapa, el objetivo inicial del proyecto era implementar los tres primeros bloques del módulo `gr-isdbt` correspondientes a la etapa de sincronismo: `OFDM Sym Acquisition`, `FFT` y `Sync And Channel Estimation`. Esto no fue posible debido a que la investigación llevó más tiempo del estimado y surgió la necesidad de la implementación de los bloques de comunicaciones que en un principio no se habían previsto. Posteriormente, se redujo el alcance del proyecto a la implementación en el FPGA de tan solo el primer bloque: `OFDM Sym Acquisition`. En la etapa de investigación esta reducción aún no se había definido por lo que la misma abarcó los tres bloques, para la definición de la arquitectura del FPGA sucedió lo mismo. En este sentido, se observará que ciertos aspectos de la arquitectura consideran la implementación inicial.

En este sentido, el documento abarca el presente capítulo introductorio, un segundo capítulo donde se plantearán los elementos investigados que fueron necesarios posteriormente para la implementación: el estándar ISDB-T, los sistemas de radio definidos por *software* (tanto *software* como *hardware*. En el tercer capítulo se estudia el sistema original como solución actual, incluyendo los puntos `gr-isdbt`, bladeRF y la interfaz bladeRF - GNU Radio. En los capítulos cuarto y quinto se presentan las modificaciones del sistema original sobre las comunicaciones y el *hardware* respectivamente. Para las comunicaciones se abarca la definición de un protocolo de comunicaciones y las modificaciones realizadas sobre *software* y *hardware*. En el caso del *hardware* abarcará la definición de una arquitectura de procesamiento en el FPGA y la implementación del primer bloque de sincronismo del receptor OFDM: el bloque `sym_acquisition`. En el sexto capítulo se presentan las pruebas realizadas para la validación del sistema en su conjunto. Por último, en el capítulo séptimo se presentan las conclusiones del proyecto.

Capítulo 2

Fundamentos Teóricos

En el presente trabajo se estudia la implementación de bloques de procesamiento de señales sobre *hardware*, para la recepción de señales OFDM (*Orthogonal Frequency Division Multiplexing*, por su sigla en inglés) correspondientes a televisión digital partiendo de un sistema original que realiza todo el proceso de sincronización y decodificación en *software*.

En este sentido, en el presente capítulo se abarcarán aquellos temas que comprenden el fundamento teórico del problema a resolver. En primer lugar, se realiza una introducción al estándar de DTV (*Digital TeleVision*, por su sigla en inglés) ISDB-T, en el cual se basa la implementación ya que es utilizado en Uruguay y en la región. También se mencionan las características particulares que resultan relevantes. En segundo lugar, se estudiarán los sistemas de radio definidos por *software* diferenciando los elementos que comprenden el *software* de los que comprenden el *hardware*.

2.1. Estándar ISDB-T

El estándar *Integrated Services Digital Broadcasting - Terrestial* (ISDB-T [27]) es un estándar japonés de televisión digital, creado y mantenido por Association of Radio Industries and Businesses (ARIB). Está basado en el estándar European Digital Video Broadcasting (DVB [13]) adoptado por la mayoría de los países del mundo (Europa, algunos países de África, Asia y Oceanía). El mismo fue adaptado en Brasil con el nombre ISDB-Tb y posteriormente adoptado por varios países de la región, incluyendo Uruguay. Actualmente lo utilizan varios canales de televisión como: La Tele, Monte Carlo, TV Ciudad y Saeta en la ciudad de Montevideo, Uruguay. El estándar permite multiplexar varios servicios sobre un mismo canal de ancho de banda de 6 MHz, utilizando la modulación por portadoras OFDM. Para esto, en transmisión se aplica un diagrama de bloques como la Figura 2.1 a la señal a transmitir. En el mismo se aplican técnicas de entrelazado, adición de portadoras auxiliares e intervalo de guarda para agregar redundancia. El bloque de IFFT permite obtener el multiplexado en frecuencia mediante subportadoras ortogonales.

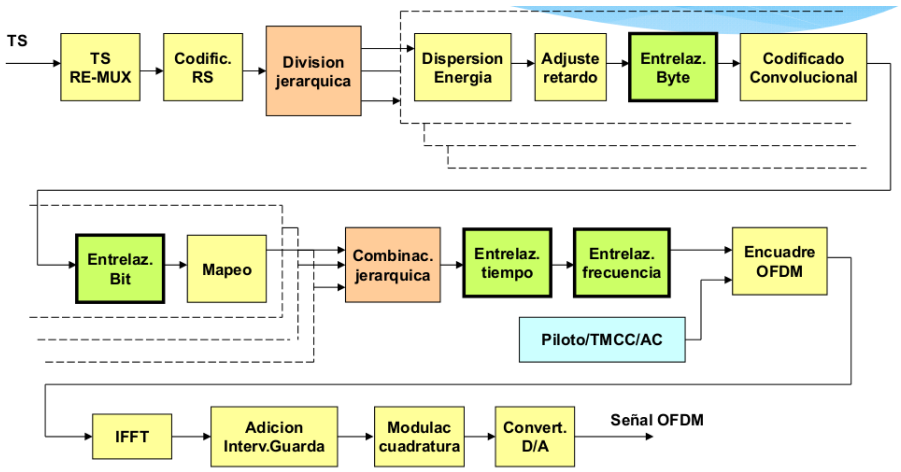


Figura 2.1: Diagramas de bloques sistema ISDB-T [25].

2.1.1. OFDM

El intervalo de guarda, se denomina Prefijo Cíclico (CP por su sigla en inglés). El mismo corresponde a una copia de la primera parte del símbolo OFDM (1/4, 1/8, 1/16 o 1/32 del total) la cual se ubica a continuación del mismo. Esto permite realizar en recepción la alineación con el símbolo OFDM y una parte de la corrección en frecuencia, además de eliminar la interferencia intersimbólica (ISI). En la Figura 2.2 se visualiza un esquema de un símbolo y la copia del CP. En la misma la duración del símbolo es T_s y la del CP $T_g = 1/4 * T_s$.

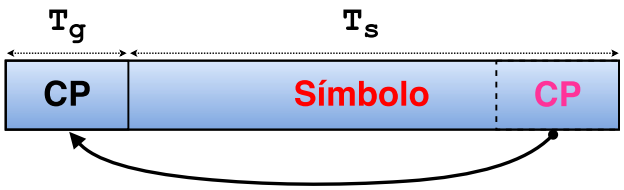


Figura 2.2: Símbolo con prefijo cíclico.

En la modulación se utiliza un largo de símbolos potencia de 2 para poder realizar el algoritmo de la Transformada Rápida de Fourier (FFT por sus siglas en inglés). Los valores están determinados por el Modo de Transmisión, los mismos son 2^{11} (modo 1), 2^{12} (modo 2) y 2^{13} (modo 3). La frecuencia de muestreo del estándar está fija en el valor $f_s = 512/63$ MHz $\approx 8{,}13$ MHz. El modo 3 de transmisión utiliza 5617 portadoras (*active carriers*) de las 8192 disponibles (*total carriers*). Las mismas se agrupan en 13 segmentos de 432 portadoras (existe una portadora particular ubicada fuera de los segmentos) separados en el espectro como se observa en la Figura 2.3.

Los segmentos son independientes entre sí pero se pueden agrupar en tres capas jerárquicas para transmitir distintos flujos de datos, utilizando distintos esquemas de modulación, esta técnica se conoce como Transmisión Jerárquica por

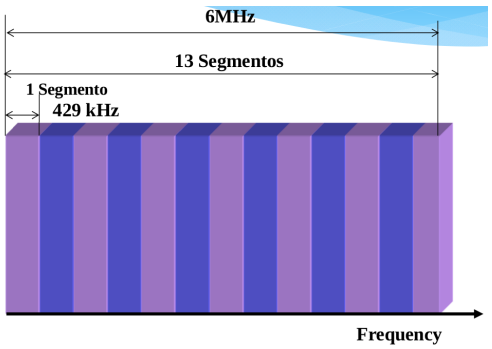


Figura 2.3: Agrupamiento de segmentos OFDM [25].

OFDM Segmentados (*Base Segment Transmission* BST-OFDM). El segmento central (segmento 0) se utiliza para recepción parcial. Luego, los demás segmentos se enumeran de forma ascendente y alternada alrededor del segmento central. Estos últimos pueden utilizar distintos esquemas de modulación, en el caso de los segmentos concentrados junto al segmento central pueden utilizar una modulación diferencial DQPSK, los restantes se ubican hacia los bordes del espectro y pueden utilizar una modulación coherente (QPSK, 16QAM o 64QAM).

Las capas de segmentos pueden tener diferentes características de acuerdo a los distintos requerimientos de la transmisión. Por ejemplo, pueden utilizar distintas combinaciones de tasa de corrección de errores, profundidad de entrelazado o cantidad de segmentos. En el caso de DTV, el segmento central del espectro se utiliza para transmitir vídeo de calidad estándar (SD) conformando la capa A y los restantes 12 segmentos se utilizan para transmitir vídeo en alta calidad (HD) conformando la Capa B.

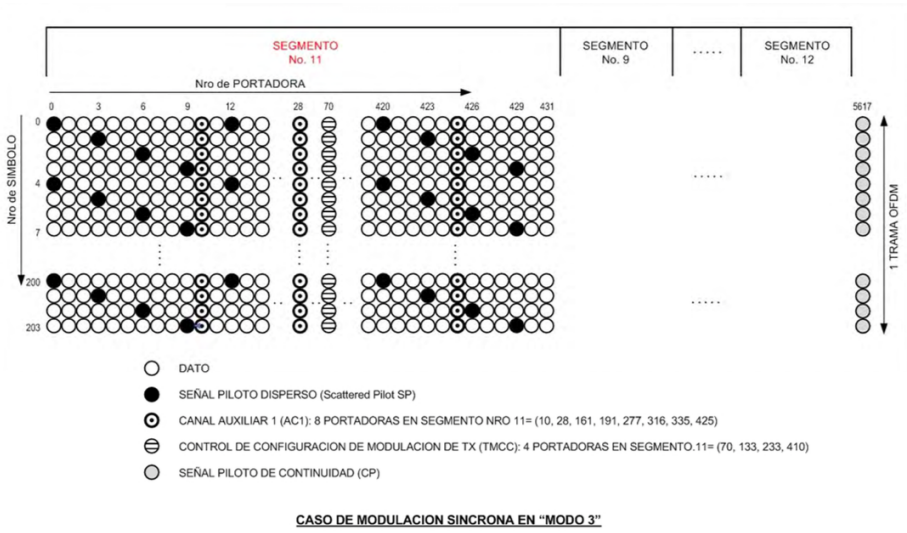


Figura 2.4: Distribución de portadoras para una trama OFDM [25].

Algunas de las portadoras transmitidas son utilizadas como pilotos para la

ecualización en recepción debido a los efectos del canal, pilotos espurios (SP por su sigla en inglés) y otras para enviar información de control (portadoras para *Transmission Multiplexing Configuration Control* TMCC). En el estándar sólo hay una portadora continua (CP), las SP varían de forma cíclica y las de TMCC varían de acuerdo a la información particular que se transmite para control. En la Figura 2.4 se observa la distribución de las distintas portadoras a lo largo de cada símbolo y a lo largo de una trama.

Las portadoras SP son transmitidas con valores de amplitud y fase conocidos, con modulación BPSK, de forma de poder estimar la transferencia del canal y compensar su efecto. Las mismas tienen ubicaciones predefinidas pero varían entre símbolos consecutivos entre cuatro posibilidades cíclicas. Las portadoras TMCC se utilizan para transmitir un mensaje compuesto por 204 *bits* a lo largo de una trama OFDM la cual consta de 204 símbolos. Las mismas se modulan en DBPSK sobre ubicaciones predefinidas. Cada mensaje TMCC contiene 102 *bits* de información de control los cuales son comunes a cada portadora TMCC de la trama, los restantes *bits* corresponden a información de sincronismo y corrección de errores que pueden variar para cada portadora de forma cíclica. Los mismos permiten enviar información referente a la transmisión como por ejemplo los parámetros de transmisión de cada capa.

En la Tabla 2.1 se observan algunos parámetros del estándar ISDB-T, por más detalles referirse a [27].

Parámetro	**Valor**
Ancho de banda total	6 MHz
Frecuencia de Muestreo	$f_s = 512/63$ MHz $\approx$ 8,13 MHz
Cantidad de muestras símbolo activo	2048 (modo 1) 4096 (modo 2) 8192 (modo 3)
Duración de símbolo activo	$252\mu s$ (modo 1) $504\mu s$ (modo 2) $1008\mu s$ (modo 3)
Duración de intervalo de guarda (CP)	1/4, 1/8, 1/16, 1/32 (de la duración de símbolo activo)
Frecuencia de código convolucional	1/2, 2/3, 3/4, 5/6, 7/8
Profundidad de entrelazado temporal	0, 1, 2, 4 (modo 1) 0, 2, 4, 8 (modo 2) 0, 4, 8, 16 (modo 3)
Esquemas de modulación	DQPSK, QPSK, 16QAM, 64QAM

Tabla 2.1: Parámetros de transmisión del estándar ISDB-T.

Los canales de DTV en Uruguay transmiten con los parámetros de la Tabla 2.2 sobre la banda UHF (utilizando el modo 3 de transmisión).

Nombre	Frecuencia Central (MHz)	CP	Capa B	
			Profundidad	Código
La Tele	557.143	1/8	2	2/3 3/4
Monte Carlo	563.143	1/8	4	2/3
TNU	569.143	1/16	2	3/4
Saeta	575.143	1/8	2	2/3
Tv Ciudad	581.143	1/16	2	3/4

Tabla 2.2: Parámetros de transmisión de canales de televisión en Montevideo, Uruguay.

2.1.2. Sincronismo

Como sucede con todo sistema de comunicación, existen no idealidades que llevan a que la señal recibida no sea la misma que fue transmitida. Para este sistema en particular, la señal recibida se puede modelar con la siguiente ecuación:

$$r[k] = s[k-\theta]e^{\frac{j2\pi\epsilon k}{N}} + n[k], \tag{2.1}$$

con $s[k]$ la señal transmitida y $n[k]$ el ruido AWGN producido por el canal y la recepción de la señal. El momento de llegada del símbolo OFDM (θ) y el *offset* en frecuencia de las portadoras (ϵ) son las dos parámetros que debe resolver el presente bloque. Ambos son producidos por falta de sincronización de la frecuencia de muestreo debido a pequeñas diferencias en los osciladores locales entre el transmisor y el receptor y a efectos del canal.

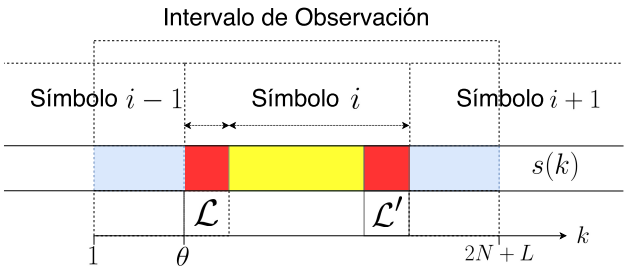


Figura 2.5: Estructura de señal OFDM con CP ($s(k)$). El conjunto $\mathcal{L}$ contiene el prefijo cíclico, los datos originales que se copiaron en $\mathcal{L}$ se encuentran en $\mathcal{L}'$ [17].

Para hallar θ y ϵ van de Beek et al. [17] propusieron un algoritmo que consiste en observar un intervalo de 2N + L muestras consecutivas (con N la cantidad de muestras en un símbolo y L la cantidad de muestras del prefijo cíclico (CP)) clasificándolas en los dos siguientes grupos (ver Figura 2.5):

$$\mathcal{L} = \{\theta, \cdots, \theta + L - 1\},$$
$$\mathcal{L}' = \{\theta + N, \cdots, \theta + N + L - 1\},$$

siendo $\mathcal{L}$ el conjunto de índices de las muestras del CP y $\mathcal{L}'$ el conjunto de índices de los datos originales que fueron copiados al CP. De esta división en conjuntos, se observa que existe una correlación entre los mismos:

$$\forall k \, \epsilon \, \mathcal{L} : E\{r[k]r^*[k+m]\} = \begin{cases} \sigma_s^2 + \sigma_n^2 & m = 0 \\ \sigma_s^2 e^{-j2\pi\epsilon} & m = N \\ 0 & \textit{en otro caso.} \end{cases}$$

De lo anterior, en [17] proponen el siguiente algoritmo de máxima verosimilitud en función de los parámetros buscados:

$$\Lambda(\theta, \epsilon) = |\gamma(\theta)| \cos(2\pi\epsilon + \angle\gamma(\theta)) - \rho\Phi(\theta), \tag{2.2}$$

con:

$$\gamma(m) = \sum_{k=m}^{m+L-1} r[k]r^*[k+N],$$

$$\Phi(m) = \sum_{k=m}^{m+L-1} \frac{|r[k]|^2 + |r[k+N]|^2}{2},$$

y $\rho = \sigma_s^2/(\sigma_s^2 + \sigma_n^2) = SNR/(1+SNR)$.

Por último, maximizando la Ecuación 2.2 en el intervalo de observación se obtienen los parámetros θ y ϵ buscados. En particular, una optimización de dos pasos se puede llevar a cabo en donde ϵ es hallado en función de θ:

$$\widehat{\epsilon}_{EMV} = -\frac{1}{2\pi}\angle\gamma(\widehat{\theta}_{EMV}), \tag{2.3}$$

con $\widehat{\theta}_{EMV}$:

$$\widehat{\theta}_{EMV} = arg_\theta max \, |\gamma(\theta)| - \rho\Phi(\theta). \tag{2.4}$$

Utilizando el resultado anterior, la resolución del problema de sincronización temporal se divide en dos etapas en las cuales se calcula $\widehat{\theta}_{EMV}$ en distintos intervalos de observación. Las mismas se podrían identificar como una etapa de procesamiento transitorio y otra de régimen. En el transitorio, cuando la sincronización aún no se ha establecido (por ejemplo al comienzo de la adquisición), la Ecuación 2.4 se calcula para toda la ventana en observación ($2N + L$ muestras), a partir del θ hallado (siendo $\theta = \widehat{\theta}_{EMV}$) se calcula ϵ mediante la Ecuación 2.3. Una vez obtenido el valor de θ el símbolo corresponde a las muestras en el rango $[\theta + L, \theta + L + N]$, descartando las muestras del conjunto $\mathcal{L}$. Finalmente las muestras del símbolo son rotadas para realizar el ajuste en frecuencia (cada muestra es "rotada" en el plano complejo).

La rotación consiste en aplicar una diferencia de fase a cada muestra de forma de compensar el *offset* ϵ. Para realizar esto la diferencia que se aplica sobre el símbolo debe crecer linealmente según el valor de $-2\pi\epsilon/N$ entre muestras consecutivas. De esta forma, la diferencia de fases resultante entre las muestras del conjunto $\mathcal{L}$ y las de $\mathcal{L}'$ será aproximadamente nula (en en caso ideal deberían ser idénticas).

Por otro lado, para el caso régimen (la sincronización ya se estableció), para calcular ϵ se hace uso de que la posición del máximo en la Ecuación 2.4 no debería variar significativamente de símbolo a símbolo por lo que se calcula θ en un intervalo estimado en función al hallado previamente. En caso de que no se encontrara un resultado aceptable (por ejemplo, el máximo es demasiado pequeño para un piso establecido o se encuentra en un borde del intervalo) se trata como un caso transitorio, si no se tuviera éxito, se descartan las muestras y se comienza nuevamente.

2.2. Sistemas de Radio definidos por *Software* - SDR

Los sistemas de radio definidos por *software* de ahora en más SDR son sistemas de comunicación por radio implementados en *software*. En este sentido, tienen por objetivo implementar en *software* la mayor cantidad posible de elementos que típicamente estarían en *hardware*, (por ejemplo, filtros, amplificadores, moduladores, etc). Sin embargo, cierto *hardware* básico es necesario para sintonizar y muestrear la señal de interés que luego será procesada de manera digital, este es comúnmente conocido bajo la misma sigla SDR. De manera análoga también sirven para transmitir datos. En este sentido, un mismo SDR puede ser utilizado para numerables aplicaciones tan solo cambiando de programa en la PC (siempre y cuando se cumpla con los requerimientos de cada una).

Esquemáticamente (Figura 2.6) un sistema SDR está compuesto por los siguientes elementos de RF: una antena conectada a un modulador RF que sintoniza la señal a una frecuencia intermedia (IF), un conversor AD/DA y un conversor de frecuencia (el modulador podría sintonizar la señal directamente a bandabase). Además, algunos sistemas SDR cuentan con una plataforma en *hardware* para realizar procesamiento *on board* (FPGAs, DSPs, ASICs). En última instancia se encuentra un bloque de procesamiento llevado a cabo en la PC. En la siguiente sección se presenta brevemente un programa para desarrollo de aplicaciones SDR en una PC llamado GNU Radio.

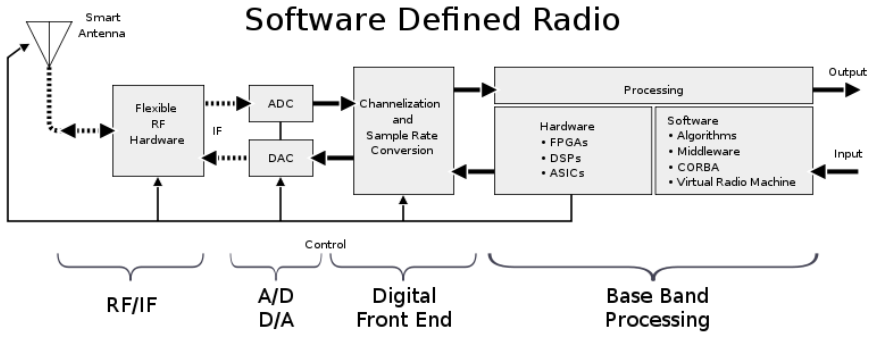


Figura 2.6: Concepto de los sistemas de radio definidos por *software* (SDR) [14].

2.2.1. GNU Radio

GNU Radio [30] es una herramienta de desarrollo de sistemas de radio definidos por *software* de código abierto, libre y gratuito. Provee una amplia librería de bloques con diversas funciones que van desde filtrado, detectores de pico, operadores matemáticos hasta moduladores OFDM y decodificadores y otras funciones genéricas de procesamiento. Los mismos pueden ser implementados tanto en lenguaje de programación Python como en el lenguaje C++. Típicamente Python se utiliza para la interconexión de bloques y C++ para el procesamiento. Mediante el entorno gráfico, GNU Radio Companion (GRC), el usuario puede instanciar los bloques interconectándolos entre sí para crear distintos SDR. Asimismo, el entorno gráfico cuenta con numerosas herramientas para visualizar señales intermedias en tiempo real y guardar datos en cualquier etapa del procesamiento. Otra gran ventaja de GNU Radio es que los sistemas SDR se pueden simular utilizando como fuente de datos señales guardadas en la PC. También permite guardar información en la PC, pudiendo salvar la misma o enviarla hacia el exterior como por ejemplo audio a través de la tarjeta de sonido o datos hacia puertos TCP/UDP. Las aplicaciones realizadas en GNU Radio son conocidas como *flowgraphs*.

Por último, pero no menos importante, mediante la función *out-of-tree modules* (módulos fuera de árbol) la plataforma permite desarrollar a los particulares nuevas funcionalidades.

2.2.2. *Hardware* SDR

Actualmente existen diversas ofertas de RF *front end* para SDR. Las mismas van de unos pocos dólares (RTL-SDR[29]) hasta unos cuantos miles (USRP X310[15]). Las principales características de este tipo de equipos son: rango de frecuencia, resolución conversor A/D, ancho de banda instantáneo, RX/TX (la radio puede transmitir y recibir), pre-selectores (filtros analógicos). El RTL-SDR por ejemplo, solamente funciona como receptor, tiene un rango de frecuencia de 24 MHz a 1766 MHz, un conversor A/D de 8 *bits* y un ancho de banda máximo estable de 2.4 o 2.8 MHz. Otra característica importante de los SDR, es si cuentan o no con algún tipo de *hardware* embebido que permite implementar funciones *on-board* y a su vez si sus fuentes se encuentran disponibles al usuario o no. La placa USRP B200, por ejemplo, cuenta con un FPGA Spartan 6 cuyo esquemático y código fuente se encuentran *online*.

2.3. Definición del Problema a Resolver

Se deberá implementar un receptor SDR de televisión digital del estándar ISDB-T cuyo procesamiento se realice parcialmente en *hardware* y en *software*. En particular, la parte en *hardware* se deberá implementar en un FPGA de una placa SDR y el procesamiento en *software* en el programa GNU Radio. En este último se trabajará con un sistema ya implementado llamado `gr-isdbt`.

En el próximo capítulo se presenta la solución actual al problema presentado y

en los dos siguientes la solución propuesta. La primera consiste en un procesamiento 100 % en *software* mientras que en el segundo caso,parte del mismo se realiza en el FPGA de un SDR.

De la solución actual se describe la plataforma de *hardware* SDR utilizada: la placa bladeRF, el proyecto `gr-isdbt` implementado en GNU Radio por el grupo ARTES de la Facultad de Ingeniería, el primer receptor abierto de TVD del estándar en cuestión, el driver de comunicaciones entre ambas interfaces y su interfaz.

De la solución propuesta se presenta tanto la interfaz de comunicaciones modificada como la implementación propiamente en el FPGA.

2.4. Conclusiones

Se presentó el estándar ISDB-T focalizando en aquellos aspectos que serán importantes para la implementación de la etapa de sincronismo del receptor. En particular los parámetros de modulación OFDM utilizada por la norma: ancho de banda, frecuencia de muestreo, prefijos cíclicos. En particular, se presentaron los parámetros que utilizan los canales de DTV de Uruguay.

Se introdujo el concepto de radio definida por *software* (SDR por su sigla en inglés) dentro de este tema se presentó GNU Radio, un entorno de desarrollo muy utilizado para los SDR.

Por último se presentó el problema a resolver y se introdujeron los próximos capítulos.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Solución Actual

En el presente capítulo se estudian los elementos principales que componen a la solución actual del problema en estudio. Por un lado, la plataforma de *hardware* utilizada, una placa SDR llamada bladeRF, por otro, dentro del *software*, el proyecto `gr-isdbt` de GNU Radio. Este último es una implementación completa de la decodificación del vídeo de la señales de televisión sobre GNU Radio. Se utilizó de referencia para el desarrollo de bloques sobre el *hardware* y se mantuvo como decodificador hasta obtener las señales de vídeo luego de la implementación parcial sobre el *hardware*.

El capítulo continua con el estudio de un segundo módulo en GNU Radio llamado `gr-osmosdr` que sirve para comunicarse con la placa bladeRF entre otras.

Por último se describe la interfaz del sistema actual el cual corresponde al uso de las versiones por defecto de los proyectos de bladeRF y `gr-isdbt` presentados en las secciones 2.2.1 y 3.1 respectivamente.

En la Figura 3.1 se observa la comunicación entre la placa bladeRF y un *flowgraph* parcial del proyecto `gr-isdbt` de GNU Radio.

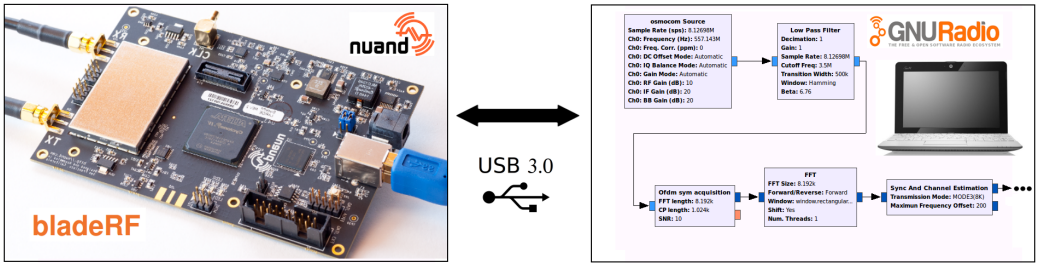


Figura 3.1: Interfaz solución ActualbladeRF - GNU Radio.

3.1. bladeRF

La placa que se utilizó para este proyecto fue la bladeRF x115 (de ahora en más, bladeRF solamente) desarrollada por Nuand [20], es un proyecto que sigue activo. Es una plataforma abierta, de costo medio (alrededor de 650 USD), con

un activo foro de consultas e intercambio, diseñada con el objetivo de permitirle a estudiantes, radioaficionados o profesionales del área de radio frecuencia explorar el mundo de las comunicaciones inalámbricas proporcionando una plataforma de desarrollo versátil. Con la bladeRF se puede recibir y transmitir señales de RF. La misma trabaja en el rango de 300 MHz a 3,8 GHz con un ancho de banda máximo de 28 MHz. Tiene un conversor AD de 12 *bits*. Cuenta con un FPGA 115KLE Altera Cyclone 4 E cuyos archivos fuente se encuentran disponibles en el repositorio Git de Nuand [22]. El esquemático de la bladeRF también se encuentra *online* en [19]. En la Figura 3.2 se presenta la placa de la bladeRF.

Esta se fabrica en dos modelos que varían únicamente en el FPGA: el modelo x115 que contiene un FPGA Cyclone 4 E de Altera EP4CE115 y el modelo x40 cuyo FPGA es de la misma familia pero el modelo EP4CE40. El primer modelo tiene un FPGA con mucho más recursos que el segundo [1] haciendo de esta versión una opción más económica.

Figura 3.2: Placa bladeRF [20].

Otro elemento a destacar de este *hardware* es que por muy bajo costo se puede adquirir una placa de expansión *GPIO expansion board* que permite conectarse a los pines del GPIO del microprocesador embebido en el FPGA. Para ello cuenta con LEDs e interruptores para generar entradas y ver salidas del GPIO. Esto puede ser útil para etapas de prueba y depuración.

En las próximas secciones se describirán tanto el esquemático de la placa como la arquitectura del FPGA que integra la misma. Esto se hará de manera general profundizando únicamente en los aspectos relevantes a la implementación llevada a cabo en el presente proyecto. Por más detalles se sugiere al lector referirse al Apéndice A para una descripción más detallada de la arquitectura del circuito y al Apéndice B de la arquitectura del FPGA.

Esquemático de la placa bladeRF

En la presente sección se estudiará sin entrar en detalles, el diagrama eléctrico de la bladeRF. El lector puede encontrar en el Apéndice A el esquemático completo provisto por Nuand.

En la Figura 3.3 se presenta un diagrama de bloques del esquemático en cuestión. Como se observa en la misma, se identifican cuatro grandes bloques:

- El bloque de adquisición y transmisión de datos compuesto principalmente por el *transceiver* multi-banda, multi-estandár LMS6002D de Lime Micro (LMS de aquí en adelante).
- El bloque de control y DSP integrado por un FPGA 115KLE Cyclone 4 E de Altera (EP4CE115). El sistema dentro del FPGA y sus funciones se describirán en la siguiente sección. Haciendo abuso de notación, sin intenciones de confundir al lector, al sistema dentro del FPGA se le referirá como Arquitectura del FPGA.
- El bloque de comunicaciones *hardware*-PC integrado por el Controlador USB *SuperSpeed* USB CYUSB301X de Cypress (FX3 de ahora en más).
- El bloque correspondiente a la generación del reloj compuesto por: un generador de reloj programable de cuatro salidas Si5338 de Silicon Labs, un oscilador de cristal de temperatura compensada de 400 MHz ASTX12 de Abracon Corporation y un conversor DAC161S055 de Texas Instruments, de 16 *bits* de alta precisión configurable, con una salida *buffereada* de voltaje.

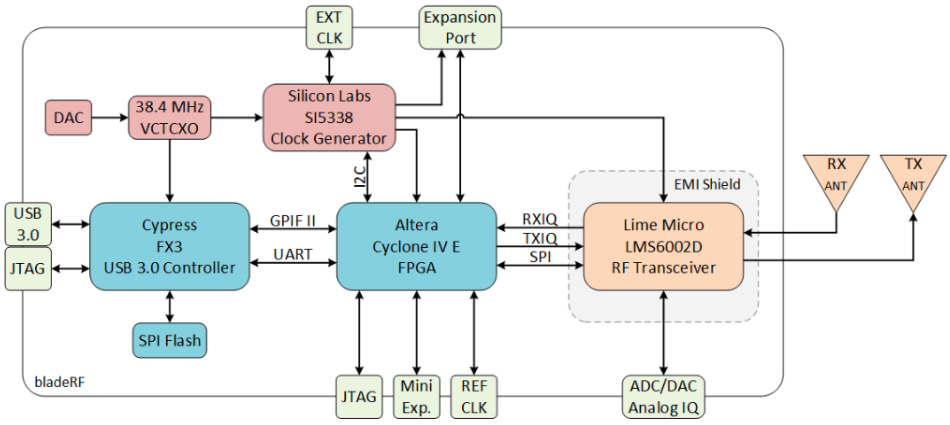


Figura 3.3: Diagrama de bloques del esquemático de la placa bladeRF[21].

Otro bloque básico de la bladeRF es el de alimentación. La placa puede ser alimentada por el USB o por una fuente de continua externa a la PC (cuenta con un conector hembra tipo *jack*). La fuente de alimentación se configura mediante el *jumper* 70. La fuente externa debe tener las siguientes características: 5 VDC con una corriente entre 1,5 y 2 A.

Como se adelantaba en la enumeración anterior, el integrado LMS es un *transceiver* multi-banda, multi-estandár, que trabaja en el rango de 0,3 – 3,8 GHz

adquiriendo o recibiendo muestras a una frecuencia de muestreo f_s configurable [1] que puede ser como máximo 40 MS/s, configurada mediante una interfaz SPI (*serial port interface*) por el FPGA. Las muestras se cuantizan con 12 *bits* con signo, en complemento a 2 (dando lugar a valores en el rango $[-2048, 2047]$). Mediante la interfaz SPI, el FPGA es capaz de realizar otras configuraciones, como ser el ancho de banda que puede tomar un valor de hasta 28 MHz (16 opciones discretas). Por detalles de la configuración del LMS a través del SPI referirse a [23].

En la Figura 3.4 se presenta un diagrama de bloques del LMS [24]. Para el camino de recepción de datos, en primer lugar hay tres entradas diferentes (y una alternativa que permite la realimentación interna de la señal *RF loopback*) con un amplificador dedicado LNA (*Low Noise Amplifier*). La señal adquirida por cada puerto es pre-acondicionada mediante una primer etapa de amplificación de bajo ruido (RXLNA) que es programable. La señal de radio frecuencia es entonces mezclada con el PLL (*Phase Locked Loop*) obteniendo a la salida la fase y la cuadratura de la misma (RXPLL) para ser convertida a banda base. La demodulación continúa con otra etapa de amplificación de ganancia variable (RXVGA1) previa al filtro pasa bajos programable (RXLPF). La señal IQ recibida es nuevamente amplificada por otro amplificador programable RXVGA2 para prevenir saturación y preservar el rango dinámico del conversor AD. Luego, cada parte de la señal es convertida al dominio digital en 12 *bits* en paralelo. En cada par de flancos de RX_CLK se entrega en primer lugar I y en el segundo Q. Las limitantes expresadas anteriormente en relación a las distintas configuraciones que puede tener el integrado determinan el funcionamiento de la placa bladeRF.

El *chip* puede ser usado en otras aplicaciones como ser para uso en *small cells* o equipos de comunicaciones inalámbricas de los estándares WCDMA/HSPA, LTE, GSM, CDMA2000 e IEEE802.16.

Los datos son entregados al FPGA junto con una señal de dato válido. Cada dos períodos de reloj se obtiene una muestra válida. En la siguiente sección se profundizará en la arquitectura del FPGA, sin embargo aquí se explicarán sus funciones y su razón de ser.

En primer lugar, al lector le podría surgir la siguiente pregunta: ¿Para qué es necesario el FPGA en la bladeRF? Como se explicará en breve, el FX3 tiene una interfaz GPIF II de 32 *bits* que trabaja a una frecuencia de 100 MHz. Por otro lado el LMS tiene una interfaz de 12 *bits* a una frecuencia arbitraria dada por la frecuencia de muestreo que puede llegar a los 80 MHz. Probablemente bajando la frecuencia del FX3 se podrían comunicar estos dos integrados. De lo contrario, sería necesario utilizar un sincronizador o *fifo* de algún tipo. También es necesario configurar el LMS, Si5338, y VCTCXO mediante los protocolos SPI o I^2C según corresponda. El FX3 probablemente pueda realizar estas tareas pero no necesariamente al mismo tiempo. Además, el FX3 bootea de una memoria SPI Flash (lo que podría llegar a complicar un poco las cosas) y habría que analizar si los pines de entrada-salida de propósito general del FX3 son suficientes para dicha

[1] Algunas placas están restringidas a valores discretos de frecuencia de muestreo, en ese caso es necesario remuestrear si se quiere utilizar una frecuencia de muestreo arbitraria para procesar la señal.

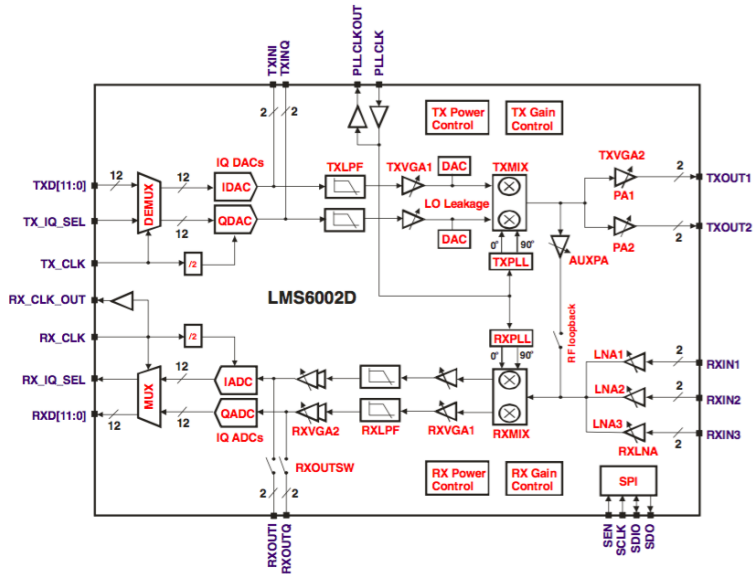


Figura 3.4: Diagrama de bloques del LMS6002D de Lime Micro.

aplicación.

Teniendo en cuenta estos aspectos, lo que el FPGA proporciona, es flexibilidad. Además de encargarse de todos los elementos mencionados anteriormente en una imagen FPGA por defecto llamada "hosted", le da la posibilidad al usuario de implementar procesamiento en un futuro. La imagen por defecto, para los datos es tan sólo un gran *fifo* y un generador de reloj. Sin embargo, desde el punto de vista de control, es responsable del bus de comunicaciones SPI con el DAC para sincronismo con el VCTCXO, del bus SPI con el LMS, del I^2C con el Si5338 para la generación de relojes, del control mediante la UART desde y hacia el FX3, de las comunicaciones con las placas de expansión, los periféricos y señales de referencia (por ejemplo el reloj externo).

Un ejemplo de procesamiento digital de señales (DSP) que viene implementado en la imagen por defecto, consiste en la señalización temporal de muestras mediante el uso de *timestamps* (se estudiará en secciones posteriores). Como es imposible garantizar bajo *jitter* (variabilidad en la cadencia del flujo de muestras) entre la bladeRF y el *software* corriendo en la PC, los *timestamps*, son usados entre el FPGA y la librería `libbladerf` para permitirle a las aplicaciones eliminar los efectos producidos por el *jitter* en el caso de recepción.

Otro aspecto a destacar en relación al FPGA es que cuenta con un puerto JTAG que permite comunicarse con la placa para cargar la imagen del FPGA y para verificación mediante la herramienta de Altera: *Signal Tap II Logic Analyzer*, analizador lógico en castellano. Para hacer uso de esta opción es necesario el cable USB Blaster. Mediante un bloque se configura un *trigger* o fuente de disparo y qué señales se desea adquirir. Otra de las configuraciones que se pude realizar es durante cuánto tiempo se adquieren las mismas. Por la naturaleza de la herramienta requiere cierto espacio de memoria por lo que para el diseño suele convertirse en

un compromiso que se deba tomar.

El siguiente paso en la cadena de comunicaciones de datos es el integrado FX3 encargado de las comunicaciones entre la PC y la placa bladeRF. El *chip* FX3 es un Controlador USB de alta velocidad (*SuperSpeed*).

El FX3 cuenta con una interfaz general programable llamada GPIF II [11] por su sigla en inglés, que funciona a 100 MHz y puede conectarse a cualquier procesador. En particular, puede conectarse al FPGA de la bladeRF. El protocolo GPIF II es una versión mejorada del protocolo GPIF. El mismo está basado en una máquina de estados programable que brinda la flexibilidad de implementar un estándar industrial o una interface propietaria. Puede funcionar como esclavo o como maestro. Proporciona una interfaz paralela de datos de 32 *bits* bidireccional. Cuenta con 13 señales de control que pueden ser configuradas como salida o entrada. Una máquina de estados programable se encarga de manejar todas estas señales y puede ser manejada desde un procesador externo.

Otra interfaz del FX3 es la de SPI utilizada en la bladeRF para cargar el firmware del FX3 desde una memoria Flash en la inicialización. Este último puede ser actualizado de la página web de Nuand. La última interfaz del integrado en cuestión que mencionaremos es la UART que se utiliza para comunicaciones de configuración desde el FPGA así como también para realizar chequeos de estado. Por esta interfaz es que se carga la imagen del FPGA.

El FX3 tiene integradas las capas físicas tanto para USB 3.1 generación 1 como el USB 2.0. Además cuenta con un procesador de 32 *bits* ARM926EJ-S para procesamiento de datos y para diseño de aplicaciones particulares. En la Figura 3.5 se puede observar un diagrama de bloques del integrado en cuestión.

Como se adelantaba, la placa cuenta con un oscilador VCTCXO (*Voltage Controlled Temperature Compensated Crystal Oscillator*) de 38,4 MHz, controlado por un DAC. El mismo es el reloj de entrada del FX3 y es utilizado como entrada de referencia del *chip* Si5338 el cual genera otros relojes a utilizar. El Si5338 se programa desde una interfaz I^2C dando la posibilidad de generar otros cuatro relojes. Entre ellos se encuentran los que se utilizan para muestreo de transmisión y recepción del LMS, el que se utiliza en la parte de control del FPGA y uno para el XB. Por otra parte el Si5338 puede recibir una entrada de reloj desde el exterior de la placa.

Arquitectura del FPGA de la bladeRF

En esta sección se describirá de manera general la arquitectura del FPGA 115KLE Altera Cyclone 4 E que es parte de la placa bladeRF. Teniendo en cuenta la aplicación en cuestión, se estudió de manera superficial el comportamiento del FPGA en su completitud y se profundizó en el camino de recepción de datos. Con el mismo criterio se hará una exposición en la presente sección del funcionamiento genérico del FPGA.

Desde el punto de vista funcional, el FPGA está encargado de realizar cinco grandes funciones: tres de configuración (del integrado LMS, FX3 y los correspondientes al sistema de reloj), una de recepción de datos y otra de transmisión de datos. Para las tres primeras, el FPGA cuenta con una microprocesador integrado llamado `nios_system` que cuenta con diversos periféricos específicos para comuni-

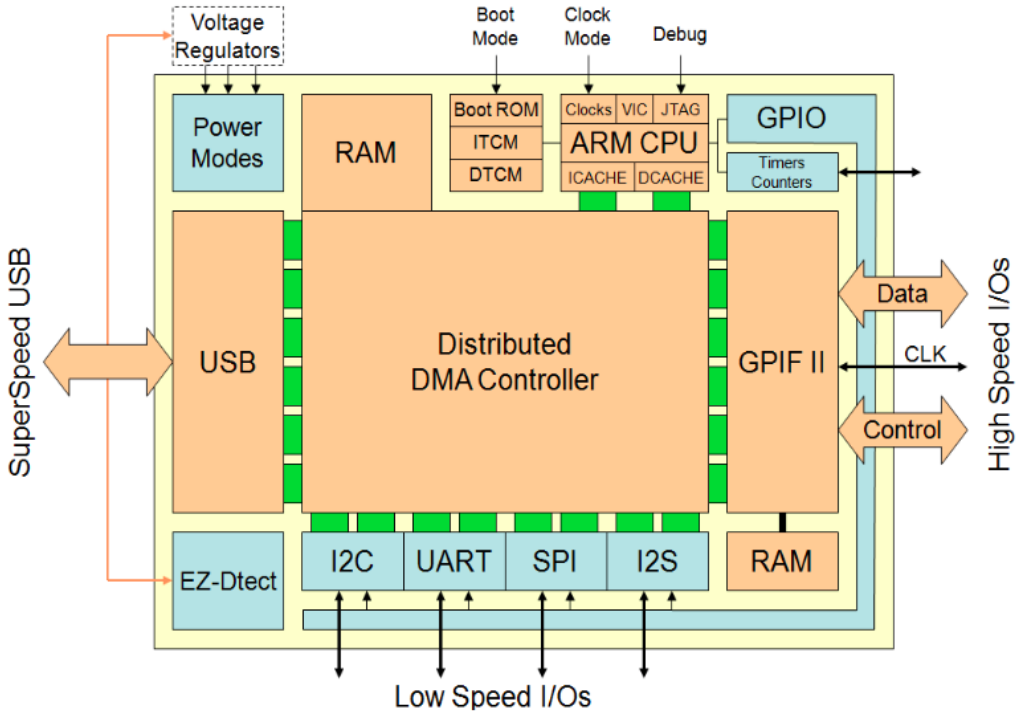


Figura 3.5: Diagrama lógico del integrado CYUSB301X de Cypres (FX3) [12].

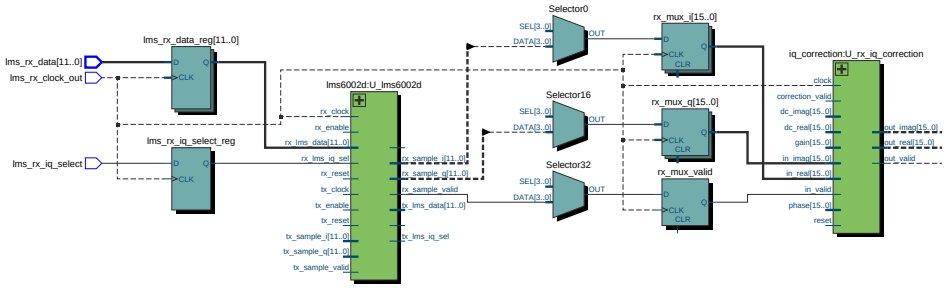
carse con cada *chip* por separado. Para el caso del LMS, por ejemplo, implementa una serie de funciones en el lenguaje *C* que son utilizadas para la configuración y manejo de datos respetando el protocolo SPI definido en [23].

En cuanto a la recepción de datos (ver Figura 3.6), los mismos son entregados por el LMS al bloque `lms_6002d`. Luego, el bloque `lms_6002d` genera cada 2 flancos de reloj, el par I y Q válido y a la salida del mismo se extiende la representación de cada componente a 16 *bits*. De forma alternativa, los datos pueden provenir de otras fuentes (ver Sección B.4). Una es desde un bloque que genera un contador y en la otra se realiza una realimentación con datos que correspondientes al camino de transmisión (*Digital Loopback*). El procesamiento continúa con el bloque `rx_iq_correction` que se encarga de realizar una corrección relativa al desbalance IQ y a continuación, el bloque `fifo_writer`, de armar el contenido de paquetes de muestras y escribirlos en los *fifos* (memorias del tipo *first in first out*) hasta que el bloque `fx3_gpif` re-arma los mismos y los envía hacia el *chip* FX3.

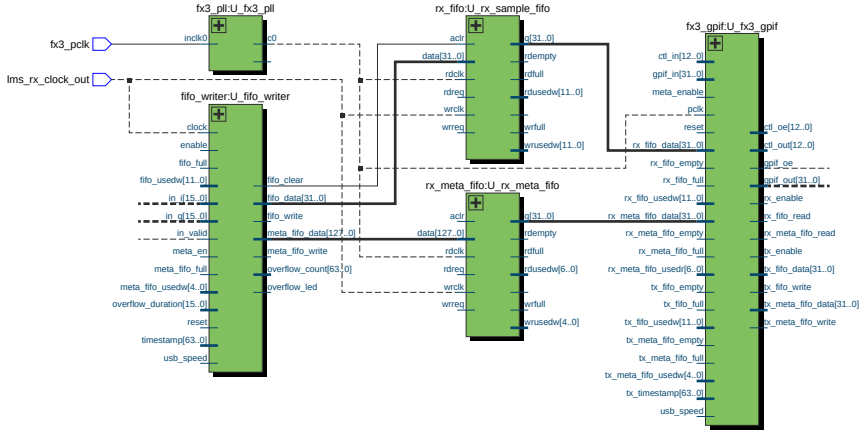
En este sentido, se estudió dónde sería posible incorporar bloques de procesamiento: entre los bloques `rx_iq_correction` y `fifo_writer`. Esto se definió de esta forma porque en el primero se realiza una corrección de las muestras que debe ser previa a cualquier tipo de procesamiento y el `fifo_writer` prepara los datos para su entrega al bloque que se comunica con el FX3.

El camino de transmisión de datos es muy similar al de recepción y se utilizan bloques análogos, el diagrama del mismo se encuentra en la Sección B.2.

Por último se estudió el uso de recursos con la arquitectura sin modificar,



(a) Bloques: `lms_6002d` y `rx_iq_correction`.



(b) Bloques: `fifo_writer`, `rx_sample_fifo`, `rx_meta_fifo` y `fx3_gpif`.

Figura 3.6: Bloques pertenecientes al flujo de datos para el camino de recepción en el FPGA para el sistema actual.

esto será determinante para la definición de la arquitectura a implementar. En la Tabla 3.1 se presentan los recursos más importantes para esta aplicación y su uso sin realizar modificaciones al FPGA.

Elementos lógicos	8886/114.480 (8 %)
M9Ks	57/432 (13 %)
Multiplicadores	12/532 (2 %)
Bits de Memoria	376.832/3.981.312 (9 %)
Bits de Implementación de Memoria	525.312/3.981.312 (13 %)

Tabla 3.1: Recursos utilizados por el FPGA sin modificar en la bladeRF.

3.2. gr-isdbt

`Gr-isdbt` [9] es la primer implementación en código abierto de un receptor *full-seg* de Televisión Digital del estándar ISDB-T (presentado en la Sección 2.1) en GNU Radio. Fue específicamente diseñado con el objetivo de poder realizar diferentes mediciones a lo largo de toda la cadena de procesamiento. El proyecto fue desarrollado en el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería de la UDELAR por el grupo ARTES [8].

El sistema actual, completo de recepción `gr-isdbt` visualizado en el entorno GRC se presenta en la Figura 3.7.

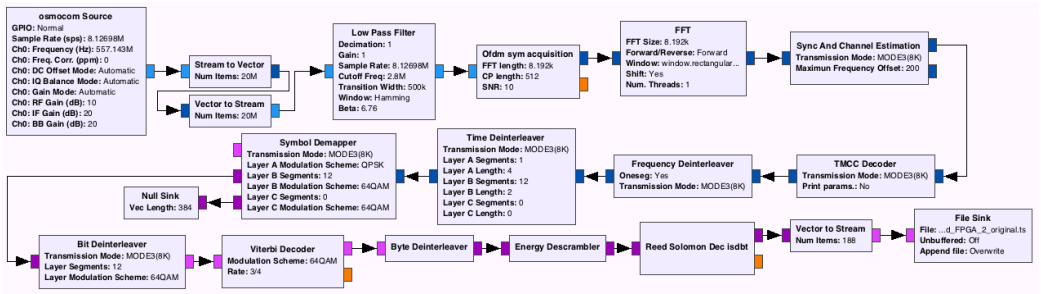


Figura 3.7: *Flowgraph* de GNU Radio Companion del receptor `gr-isdbt`.

El módulo `gr-isdbt` [18] utiliza algunos bloques de propósito general: el `Low Pass Filter` (filtro pasa bajos), la `FFT` y los `Stream to Vector` y `Vector to Stream` que sirven para pasar de un flujo de muestras a un flujo de vectores de muestras. Se hará una presentación general del receptor.

En primer lugar se adquieren los datos del *hardware* a través del `Osmocom Source` que se comunica en particular con la bladeRF (hay disponibles otros bloques fuente para comunicarse con otras placas). Como se observa en la Figura 3.7 este se configura con una frecuencia de muestreo $f_s = 512/63$ MHz $\approx 8{,}13$ MHz como se mencionó en la Sección 2.1 (el *hardware* podría utilizar otra frecuencia de muestreo y en dicho caso habría de remuestrear sobre GNU Radio) y una frecuencia central f_c de 557,143 MHz que corresponde al canal La Tele (Tabla 2.2). Se agrega el filtro pasa bajos con una frecuencia de corte de 2,8 MHz para eliminar posibles ruidos de canales adyacentes.

El próximo paso en el procesamiento consiste en la sincronización OFDM. Esto implica la detección de cada símbolo, la eliminación del prefijo cíclico, la corrección fraccional y entera en frecuencia y por último la corrección por no idealidades del canal. Esto es realizado en dos etapas en los bloques: `OFDM Sym Acquisition` y `Sync And Channel Estimation`. El primero, está a cargo de la sincronización temporal del símbolo y de la corrección en frecuencia pre-FFT. El procesamiento de este bloque es bastante similar para el estándar DVB-T, el módulo provee de la versión correspondiente para DVB-T. Dicha versión fue incorporada a la librería oficial de GNU Radio.

Una vez hechos estos ajustes, como en todo sistema OFDM, se calcula la FFT. El bloque `Sync And Channel Estimation` se encarga de realizar el procesamiento

post-FFT que consiste en la corrección entera en frecuencia de las portadoras, la estimación del canal y la ecualización correspondiente. Referirse a Apéndice D por una explicación más detallada de este bloque. En la Figura 3.8 se puede observar la constelación de los símbolos complejos luego de haber sido procesados por los tres bloques mencionados. En la misma se pueden observar todos las portadoras de los segmentos del símbolo OFDM, la capa 1-seg corresponde a una modulación QPSK y la capa full-seg a una modulación 64QAM, las portadoras TMCC corresponden a una modulación BPSK.

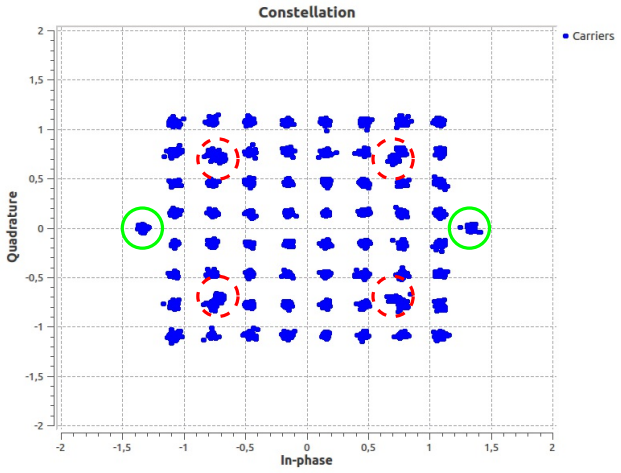


Figura 3.8: Un ejemplo de una constelación para algunos símbolos recibidos. Se observan dos capas: en línea punteada, la modulación QPSK correspondiente a 1-seg , y sin marcar la 64QAM correspondiente a full-seg, los pilotos también se pueden apreciar (modulados en DBPSK/BPSK) marcados con línea continua.

La última etapa de la sincronización OFDM la lleva a cabo el bloque `TMCC Decoder`. Este se encarga de detectar la ubicación de cada trama OFDM y de realizar la lectura de los mensajes TMCC, por más información sobre los TMCC, se invita al lector a repasar la Sección 2.1. Mediante la información se realiza la decodificación del canal y el *symbol demapping*. Con esta se configuraran adecuadamente los siguientes bloques en la cadena del procesamiento: `Symbol Demapper` y `Viterbi Decoder`.

Los bloques posteriores al `TMCC Decoder` son responsables de la decodificación de la señal. Por último, el bloque `File Sink` es utilizado para guardar en disco la salida del procesamiento.

3.3. `gr-osmosdr`

`Gr-osmosdr` es un módulo desarrollado en GNU Radio cuyo objetivo principal es implementar un sólo juego de bloques *source* y *sink* que son capaces de comunicarse con la mayoría de los SDRs en el mercado actual (ver Sección 2.2.2). El *source* llamado `Osmocom Source` se comunica con los SDRs para adquirir muestras

desde el *hardware* y el *sink* (`Osmocom Sink`) para enviarlas. Este módulo es estudiado debido a que se utilizó para comunicarse con la placa bladeRF desde GNU Radio. En la Figura 3.9 se puede observar el bloque `Osmocom Source` y su cuadro de diálogo de propiedades. Entre las configuraciones más generales se encuentra la frecuencia de muestreo, el ancho de banda y la ganancia.

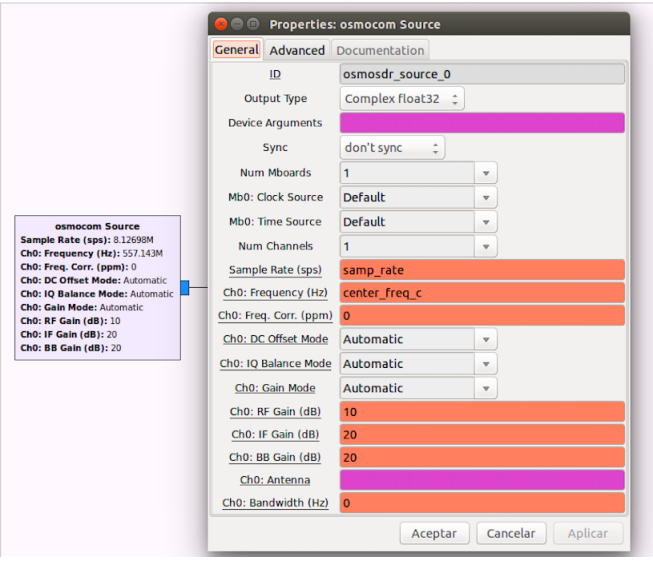


Figura 3.9: Bloque `Osmocom Source` en GNU Radio Companion, diálogo de propiedades.

La comunicación con la bladeRF se hace a través de la librería `libbladerf`. La librería permite entre otras funciones: abrir y cerrar el dispositivo y solicitar información del mismo, sintonizarlo a varias frecuencias, configurar la frecuencia de muestreo, configurar el ancho de banda y la ganancia de la cadena de RF, transmitir y recibir muestras complejas en banda base, actualizar la imagen del FPGA y realizar pruebas en la placa de bajo nivel de acceso. Algunas funciones son transparentes para el usuario y otras se pueden configurar mediante argumentos escritos en formato *string* (mediante *Device Arguments* en la Figura 3.9). Las configuraciones relacionadas con RF están fuertemente ligadas con los puntos vistos en la Sección 3.1 donde se describe el *transceiver* RF y sus configuraciones.

Los datos entregados por el bloque `Osmocom Source` se encuentran en el formato `gr_complex` que es una definición del programa para representar a los números complejos en un par de números representados en punto flotante. Es importante tener presente que los datos de la salida se encuentran escalados por un factor de 1/2048 normalizándolos al intervalo [-1,1].

3.4. Interfaz Solución Actual

El sistema actual, envía datos entre la interfaz de radio y la PC mediante paquetes denominados mensajes, respetando el orden de las muestras. El tamaño de los mensajes es de 2048 *bytes* (para el caso USB 3.0, requerido por el estándar

debido a la frecuencia de muestreo de la aplicación). Por lo tanto, los mensajes contienen 512 muestras (16 *bits* para I y 16 *bits* para Q por cada muestra). Los mensajes están compuestos por muestras que pueden corresponder a datos o a metadatos, los cuales corresponden a información extra referida a cierto conjunto de datos. Como se explicará en la sección de Uso de Metadatos, los metadatos están compuestos por *flags*, *bytes* reservados y *timestamps*.

Como recordará el lector de la sección anterior, los datos son provistos por el integrado LMS (*RF transceiver*) hacia el FPGA a la frecuencia de muestreo $2 * f_s$. Las muestras se cuantizan en el *chip* LMS con 12 *bits* con signo, en complemento a 2 (dando lugar a valores en el rango $[-2048, 2047]$). Luego dentro del FPGA, el bloque `lms_6002d` genera cada 2 flancos de reloj, el par I y Q válido y a la salida del mismo se extiende la representación de cada componente a 16 *bits*. Luego, el bloque `rx_iq_correction` se encarga de realizar una corrección relativa al desbalance IQ y a continuación, el bloque `fifo_writer`, de armar el contenido de los mensajes y escribirlos en los *buffers* hasta que el bloque `fx3_gpif` re-arma los mismos y los envía hacia el *chip* FX3. Por último, entre el integrado FX3 y el *host* sobre la PC, el contenido de los mensajes es inalterado (en particular los metadatos) y puede ser leído tal cual fue generado en el FPGA. En el sistema actual, los mensajes están compuestos únicamente por muestras (son entonces 512).

Del lado del *software*, la librería `libbladeRF` se encarga de manejar la comunicación entre la placa bladeRF y la PC mediante el USB. El módulo `gr-osmosdr` hace uso de la misma para la lectura de los datos en GNU Radio.

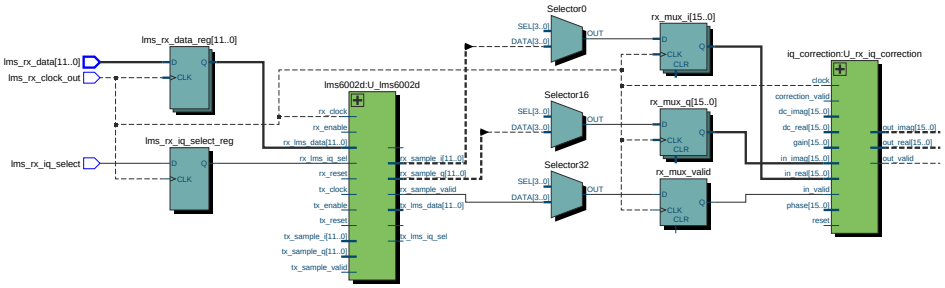
El sistema actual cuenta con la posibilidad del envío de muestras a la PC con información extra referida a cierto conjunto de datos conocida como metadato.

3.4.1. Flujo de datos sobre el *Hardware*

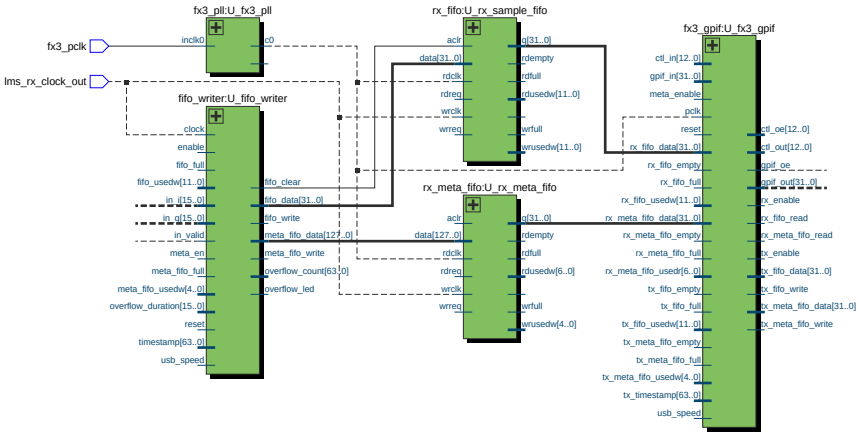
A continuación se hará una profundización en el funcionamiento de los bloques del FPGA que generan el contenido de los mensajes que contienen datos y metadatos (siempre y cuando se encuentren activados), para el caso de recepción (Rx). Los bloques a estudiar son los siguientes:

- `lms_6002d`
- `rx_iq_correction`
- `fifo_writer`
- `rx_sample_fifo`
- `rx_meta_fifo`
- `fx3_gpif`

En la Figura 3.10 se observa la interconexión de los mismos.



(a) Bloques `lms_6002d` y `rx_iq_correction`.



(b) Bloques `fifo_writer`, `rx_sample_fifo`, `rx_meta_fifo` y `fx3_gpif`.

Figura 3.10: Bloques pertenecientes al flujo de datos para el camino de recepción en el FPGA para el sistema actual.

`lms_6002d` El bloque `lms_6002d` se encarga de recibir las muestras I y Q provenientes del integrado LMS por los pines `lms_rx_data[11..0]` del FPGA mediante la señal de reloj `lms_rx_clock_out` a la frecuencia de muestreo $2 * f_s$. En cada flanco, se alterna entre la muestra en fase I y en cuadratura Q. En este sentido, el bloque genera cada 2 flancos de reloj, el par I y Q válido indicado por la señal de habilitación `rx_sample_valid`. La salida del `lms_6002d` es extendida a 16 *bits*. En el caso de transmisión, el bloque envía las muestras hacia el *chip* LMS de forma recíproca al caso de recepción.

`rx_iq_correction` El bloque `rx_iq_correction` cumple la función de realizar la corrección del desbalance IQ. Para ello, utiliza los parámetros de ganancia y fase indicados mediante las entradas `gain` y `phase` respectivamente, los cuales son provistos por el bloque `nios_system`. Estos parámetros se los aplica a cada

componente de la señal de forma de compensar el desbalance[2]. El bloque también puede realizar un ajuste en el desbalance de continua (DC *offset*), por defecto está configurado para no realizarlo ya que dicho ajuste se realiza en el *chip* LMS. En definitiva, ninguna de las dos funciones de corrección implementas en el `rx_iq_correction` están activas por defecto por lo que las muestras al pasar por este bloque permanecen inalteradas. Eventualmente la corrección de ganancia y fase podría efectuarse si se activa desde la PC.

`fifo_writer` El bloque `fifo_writer` se encarga de armar el contenido de los mensajes y escribirlos en bloques de memoria tipo *fifo*. Si el uso de metadatos está activado, el mismo arma el contenido de metadatos y los escribe en el bloque de memoria `rx_meta_fifo`. Los datos se escriben en el bloque de memoria `rx_sample_fifo` siempre que son válidos. Este último, funciona con el reloj de muestreo `lms_rx_clock_out`. El contenido correspondiente a los *flags* y a los *bytes* reservados (metadatos) está fijo en valores por defecto. Los *timestamps* ingresan por el puerto de entrada llamado `timestamp`. Estos son generados por el bloque `nios_system` y permiten enumerar las muestras. El bloque puede *resetear* el contenido del *fifo* de datos, ya que maneja su señal de *reset*. En la Sección B.1 se explican detalles de su funcionamiento debido a que el mismo fue modificado para implementar la señalización de vectores. En la Sección 4.2.1 se explican las modificaciones realizadas.

`rx_sample_fifo` El bloque `rx_sample_fifo` es una memoria de tipo *fifo*. Cumple la función de *buffer* para las muestras. Es escrito con datos por el bloque `fifo_writer` de forma síncrona con el reloj de muestreo. Los datos son leídos por el bloque `fx3_gpif` de forma síncrona con el reloj asociado a la interfaz GPIF entre el FPGA y el integrado FX3.

`rx_meta_fifo` El bloque `rx_meta_fifo` es análogo al anterior con la diferencia de que almacena metadatos en lugar de datos. Esto se lleva a cabo una vez por cada mensaje.

`fx3_gpif` El bloque `fx3_gpif` es el responsable de manejar la interfaz GPIF con el controlador USB FX3, la cual se utiliza para el flujo de datos. El contenido de los mensajes atraviesa la interfaz por un bus de 32 *bits* de ancho (`gpif_out[31..0]`), de forma síncrona con el reloj `fx3_pclk` de 100 MHz utilizado por la interfaz. Como se especifica en la página web de Nuand [20] el FX3 soporta perfectamente la máxima frecuencia de muestreo permitida (40 MS/s) por el LMS. Considerando que cada muestra tiene 32 *bits* la cadencia de datos es para este caso 1,28 Gbps menor a 5 Gpbs para el caso *full-duplex* del USB 3.0. Para frecuencias muy lentas y su consecuencia en el sistema referirse a la Sección 3.3.

El FX3 también se encarga de manejar los datos y metadatos tanto para la recepción como para la transmisión de muestras. En el caso de recepción, el mismo

[2] A pesar de que el sistema cuenta con esta función, la misma esta deshabilitada desde la PC haciendo que las muestras queden inalteradas.

arma el contenido de los mensajes, mediante los datos y metadatos (en caso de estar activo el uso de los mismos) escritos en los *buffers*.

3.4.2. Flujo de datos sobre el *Software*

Como se mencionó en la Sección 3.3, la librería `libbladeRF` se encarga de manejar la comunicación entre la placa bladeRF y la PC mediante el USB. En el caso de recepción, el bloque `Osmocom Source` lee el contenido de los mensajes y los convierte al formato `gr_complex` utilizado por GNU Radio. En particular, el contenido es normalizado al intervalo $[-1, 1]$, mediante un factor de 2048. El bloque genera a su salida muestras independientes distribuidas en cada mensaje recibido desde la placa. Los mensajes recibidos se pueden rearmar si se coloca a continuación del bloque `Osmocom Source` un bloque `Stream to Vector` configurado con tamaño de vector igual al largo de los mensajes recibidos.

3.4.3. Uso de Metadatos

Los metadatos corresponden a información extra referida a cierto conjunto de datos. Una opción que permite la arquitectura original de la placa bladeRF es activar el uso de metadatos para señalizar los mensajes, función que por defecto se encuentra desactivada. Para activar el uso de metadatos es necesario asignarle el valor lógico 1 a la señal `meta_en`, que por defecto, esto puede hacerse desde la PC a través de el `gr-osmosdr`. La señal `meta_en` se encuentra conectada a un *bit* del bus GPIO del `nios_system` (ver Sección B.3).

El formato de los mensajes con metadatos está definido según la Tabla 3.2. El tamaño de los metadatos es de 16 *bytes* ocupando el lugar correspondiente a 4 muestras (índices 0 a 3). Debido a que un mensaje está compuesto por 2048 *bytes*, con los metadatos activados, cada uno tiene 508 muestras de datos (índices 4 a 511). El contenido de los metadatos está definido por el bloque `fifo_writer`. Se utilizan 4 *bytes* de *flags*, 8 *bytes* de *timestamps* y 4 *bytes* reservados. Los *bytes* reservados y los *flags* contienen por defecto una constante mientras que los *timestamps* indican el número de muestra correspondiente a la primer muestra del mensaje. En este sentido, los mismos se incrementan un valor de 508 entre mensajes consecutivos.

Si bien la librería de *software* `libbladeRF` contiene estructuras para manejar información de metadatos, los metadatos recibidos son ignorados como tales los mismos son inalterados y son leídos como cualquier muestra del mensaje. Por lo tanto, se puede alterar el contenido de los metadatos generados en el FPGA con determinado criterio y luego interpretar el contenido en el *software* como cualquier otra muestra.

3.5. Conclusiones

El capítulo presentó diversos aspectos claves de la solución actual que serán importante conocer en mayor o menor profundidad para comprender y llegar a la solución implementada.

	Índice (Muestras)	**Valor Q**	**Valor I**	Representante
Metadatos	1	**0x1234**	**0x4321**	*Flags*
	2	**0x0000**	**0x0000**	*Timestamp low*
	3	**0x0000**	**0x0000**	*Timestamp high*
	4	**0xFFFF**	**0xFFFF**	Reservados
Datos	5	$\boldsymbol{X_{q0[15..0]}}$	$\boldsymbol{X_{i0[15..0]}}$	Muestra 0
	6	$\boldsymbol{X_{q1[15..0]}}$	$\boldsymbol{X_{i1[15..0]}}$	Muestra 1
	⋮	⋮	⋮	⋮
	⋮	⋮	⋮	⋮
	511	$\boldsymbol{X_{q506[15..0]}}$	$\boldsymbol{X_{i506[15..0]}}$	Muestra 506
	512	$\boldsymbol{X_{q507[15..0]}}$	$\boldsymbol{X_{i507[15..0]}}$	Muestra 507

Tabla 3.2: Contenido de mensaje con metadatos. Las primeras cuatro muestras corresponden a metadatos, las restantes a datos. Las mismas están compuestas por 4 *bytes*: 2 correspondientes a I y 2 a Q.

Se comenzó en el *hardware* del sistema actual, placa bladeRF x115 de Nuand. Se estudió la arquitectura de la misma y en particular del FPGA ya que dentro de este se realizará el diseño.

Se estudió el módulo `gr-isdbt` de GNU Radio que implementa un receptor *full-seg* de Televisión Digital del estándar ISDB-T implementado en la Facultad de Ingeniería de la Universidad de la República del Uruguay. Se presentaron los bloques del mismo. Se destaca, en particular, el primero cuya función es la de sincronización ya que será el que se implemente en *hardware* y se tomará como referencia.

En GNU Radio también se vio el módulo `gr-osmosdr` el cual hace de interfaz entre el programa y la bladeRF.

Se estudió la interfaz del sistema actual entre la placa bladeRF y la plataforma GNU Radio. Se estudió el uso de metadatos sobre los paquetes que envía la placa hacia la PC.

Capítulo 4

Solución Propuesta - Comunicaciones

En este capítulo se presentan las modificaciones realizadas sobre las comunicaciones sobre la interfaz de datos entre la placa bladeRF y la plataforma GNU Radio presentada en el capítulo anterior para el envío de señales vectoriales. En primer lugar se definen los requerimientos para la aplicación y luego se detallan las modificaciones realizadas, puntualizando los criterios elegidos, los cambios realizados y los resultados obtenidos así como el alcance de la solución.

Los bloques de procesamiento que se desarrollaron en el FPGA generan a su salida señales vectoriales, como por ejemplo el `Ofdm Sym acquisition` de la Figura 3.1, a diferencia del sistema original donde la señal que se envía del *hardware* hacia la PC en *streaming* de muestras independientes. Por lo tanto, resultó necesario modificar la forma en la que el FPGA envía los datos para que sea posible obtener en GNU Radio los vectores generados por los bloques de procesamiento en el FPGA. Para lograr esto, se estudió la interfaz original (ver Capítulo 3)y se analizaron posibles alternativas. Se definió un protocolo para señalizar vectores mediante el uso de metadatos sobre paquetes. Para activar el uso de metadatos se modificó el módulo `gr-osmosdr`. Finalmente, se implementó un bloque en GNU Radio, de forma que fuera posible rearmar los vectores y continuar con el procesamiento de forma transparente. En otras palabras, de forma que los bloques aguas abajo en la cadena de procesamiento no distinguieran que los datos ya vienen procesados del *hardware*.

La solución planteada implicó realizar cambios en el FPGA, la modificación del módulo `gr-osmosdr` de GNU Radio y la implementación de un bloque sobre GNU Radio.

4.1. Protocolo para señalización de vectores

Teniendo en cuenta la posibilidad de enviar metadatos sobre mensajes (ver Sección 3.4.3), se definió un protocolo para señalizar vectores mediante la generación de *flags* (banderas para señalización de muestras) en el bloque `fifo_writer`. Los *flags* se denominaron `n` y `m` los cuales, de acuerdo al par de valores que toman, indican comienzo, continuidad o fin de vectores sobre cada mensaje. Los *flags* `m`

utilizan la parte baja de los *flags* de metadatos y los *flags* `n` la parte alta. El lector no deberá confundir el abuso de notación de los *flags*.

El valor de `n` indica qué muestra es la primera en el caso de que el mensaje contenga el comienzo de un vector. El valor de `m` indica, si hay un vector en ese mensaje que termina, en qué muestra está el último dato. La primer muestra del mensaje se identifica con el 1 y la última con el 512. De esta forma `n`, `m` $\in [5, 512]$, para comienzo o fin de vectores válidos. Los valores de $m = n = 1$ indican que los vectores trascienden el contenido del mensaje.

En la Tabla 4.1 se define el criterio para los posibles valores de `n` y `m`.

Comienzo	Fin	Significado
`n`	`m`	
XX	1	Comienza un nuevo vector que no termina en ese mensaje.
XX	YY	Termina el vector actual (en YY) y empieza uno nuevo (en XX).
1	1	Continua el vector actual.
1	YY	Termina el vector actual (en YY).
0	0	Mensaje inválido.

Tabla 4.1: Criterio de señalización de vectores.

Por lo tanto, para implementar la señalización es necesario generar los valores correctos de `n` y `m` en el bloque `fifo_writer` del FPGA para cada vector a enviar y sobre el *software* interpretar los mismos para re-armar los vectores con las muestras de mensajes. Esto último debe ser transparente para el resto de la cadena de procesamiento sobre la plataforma GNU Radio. A continuación se presentan las modificaciones realizadas sobre el *hardware* y el *software* para la implementación del protocolo para señalizar vectores con el criterio definido en la Tabla 4.1.

4.2. Modificaciones e implementación sobre el *Hardware*

4.2.1. Modificación en el bloque `fifo_writer`

Para la generación de los *flags* `n` y `m` se modificó el bloque `fifo_writer`. Se implementó una máquina de estados que calcula los valores de `n` y `m` a escribir en los *flags* de metadatos. El resultado del cálculo será el definido en la Tabla 4.1, en otras palabras, la máquina de estados no tiene los valores almacenados en una tabla sino que los calcula. También se agregaron señales auxiliares para poder controlar la máquina de estados.

Se consideraron las siguientes hipótesis y condiciones para la implementación de la señalización sobre el bloque:

- El bloque funciona con el mismo reloj original, `lms_rx_clock_out`, el cual es el reloj de muestreo.
- Se debe utilizar un puerto USB 3.0 debido a la frecuencia de muestreo de la aplicación, la señal *usb_speed* con valor *usb_speed* = '0' indica este caso.

Por lo tanto el tamaño de mensajes es de 512 muestras y los metadatos son generados para dicho tamaño.

- Los vectores que llegan al bloque tienen un largo fijo denominado `symbol_length` y las muestras pertenecientes a los distintos vectores están indicadas por la señal de habilitación `in_valid`. También deben estar activas las señales `enable` y `meta_en` (habilitación de metadatos).

- Los valores de `n` y `m` serán los de la Tabla 4.2 para el caso de `symbol_length`= 8192. En este caso serán necesarios 17 o 18 mensajes para completar un vector.

Mensaje	n	m	
0	**5**	**1**	Símbolo 1
1	**1**	**1**	
⋮	⋮	⋮	
⋮	⋮	⋮	
16	**69**	**68**	
17	**1**	**1**	Símbolo 2
⋮	⋮	⋮	
⋮	⋮	⋮	
33	**133**	**132**	
34	**1**	**1**	Símbolo 3
⋮	⋮	⋮	
etc	**etc**	**etc**	

Tabla 4.2: Primeros valores de `n` y `m` del protocolo de señalización de vectores.

De la tabla anterior se desprende que la primer muestra recibida en el `fifo_writer` corresponde a la primer muestra del primer símbolo o vector a enviar a enviar a la PC. Las siguientes 8191 muestras corresponden al mismo símbolo. Se necesitarán 17 mensajes para enviar todos los datos del primer símbolo. Los metadatos del primer mensaje indicarán que la primer muestra se encuentra en el lugar 5, los de los siguientes 15 mensajes ($n = m = 1$) indican que continúan los datos del símbolo que empezó y los metadatos del último mensaje ($n = 69, m = 68$) indican que la última muestra del vector que se estaba enviando se encuentra en el lugar 68 y que comienza uno nuevo en el lugar 69.

- Cada vez que se complete un vector se generarán en los *flags* los valores correspondientes de `n` y `m` según los valores escritos anteriormente. De esta forma los vectores quedarán de forma consecutiva a lo largo de las muestras de los mensajes utilizando todas las muestras disponibles de cada mensaje.

Una alternativa podría haber sido utilizar un número entero de mensajes para alojar cada vector. En ese caso, tendría la ventaja de hacer más sencilla la señalización pero las desventaja de que se haría un uso ineficiente de la interfaz entre la placa y la PC.

- El *host* admite pausas del orden de un par de segundos sin recibir datos de la placa. Por lo tanto, el bloque también puede admitir pausas del mismo orden en la generación de vectores. Por más información de este punto referirse a la Sección 3.2.
- El bloque debe ser *reseteado* al comienzo de cada captura y en caso de haber algún vector incompleto. Esto último sucede si el vector entregado en la entrada del `fifo_writer` queda incompleto y por lo tanto se debe borrar el contenido de los *fifos* para evitar el envío de datos erróneos.

La implementación de la máquina de estados para generar los valores de `n` y `m` podría haberse realizado en un bloque independiente. La misma se realizó sobre el bloque `fifo_writer` debido a que se reutilizó la mayoría de los procesos y señales de control del bloque.

4.2.1.1. Implementación de señalización

A continuación se presenta el desarrollo de las modificaciones sobre el bloque `fifo_writer` para la generación de los *flags* `n` y `m` mediante una máquina de estados.

Se implementó una máquina de estados en la cual se define el valor de `n` y `m` de acuerdo a la Tabla 4.2 para cada mensaje. En la Figura 4.1 se presenta un diagrama de estados y las condiciones de transición.

A continuación se explican algunos detalles sobre los estados (ver Figura 4.1 y Figura 4.2):

st_idle:	Primer estado luego de un *reset*. Inicializa los valores de señales auxiliares como los contadores de muestras y mensajes. La salida de este estado corresponde al momento de escritura de los metadatos en el *fifo* de metadatos.
st_first:	Se asignan los primeros valores de `n` y `m` del vector. Además se guarda en la señal `nl` el valor de n. La salida de este estado es incondicional.
st_calc_0:	Se calcula la cantidad de muestras enviadas en el primer mensaje del vector actual. Al igual que en el estado anterior, la salida de este es incondicional.
st_calc_1:	Estado para calcular los mensajes necesarios para completar el vector dadas las primeras muestras ya enviadas y el largo total del vector. La salida de este estado se da al finalizar el cálculo de la cantidad de mensajes necesarios para el envío completo del vector.

st_calc_fin: Se determinan los valores de `n` y `m` para el próximo vector. Los mismos se guardan en señales auxiliares. Esto se hace en función de los resultados del estado **st_calc_1**. Además se reinicia el contador de muestras y se define si se está en un caso borde (donde se reinicia la tabla de `n` y `m`) o no. La salida de este estado es incondicional.

st_cont: Estado para asignar los valores `n` y `m`. Estos dependen de en qué ubicación relativa al vector se encuentre la iteración. En el caso de los mensajes intermedios, toman los valores $1, 1$ y en el caso del último mensaje, toman los valores calculados en **st_calc_1** (ver Tabla 4.2). La salida de este estado sólo se da cuando se completó hasta el penúltimo envío de los mensajes del vector.

st_last: Se reinicia el contador de mensajes. La salida de este estado es incondicional.

st_downcount: Estado para esperar el comienzo del próximo mensaje. Además en este estado se escriben en el puerto de salida los valores de `n` y `m` para el próximo mensaje. En caso de que la máquina de estados se encuentre en el caso borde mencionado el próximo estado no es el **st_calc_0** sino, el **st_first**. La salida de este estado se da al finalizar el envío del último mensaje del vector. El próximo estado dependerá de una señal que indica un caso borde.

En la Figura 4.2 se muestran los valores de `n` y `m` y de las señales auxiliares para cada estado.

Los valores de las siguientes señales serán los que produzcan las transiciones:

```
signal dma_downcount   : signed(12 downto 0);
signal message_count_c : unsigned (7 downto 0);
signal message_count   : unsigned (7 downto 0);
signal sample_count    : unsigned (13 downto 0);
signal sample_count_c  : unsigned (15 downto 0);
signal flag_odd        : std_logic;
```

La señal `dma_downcount` indica la posición actual en términos de muestras respecto al mensaje que se está armando. En particular, mediante su valor se determina en qué momento comienza un nuevo mensaje y por lo tanto cuándo se escriben los metadatos. Esta señal está presente en la versión original del bloque (ver Sección B.1), fue levemente modificada de forma acorde con el resto de las modificaciones realizadas. La señal `message_count_c` indica la cantidad de mensajes límite necesarios para cambiar de vector, es decir es constante para cada ciclo. La señal `message_count` lleva la cuenta de la cantidad de mensajes del vector actual. El

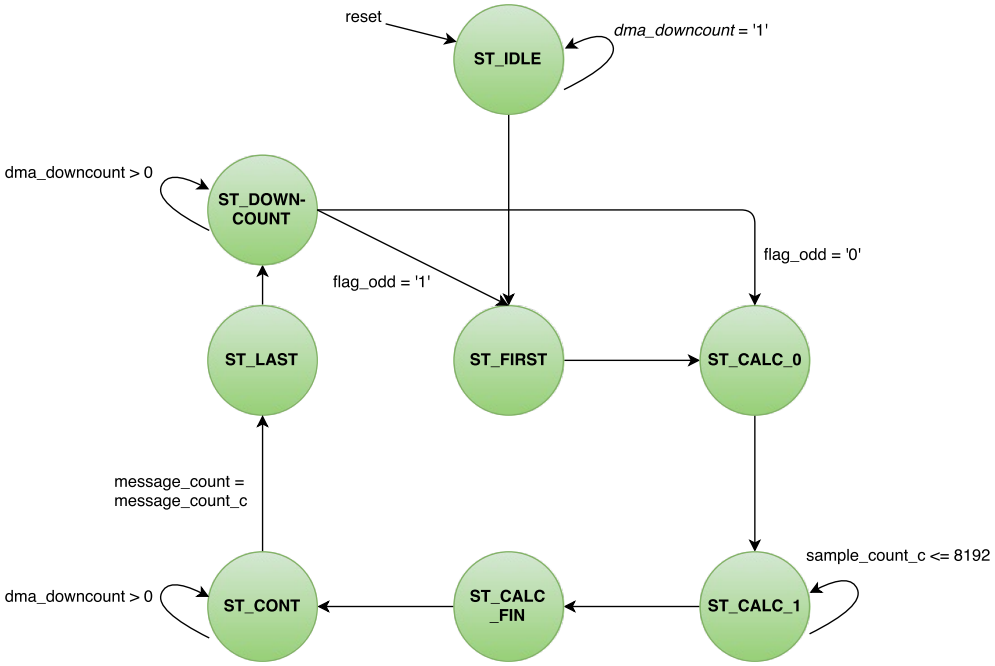


Figura 4.1: Diagrama de estados para la generación de `n` y `m` implementada en el bloque `fifo_writer`.

contador `sample_count` toma el valor de la cantidad de muestras escritas en el *fifo* de datos correspondientes al vector actual y la señal `sample_count_c` es utilizada para calcular los valores de `n` y `m`. En este sentido, la misma se inicializa con la cantidad de muestras enviadas en el primer mensaje y luego incrementa su valor de a 508 muestras (cantidad de muestras dentro de un mensaje), de esta forma se calcula la cantidad de mensajes que son necesarios para culminar el envío del vector. Esto último corresponde con el valor que deberá tomar la señal `message_count_c` a lo largo de todo el vector. La señal `flag_odd` determina un caso particular donde los valores de `n` y `m` comienzan nuevamente con los valores iniciales de la Tabla 4.2. Las siguientes señales son utilizadas para definir los valores de `n` y `m`.

```
signal   nl  : unsigned (15 downto 0);
signal   ml  : unsigned (15 downto 0);
constant n0  : integer := 5;
constant n1  : integer := 5;
```

Las señales `nl` y `ml` guardan los valores de `n` y `m` calculados para la próxima iteración y son asignadas a las mismas en el momento correspondiente. La constante `n0` vale 5, primer valor de **n** de la tabla Tabla 4.2 (podría ser por ejemplo 6 correspondiendo a otra tabla con el mismo criterio de la Tabla 4.1). Por otro lado, `n1` toma el valor de 5 correspondiente al caso borde impuesto por le protocolo de comunicaciones.

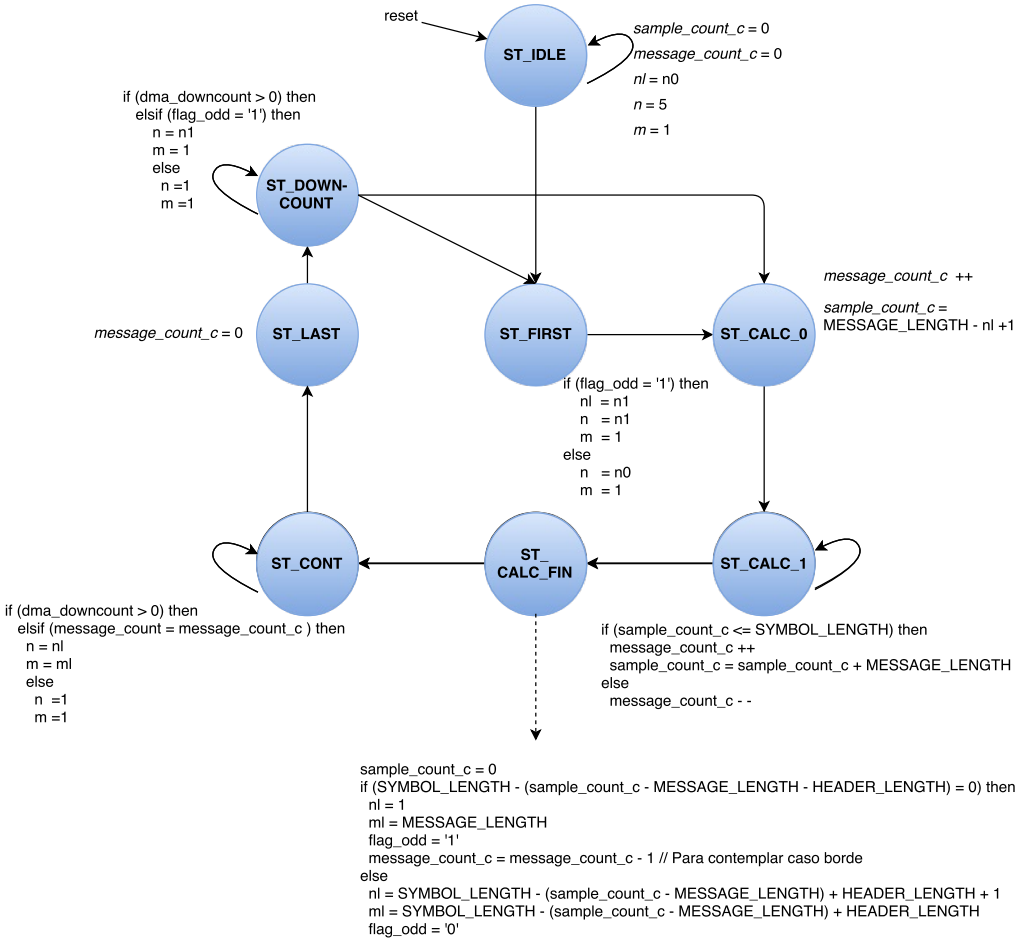


Figura 4.2: Diagrama de transición de estados `fifo_writer`. Valores de `n` y `m` y señales auxiliares

4.2.1.2. Validación de señalización

Para la validación de la señalización implementada en el bloque `fifo_writer` se realizaron una serie de simulaciones mediante el simulador del Quartus y varias pruebas de recepción de señales generadas en la placa.

Simulaciones A continuación se presentan dos capturas de la salida del vector de simulaciones. En la Figura 4.3 se puede observar el estado del bloque en los primeros 4 estados donde se realizan los cálculos correspondientes a `n` y `m`. En la Figura 4.4 el estado del `fifo_writer` durante el primer vector donde permanece la mayoría del tiempo en el estado **st_cont**. En las mismas se observa el comportamiento de las señales explicadas anteriormente.

Se realizaron pruebas de recepción de señales desde la placa con la señalización generada por el bloque `fifo_writer`. Los metadatos fueron generados de acuerdo al protocolo establecido anteriormente, de esta forma se señalizan todas las muestras en vectores del mismo largo. Los datos de las señales consistieron en el contador

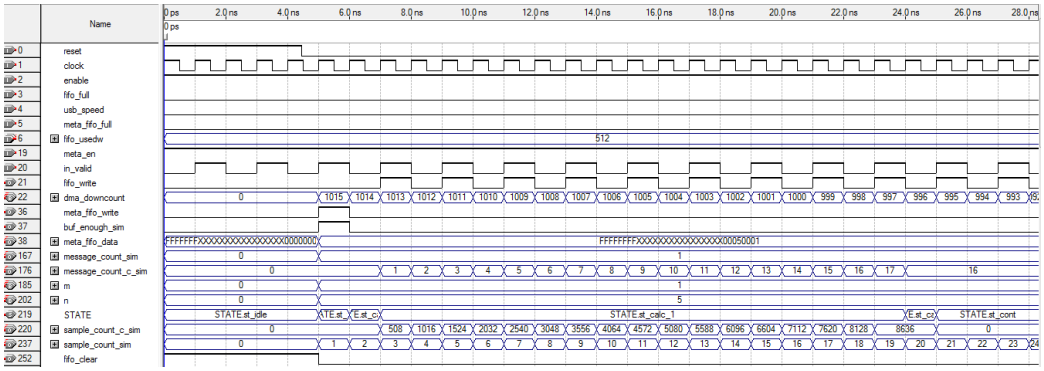


Figura 4.3: Simulación `fifo_writer`: Primeros estados para cálculo de `n` y `m`.

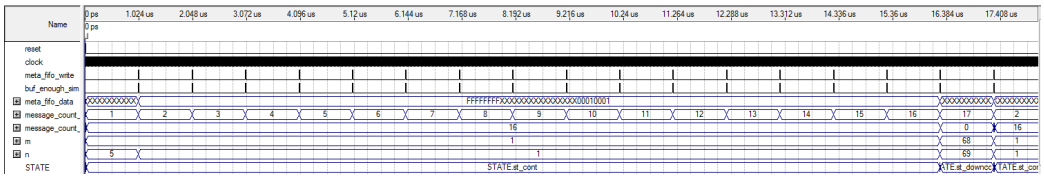


Figura 4.4: Simulación `fifo_writer`: Primer vector.

de referencia en el FPGA y muestras provenientes de la interfaz RF.

La validación de los datos se realizó en primera instancia únicamente para el caso del contador, debido a que en este caso se puede comparar contra una referencia fija. La misma se realizó mediante *scripts* de Octave y Python. Para esto se extrajeron los datos de cada mensaje ignorando los metadatos y se comparó contra la referencia. La validación de metadatos se realizó de forma similar comparando los valores de `n` y `m` recibidos contra una tabla de referencia como la Tabla 4.2 pero más extensa (la tabla tiene 2048 mensajes con *flags* distintos, luego se repiten los mismos de forma cíclica).

En algunos casos fue necesario repetir las pruebas corrigiendo el código del bloque debido a problemas en casos de borde. Los problemas se detectaron en la pruebas sobre el *hardware* debido a que las simulaciones realizadas no contemplaban todos los casos. Las simulaciones estaban limitadas por el largo de la simulación debido a la herramienta utilizada (simulador de Quartus).

Finalmente se validó la implementación mediante las pruebas realizadas. Por lo tanto la implementación realizada permite señalizar vectores en las condiciones descritas en la Sección 4.2.1.

Como se esperaba, debido a la naturaleza de los cambios implementados, se verificó que las modificaciones realizadas no alteraron cuantitativamente el uso de recursos del FPGA.

4.3. Modificaciones e implementación en el *Software*

4.3.1. Modificación del módulo `gr-osmosdr`

El módulo `gr-osmosdr` implementa un *source* sobre GNU Radio mediante la librería `libbladeRF`. Se revisó el código original del mismo y se resolvió que se podía mantener así para la implementación de la señalización. Debido a que la librería contiene funciones para la lectura y escritura del bus GPIO del FPGA, se modificó el código del módulo `gr-osmosdr` para configurar el bus GPIO en valores arbitrarios de forma de activar ciertas funciones en el FPGA. Se tuvo en cuenta que mediante el mismo también se configuran las condiciones de recepción como por ejemplo la banda utilizada en la interfaz de radio. El bloque `Osmocom Source` fue utilizado para configurar las siguientes funciones en el FPGA:

- Activar el uso de metadatos.
- Selección de fuente de datos Sección 3.1:
  - LMS (datos de la interfaz RF, fuente por defecto)
  - Contador interno del FGPA
  - *Digital Loopback*
- Agregar funciones sobre el FPGA.

En particular, se utilizó el *bit* 11 del periférico `nios_system_control` de los 5 que tiene libres (*bits* 11, 15, y 17 a 19), para la selección de funciones en el FPGA.

4.3.2. Desarrollo del bloque `messages2symbol`

Se implementó un nuevo bloque llamado `messages2symbol` sobre GNU Radio. Este se encarga de interpretar el contenido de los metadatos de los mensajes a su entrada para armar vectores según la señalización definida en la Tabla 4.1. En esta sección se usará de manera equivalente la palabra símbolo para representar los vectores.

El bloque fue desarrollado sobre un módulo *out-of-tree* de GNU Radio (ver Sección 2.2.1) denominado `gr-sdrenfpga`. A continuación se presenta la especificación del mismo:

1. Bloque de procesamiento de GNU Radio con entradas de tamaño de mensajes y tipo de datos complejos (`gr_complex`) y salida de largo de símbolo, e igual tipo de datos. El tamaño de la entrada y salida podrá ser configurable según el largo de mensajes utilizado ($message\_length = 512$ @ USB 3.0) y símbolos ($symbol\_length = 8192$ @ modo 3, ver Tabla 2.1) respectivamente. En la Figura 4.5 se presenta la interfaz del bloque sobre GNU Radio Companion.

2. El contenido de los mensajes debe tener el formato de la Tabla 3.2 a menos de la conversión de formato y escalado realizado por el bloque `Osmocom Source`.

Figura 4.5: Interfaz bloque `messages2symbol`.

3. Cada vector de salida corresponde a un símbolo completo, el cual está compuesto por muestras que vienen distribuidas a lo largo de varios mensajes de la entrada.

4. La distribución está especificada en el contenido de los *flags* de los metadatos de cada mensaje, donde el criterio es el definido en la Tabla 4.1.

El bloque `messages2symbol` interpreta el contenido de los *flags* `n` y `m` de metadatos y a su salida arma cada vector mediante las muestras válidas que contenga cada mensaje. El bloque solo entrega vectores con contenido íntegro según indique la señalización. El mismo admite ciertas secuencias en los valores de los *flags* `n` y `m`, las cuales deben ser como la Tabla 4.2. En caso de haber alguna incoherencia, se descartan las muestras recibidas y no se genera ninguna salida hasta que se arme un símbolo completo.

4.3.2.1. Funcionamiento del bloque `messages2symbol`

En los pasos descritos a continuación se explica el procesamiento del bloque `messages2symbol`, puntualizando el desarrollo de funciones en base a los requerimientos definidos anteriormente.

El bloque debe entregar a la salida vectores de tamaño `symbol_length` (8192), en función de entradas de tamaño largo de mensaje `message_length` (512) en base al contenido arbitrario de los metadatos de cada mensaje. Por lo tanto, en cada iteración de procesamiento se debe controlar la cantidad de muestras leídas, descartadas y salidas generadas garantizando el flujo correcto.

El bloque genera en promedio 1 símbolo cada 16.3 mensajes (para el caso de la aplicación), por lo tanto es necesario disponer de al menos 17 mensajes para generar cada salida. El bloque lee los metadatos y toma decisiones según su valor pero los mismos no se copian a la salida (podría agregarse una salida auxiliar con el contenido de estos). A la salida solo se copian datos, por lo tanto hay que tener en cuenta la posición de los mismos y en particular a qué posición del símbolo pertenecen, para esto se crearon distintos contadores y punteros.

Para la lectura de los metadatos hay que tener en cuenta la conversión de formato que realiza previamente el bloque `Osmocom Source`. Para interpretar los metadatos tal cual se generan en el FPGA es necesario revertir la conversión mencionada, para esto se podría hacer una función específica pero fue realizada con pocas líneas de código.

Una vez obtenidos los valores de los metadatos generados en el FPGA se deben tomar decisiones según su valor, se consideró decidir únicamente en función de los valores de `n` y `m`. Se podría haber utilizado además los valores de los *timestamps*

para la toma de decisiones. El bloque permite reportar en la interfaz los valores de los metadatos leídos, lo cual es útil si se pone a prueba la correcta generación de vectores en el FPGA. Por ejemplo, si los vectores generados tienen un salto grande en los *timestamps* de distintos mensajes esto es un indicio de problemas en la generación de los vectores, o bien en la recepción de muestras sobre GNU Radio.

En base a los valores de `n` y `m` se arma cada símbolo, leyendo las muestras que estos indiquen. Se definió que el bloque sea capaz de interpretar cierta cantidad de casos posibles de `n` y `m`. A continuación se explica en detalle el procesamiento del bloque y en particular cómo se comportan algunas variables que determinan las funciones del bloque.

4.3.2.2. Implementación del bloque `messages2symbol`

Funciones de GNU Radio utilizadas El procesamiento se basa en las siguientes funciones de GNU Radio (las mismas y otras necesarias están detalladas en la Sección C.2.1):

$forecast$:	Función para solicitar muestras en la entrada según la cantidad de salidas a generar.
$general\_work$:	Función donde se realiza el procesamiento de la entrada y se generan las salidas. Estas se indican mediante el comando *return*.
$consume\_each$:	Función para indicar la cantidad de muestras de la entrada a descartar debido a que ya fueron leídas o son ignoradas.

Variables de interés para el procesamiento, de las funciones mencionadas anteriormente:

$n\_outputs\_items$:	Cantidad de salidas posibles por llamada al $general\_work$.
$\&input\_items$:	Puntero a señal de entrada.
$\&output\_items$:	Puntero a señal de salida.
$n\_read\_samples$:	Contador de cantidad de muestras parcial del símbolo que se genera en la salida .

Etapas del procesamiento El procesamiento sigue la siguiente lógica:

1. Disponer de la cantidad de mensajes suficientes en la entrada para generar un símbolo en la salida (17, $d\_n\_messages = symbol\_length/message\_length + 1$), haciendo uso de la función $forecast$.

2. En cada llamada de la función $general\_work$ se pueden generar $n\_outputs\_items$ salidas (en caso de que los mensajes contengan información válida).

3. Realizar un for para generar las salidas en el cual se lee un mensaje por iteración. Se itera hasta completar un símbolo en la salida o hasta que se acaben los mensajes de la entrada.

4. Para cada iteración, interpretar los metadatos del mensaje, leyendo en particular los valores de `n` y `m`.

5. Decidir a qué caso de los estados corresponde. En los siguientes párrafos se presenta en primer lugar los estados posibles que puede tomar el bloque `messages2symbol` describiendo en algunos casos el comportamiento de algunas variables dentro de los mismos y en segundo lugar, las transiciones entre ellos.

   Estados del bloque `messages2symbol` A continuación se presentan los estados que puede tomar el bloque según los valores de `n` y `m` de cada mensaje.

   `first`: Primer mensaje de un nuevo símbolo, se descartan datos anteriores. Contiene muestras a partir de cierto índice indicado por `n` hasta el final del mensaje.
   La variable `n_read_samples` se inicializará en 0 y luego se acumularán las muestras que correspondan $(512 + 1 - n)$.

   `continuous`: Las muestras del mensaje están contenidas completamente en un símbolo.
   Se debe verificar (mediante `n_read_samples`) que las muestras del mensaje actual, sumadas a las acumuladas del símbolo no superen el largo del mismo ni que lo iguale. En caso de que se cumpla esta condición, se copian muestras a la salida. De lo contrario, se descartan las muestras.

   `ultimate`: Último mensaje de un símbolo.Contiene muestras de un símbolo el cual se completará con las muestras del mensaje comenzando por la primera después de los metadatos hasta la que indique el valor de `m`.
   Se debe verificar que la variable `n_read_samples` tome el valor de 8192. En caso de que se cumpla esta condición, se copian muestras a la salida. De lo contrario, se descartan las mismas.

   `end_ini`: El mensaje contiene muestras correspondientes a dos símbolos, finaliza uno anterior y comienza uno nuevo.
   Se debe verificar, una vez más mediante `n_read_samples`, que las muestras del mensaje más las acumuladas del símbolo (anterior) sean 8192. En caso de que se cumpla esta condición, se copian las muestras del símbolo anterior a la salida y se guardan las muestras del símbolo siguiente en una variable auxiliar. En caso contrario, se descartan los datos.

   `invalid`: Mensaje inválido. Se indica explícitamente que el mensaje no contiene muestras válidas. Este caso podría utilizarse para enviar un mensaje únicamente con metadatos.

   `problem`: Mensaje corrupto, no corresponde a ningún caso anterior.

   Todos estos casos están implementados mediante un $if - elsif - else$ para la llegada a alguno de los estados posibles.

`first` ← { `first, ultimate, end_ini, invalid, problem` } Siempre que el próximo estado sea un $first$ comienza un nuevo símbolo. Por ejemplo, si un símbolo anterior se rompió el bloque se puede volver a sincronizar con lo que indica el nuevo mensaje. En caso de que el mensaje anterior corresponda a un `continuous` no se admite el estado `first`. *Condiciones en* **`n`** *y* **`m`***:* $5 <= n <= 512$ & $m = 1$: Caso comienza y continúa un símbolo ó $5 <= n <= 512$ & $5 <= m < n$: Termina algo que se descarta y comienza uno nuevo: re-sincronización.
`continuous` ← {`first, end_ini, continuous` } Solo es válido que continúe el símbolo. Por ejemplo, cualquier mensaje intermedio de un símbolo o el segundo. *Condiciones en* **`n`** *y* **`m`***:* $n = 1$ & $m = 1$: Caso continúa un símbolo
`ultimate` ← {`continuous` } Solo es válida la llegada a este estado desde `continuous`. *Condiciones en* **`n`** *y* **`m`***:* $n = 1$ & $5 <= m <= 512$: Caso termina un símbolo
`end_ini` ← {`continuous` } Solo es válida la llegada a este estado desde `continuous`. *Condiciones en* **`n`** *y* **`m`***:* $5 <= m < n <= 512$: Caso termina un símbolo y comienza nuevo.
`invalid` ← {`all` } Se puede llegar a este estado desde cualquier otro y siempre se descarta el símbolo. *Condiciones en* **`n`** *y* **`m`***:* $n = 0$ & $m = 0$
`problem` ← {`all` } Análogo a estado `invalid`. *Condiciones en* **`n`** *y* **`m`***:* En este caso `n` y `m` toman cualquier valor que no cumple con ninguna de las condiciones mencionadas anteriormente.

Tabla 4.4: Transición entre estados del bloque `messages2symbol`.

Transición entre estados Los estados pueden tener únicamente ciertas transiciones válidas. Por ejemplo, para llegar a un caso **ultimate** el mensaje anterior debe corresponder necesariamente a un caso **continuous**, para llegar a un caso **continuous** el mensaje anterior debe haber sido clasificado como **first**, **end_ini** o **continuous**. Las secuencias válidas se muestran en el diagrama de la Tabla 4.4.

6. Si el estado del mensaje actual es válido se realiza el procesamiento según

el contenido del mismo si no se descarta.

7. Para procesar según el estado correspondiente del mensaje se realiza algo similar a un *case* mediante un if para cada estado.

   Según el caso se leen las muestras de la entrada correspondientes a una parte de un símbolo. En general se determinan los índices asociados a los comienzos y fin de muestras válidas (indicados precisamente por `n` y `m`) y luego se copia el rango determinado a la salida. Para el caso `end_ini` se guardan las muestras correspondientes al próximo símbolo en una variable auxiliar y luego en la próxima iteración se copian a la salida.

8. Se actualiza un contador de muestras leídas a lo largo de determinado símbolo.

9. Se consume un mensaje en la entrada mediante `consume_each`.

10. Si se completó un símbolo en la salida se indica mediante *return*.

4.3.2.3. Validación del bloque `messages2symbol`

Para la validación del bloque `messages2symbol` se definieron cuatro pruebas reproducibles. Las pruebas consisten en poner señales con metadatos con el criterio de señalización de vectores de la Tabla 4.1 en el bloque y obtener a su salida los vectores correspondientes (en particular señales que no respetan el criterio). Las mismas permitieron depurar e implementar los posibles casos de funcionamiento (por ejemplo transiciones de estados) que se terminaron de definir durante la ejecución de las mismas.

Las señales y los metadatos utilizados en las pruebas son de distintas fuentes y características. En el caso de los datos se utilizaron señales generadas en FPGA, provenientes del contador interno o bien de la interfaz RF. En el caso de los metadatos se utilizaron los *flags* generados en el FPGA sin modificar y en algunos casos se alteraron los mismos mediante *scripts* auxiliares. En particular, se realizó la integración con la implementación de la señalización sobre el FPGA y parte de la cadena de procesamiento del módulo `gr-isdbt` para verificar el funcionamiento del bloque en condiciones similares a las de la aplicación. La validación se realizó mediante *scripts* de Octave y sobre *flowgraphs* de GNU Radio.

Las pruebas que se definieron fueron las siguientes:

Prueba 1 : Datos provenientes del *hardware*.

Datos: Contador interno del FPGA.

Flags: Valores de `n` y `m` generados por el bloque `fifo_writer`.

Esta prueba permite verificar la correcta interpretación de los valores de `n` y `m` y en particular la integridad de la salida. La misma debe corresponder a un contador y por lo tanto es fácil de verificar.

Prueba 2 : Datos provenientes del *hardware*.

Datos: Interfaz de aire.

Flags: Valores de `n` y `m` generados por el bloque `fifo_writer`.

Esta prueba es similar a la anterior pero con valores de datos arbitrarios. Para verificar la salida se comparó directamente contra los datos de los mensajes de la entrada sin pasar por el bloque `messages2symbol`.

Prueba 3.1 : *Flags* alterados de forma manual.

Datos: Contador interno del FPGA.

Flags: Valores de `n` y `m` alterados a criterio

Esta prueba permite verificar el funcionamiento del bloque en casos particulares definidos a criterio. Por ejemplo, se puede verificar casos de borde patológicos que requieran que el procesamiento sea particular.

Prueba 3.2 *Flags* alterados de forma automática.

Datos: Contador interno del FPGA..

Flags: Valores de `n` y `m` modificados de forma aleatoria.

Esta prueba permite, de forma automática, generar casos donde no se respete el protocolo. Tiene la utilidad de generar los casos de forma automática pero la desventaja de que es difícil verificar el funcionamiento correcto ya que no se pudo automatizar la respuesta esperada. Esta prueba no fue realizada debido a la dificultad mencionada.

Prueba 4 : Datos de DTV capturados por la placa bladeRF.

Datos: Interfaz de aire.

Flags: Valores de `n` y `m` generados por el bloque `fifo_writer`.

Esta prueba permite integrar la recepción de señales mediante la interfaz RF, la generación de *flags* para señalizar el flujo de muestras sobre el FPGA y finalmente el correcto funcionamiento del bloque `messages2symbol` de forma continua. Se realizó en las condiciones de recepción de señales de DTV, en particular, a la frecuencia de muestreo requerida para la aplicación.

En conclusión, las pruebas realizadas permitieron la validación del bloque `messages2symbol`. En particular se verificó el correcto funcionamiento integrando el mismo al diseño de la señalización realizado sobre el *hardware* y parte de la cadena de procesamiento del módulo `gr-isdbt`. Por lo tanto, se verificó que el bloque permite armar a su salida los vectores que se generan en los bloques de procesamiento del *hardware* si se respeta el criterio definido en la Tabla 4.1 en la señalización de dichos vectores.

4.4. Conclusiones

Debido a que algunos bloques que posteriormente se desarrollaron sobre el FPGA manejan señales vectoriales correspondientes a símbolos OFDM, fue necesario modificar la interfaz para poder enviar vectores (símbolos) hacia el *software*.

En base a este requerimiento, se definió un protocolo para señalizar vectores. Se modificó el bloque `fifo_writer` del *hardware* para implementar la señalización de forma que sea posible generar vectores en el FPGA mediante el uso de metadatos. Se validó el bloque de forma individual mediante el uso de la herramienta de simulación del Quartus y la realización de distintas pruebas en el *hardware*. Se concluyó que la herramienta de simulación utilizada tenía limitaciones para el largo de la misma, que dificultaban la etapa de verificación.

Se modificó el módulo `gr-osmosdr` para poder configurar funciones sobre el FPGA de la placa bladeRF, en particular el uso de metadatos. Se verificó que esta función hubiera quedado implementada realizando distintas capturas, como también que no se hubiera afectado el comportamiento integral del mismo.

Se implementó el bloque `messages2symbol` sobre GNU Radio para interpretar los metadatos y armar los vectores que se generan en el *hardware*. Este bloque también permite realizar otras funciones según los metadatos, como por ejemplo la verificación de *timestamps*. Se validó el funcionamiento del bloque generando distintas entradas con diversas condiciones para poder cubrir la mayor cantidad de situaciones posibles.

En el marco de la validación de esta etapa del proyecto se realizaron pruebas de integración utilizando los elementos anteriores junto al módulo `gr-isdbt`, verificando que la implementación funciona en condiciones similares a la aplicación requerida.

Las modificaciones realizadas sobre la interfaz y la implementación del bloque `messages2symbol` permiten manejar señales vectoriales entre la placa bladeRF y la plataforma GNU Radio de forma transparente para el resto del sistema. La implementación permite variar el largo de vectores utilizado de forma paramétrica, cambiando constantes en el *hardware* y en el *software*, de esta forma se puede extender la utilidad a otras aplicaciones que requieran el envío de vectores.

El uso de metadatos podría extenderse a más funciones como el envío de información extra y de control del flujo. Algunos ejemplos son:

- Enumeración de los vectores generados sobre el *hardware*.
- Exponente de escalado de la señal utilizando una representación de punto flotante por bloques (*Block Floating Point*) para mejorar la precisión.
- Resultados de los bloques de procesamiento sobre el *hardware*.

Capítulo 5

Solución Propuesta - *Hardware* FPGA

En el presente capítulo se expone el diseño realizado en el FPGA. Teniendo en cuenta la implementación en *software* que resuelve el problema propuesto (módulo `gr-isdbt` de GNU Radio), la arquitectura sin modificar del FPGA de la bladeRF y los recursos disponibles del mismo, se diseñó una arquitectura funcional dentro del FPGA que realiza la misma función que el primer bloque del módulo mencionado: el bloque `OFDM Sym Acquisition` e incorpora un filtro pasa bajos a la entrada y un bloque de interfaz en la salida. El procesamiento continua en el *software* de manera transparente para el resto de los bloques.

En este sentido, el primer paso de esta etapa consistió en la investigación de los bloques de sincronismo del demodulador OFDM para un receptor ISDB-T (el módulo `gr-isdbt` mencionado en la Sección 3.2) y de los sistemas SDR, en particular la arquitectura de la placa bladeRF y del FPGA contenida en la misma. Todos estos puntos fueron críticos para el desarrollo en el FPGA, los cuales fueron presentados en el Capítulo 2 y en el Capítulo 3.

En la Sección 3.2 se vio que el bloque `OFDM Sym Acquisition` del módulo `gr-isdbt` realiza la sincronización temporal del símbolo y la corrección en frecuencia. De su estudio se definió que los dos puntos determinantes para el diseño de la arquitectura relacionados con las funciones del mismo fueron la cantidad de muestras con las que se debía trabajar y el tiempo que debían permanecer inalteradas. Se buscó una solución realizable en función de los recursos disponibles. En este sentido, la principal limitante fue la de memoria disponible en el FPGA de la bladeRF. Por otro lado, otro aspecto problemático de la implementación en *hardware* corresponde a la representación numérica, ya que las limitantes en las operaciones generan problemas numéricos.

En las secciones siguientes se comentarán los requerimientos de la implementación, los recursos y limitantes del *hardware* así como las herramientas con las que se desarrolló el diseño. La solución realizada sobre el *hardware* será presentada mencionando los aspectos generales del diseño como la arquitectura de procesamiento, criterios de diseño y algunos detalles de su implementación como los algoritmos diseñados. Se focalizará en los detalles específicos de la solución propuesta, así como también sus diferencias con la implementación en *software* ya existente. Además se verán algunas alternativas a lo realizado así como también las ventajas

y desventajas de las soluciones planteadas.

5.1. Diseño en *Hardware*: limitantes, recursos y requisitos

Para realizar el diseño sobre el FPGA fue necesario conocer y considerar en detalle los recursos disponibles sobre el mismo así como los requerimientos de la aplicación. El diseño se realizó en función del análisis de ambos aspectos de acuerdo a las herramientas disponibles.

En esta sección se presentan las características del *hardware* de los sistemas SDR que condicionaron el diseño, en particular del FPGA de la placa bladeRF.

Se estudian algunos aspectos limitantes del desarrollo sobre *hardware* y sobre el FPGA como el tipo de datos, las limitaciones de la interfaz de radio y las comunicaciones con la PC así como también el diseño en lenguaje VHDL entre otros puntos. En particular se presentarán los recursos del FPGA y el resultado del consumo de la implementación considerando de forma cuantitativa cuales fueron más restrictivos.

5.1.1. Limitantes del *hardware*

El *hardware* correspondiente a los sistemas SDR presenta diferentes características para la recepción, las cuales fueron presentadas en general en la Sección 2.2 y en particular para la placa bladeRF en la Sección 3.1. Algunos de los aspectos se deberán tener en cuenta al momento del desarrollo de bloques de procesamiento sobre el FPGA de la placa bladeRF. Entre los puntos a considerar se mencionan los siguientes debido a su influencia en el diseño:

Limitantes generadas por la arquitectura del FPGA original y placa bladeRF Se recuerda al lector, que en el caso de la placa bladeRF, las muestras correspondientes a la señal recibida por la interfaz de radio son suministradas por el *chip* LMS hacia el bloque `lms6002d` el cual se encarga de extraer la componente I y Q en distintos puertos y de generar la señal `rx_sample_valid` que indica que ambas componentes son correctas (Sección 3.1). Luego el bloque `rx_iq_correction` realiza un ajuste de fase y ganancia. El formato de las muestras está determinado por el conversor AD del LMS6002D, el cual las genera con 12 *bits* en complemento a 2 para cada uno de las componentes en fase y cuadratura. A la salida del bloque se extiende el ancho de *bits* a 16 generando las señales de entrada al sistema implementado, las cuales serán válidas cada dos períodos de reloj. El sistema funciona a una frecuencia de muestreo $f_s = 512/63$ MHz $\approx 8{,}13$ MHz, la cual está determinada por la señal de reloj `lms_rx_clock_out` de 16,26 MHz fijada desde la PC. Esta señal de reloj es utilizada por la cadena de recepción sobre el FPGA y la misma se utilizará como reloj de los bloques de procesamiento.

Tal como se estudió en los capítulos anteriores, el envío de muestras desde el FPGA hacia la PC se realiza en paquetes denominados mensajes. Mediante el

protocolo de comunicación de vectores entre el *hardware* y la PC presentado en el capítulo anterior, se permite el envío de vectores a través de mensajes señalizados (ya no de mensajes con muestras independientes). Por lo tanto, esta solución implica que los bloques de procesamiento a desarrollar respeten esta interfaz definida.

Arquitectura interna del FPGA Para definir la cadena de procesamiento sobre el FPGA es necesario conocer en detalle la arquitectura presente en el mismo. En el caso de la bladeRF fue necesario estudiar en detalle la cadena de recepción y las condiciones de operación de la misma, en particular los relojes de funcionamiento del circuito, las interfaces de datos y los procesos de control. Estos puntos fueron presentados en la Sección 3.1 y cada uno implicó condiciones para el diseño.

La cadena de recepción fue estudiada en detalle, adaptada para la comunicación vectorial y se le agregó procesamiento de señales. Una vez definida la ubicación de los bloques de procesamiento, las condiciones para el diseño fueron las de respetar las interfaces y la frecuencia de operación de las mismas. Es decir, el procesamiento interno de los bloques es bastante independiente de la cadena de procesamiento pero sus interfaces de entrada y salida están condicionadas por la misma, en particular la frecuencia de funcionamiento.

Para el procesamiento de señales se utilizó el reloj de muestreo, dado que en la aplicación la frecuencia de muestreo está fija ($f_s = 512/63$ MHz $\approx 8{,}13$ MHz), esto fue una condición permanente durante el diseño. Para el procesamiento podría utilizarse un reloj diferente al de muestreo, en particular de un valor mayor y relativamente alto. Esto último permite reducir algunos tiempos de procesamiento y posiblemente ahorrar recursos de memoria o elementos lógicos. En el caso general hay que considerar la interfaz de salida de la cadena de procesamiento sobre el FPGA ya que que se debe respetar su reloj de funcionamiento y por lo tanto es necesario adaptar la cadencia de las muestras mediante un *buffer*. En el caso de la bladeRF, la interfaz de salida corresponde a la interfaz FX3 con el USB, la cual opera a 100MHz (por más detalles ir a Sección 3.1) y el *buffer* corresponde a memorias tipo *fifo* presentes en la solución actual (ver Sección 3.4).

Parte del estudio de la cadena de recepción de muestras consistió en el análisis de los procesos de control que estaban involucrados. Los mismos son coordinados por un microprocesador embebido NIOS II de Altera que cuenta con diversos periféricos para el manejo de los integrados de la placa, por ejemplo para el LMS y el FX3. Se pudo comprobar que el agregado de procesamiento en la cadena de recepción no se vería afectado por el microprocesador en cuestión como tampoco afectaría el funcionamiento del mismo. Se puede obtener más información de este microprocesador en la Sección 3.1.

Limitaciones de la interfaz de radio Las principales limitantes de la interfaz de radio son la dependencia del transmisor de la señal de televisión digital y las condiciones de recepción.

Respecto al transmisor, solo existen disponibles los cinco canales abiertos presentados en la Sección 2.1 y en este caso la señal transmitida varía en el contenido de forma arbitraria y en potencia de acuerdo a la ubicación del receptor respecto

al transmisor generando una clara dependencia del transmisor. Una solución a este problema, podría ser retransmitir señales desde un SDR hacia la placa bladeRF. En este caso, se podría encontrar otras barreras como ser el establecimiento de un buen enlace de radio entre ambos SDR, el cual estará muy condicionado a las características del transmisor SDR (potencia transmitida y ganancia de la antena) y la configuración del SDR para transmitir (ganancia del transmisor, banda de frecuencia a utilizar). Sin embargo, esta opción tiene la ventaja de poder recibir señales conocidas de forma prácticamente repetitiva y controlada.

Respecto al receptor, las principales limitantes son: la antena utilizada para la recepción (ganancia, direccionalidad y tamaño), la configuración de frecuencia de muestreo y ancho de banda, la necesidad de compensar *offset* de continua y desbalance IQ y la representación condicionada (formato y cantidad de *bits* de las muestras). Por detalles de estos puntos se invita al lector a repasar la sección Sección 3.1.

Las condiciones que generan estas limitantes sobre el desarrollo del diseño son los problemas para reproducir pruebas con señales de RF y las condiciones sobre el ancho de banda, frecuencia de muestreo y representación de la señal recibida. Notar que la solución propuesta en *software* también tuvo que sobrevenir esta limitante. En este caso, debido al entorno de desarrollo utilizado la reproducción de pruebas idénticas sí es posible a diferencia del caso *hardware*[1].

Condiciones de operación de la interfaz entre el *hardware* y la PC. En el capítulo anterior se presentó la solución para el envío de vectores entre la bladeRF y la PC. En caso de que el *hardware* SDR sea otro, habría que realizar una solución similar sobre otra plataforma. Para el caso de la bladeRF la solución implica que la interfaz debe respetar el protocolo definido para la comunicación. Una limitante general de esta interfaz es el ancho de banda, el cual está condicionado por el USB. En el caso de la bladeRF permite utilizar un USB 3.0 para alcanzar el máximo de 40 MS/s limitado por el *chip* de muestreo LMS (ver Sección 3.1). Por lo tanto, dado que la aplicación utiliza una frecuencia de muestreo de $f_s = 512/63$ MHz $\approx$ 8,13 MHz, bastante menor al límite anterior, la condición del USB no será un problema. En este sentido, la solución resuelta en *software* a pesar de tener la misma restricción de ancho de banda para la recepción de muestras, no tuvo que limitarse a la frecuencia de muestreo de la aplicación ya que dentro de GNU Radio es posible sub-muestrear la señal de entrada.

La parte de la interfaz correspondiente a la PC puede representar limitantes como las librerías y bloques de software utilizados para la recepción de muestras. En el caso de GNU Radio y la placa bladeRF se utiliza el módulo `gr-osmosdr` presentado en la Sección 3.3 el cual hace uso de la librería `libbladerf`. Por lo tanto la comunicación está condicionada por las funciones presentes sobre la librería como las correspondientes a configurar la frecuencia de muestreo y parámetros de la recepción. Entre las configuraciones del módulo, se encuentra el tamaño

[1]La placa bladeRF cuenta con una herramienta llamada *digital loopback* que serviría para este problema. La misma no se pudo utilizar con éxito en las condiciones de operación de la aplicación.

correspondiente a los *buffers* utilizados para la recepción de muestras, el cual debe ser adecuado debido a que allí se almacenan para su posterior procesamiento. En el párrafo siguiente se mencionan los problemas con pérdidas de muestras asociados al tamaño limitado de los *buffers*.

Problemas con pérdida de muestras En la comunicación entre el *hardware* y la PC es posible que exista el problema de pérdida de muestras. Para identificar este problema en el camino hacia la PC, se realizaron pruebas de recepción de señales correspondientes al contador interno del FPGA de la bladeRF. La naturaleza de estas señales permitiría verificar fácilmente si había saltos de datos. Se contó con dos herramientas para la verificación de la integridad de los datos y los metadatos (timestamps): el bloque `messages2symbol` en GNU Radio y el USB Blaster en el FPGA. Para esto último, se realizó la conexión del USB Blaster a través de la interfaz JTAG del FPGA, para el uso de la herramienta de Altera *SignalTap II*.

Mediante la prueba se encontró que las muestras generadas en el FPGA no tenían saltos en los *timestamps* mientras que en GNU Radio sí.

Se determinó que el problema se debía a las limitaciones en el *buffer* en la PC. Se realizó un aumento de la capacidad de los *buffers* configurando el bloque de GNU Radio `Osmocom Source`. Luego se verificó que las señales generadas en el FPGA se recibían íntegramente en la PC. Para el resto del proyecto se utilizó la misma configuración de los *buffers* para evitar la pérdida de muestras. Se considera que en general es necesario revisar la configuración de los *buffers* de acuerdo a los requerimientos de la aplicación (como por ejemplo la frecuencia de muestreo).

Diseño de procesamiento mediante lenguaje de descripción de *hardware* (VHDL). El diseño en el FPGA se realiza mediante la descripción de un circuito digital con el lenguaje VHDL. VHDL es el acrónimo entre la combinación de VHSIC y HDL, donde VHSIC es el acrónimo de *Very High Speed Integrated Circuit* y HDL el acrónimo de *Hardware Description Language*. El circuito está compuesto por diferentes bloques que alojan elementos lógicos y memoria para realizar el procesamiento de señales.

En el caso de la bladeRF se utiliza como punto de partida el proyecto de Quartus provisto por el fabricante [22] el cual se presenta en general en la Sección 3.1. El procesamiento a incluir en el *chip* se realiza editando el código correspondiente al FPGA y generando imágenes a cargar sobre el mismo sobre un proyecto de Quartus. El programa Quartus se utiliza para el desarrollo de un proyecto asociado al FPGA, entre sus principales funciones se encuentra la compilación del código, generación de imágenes del FPGA y el uso de herramientas asociadas al *debug* del FPGA (analizador lógico y lectura/escritura de memoria en el *chip*). Por otra parte, para la simulación del circuito se pueden utilizar distintos programas como ModelSim el cual fue el programa utilizado en este proyecto. Este programa está disponible en una versión libre y gratuita junto al Quartus. El proyecto fue realizado mediante el uso de las versiones 15 y 10 Quartus y ModelSim respectivamente. En la Apéndice C se presentan algunos detalles de las versiones utilizadas.

Entre otras ventajas del ModelSim se destaca la rapidez para realizar simula-

ciones complejas tanto de extensión temporal como cobertura de casos. Para esto se debe desarrollar código que permita la compilación y ejecución de dichas simulaciones. También se menciona como aspecto favorable su interfaz gráfica ya que es bastante intuitiva, flexible y práctica.

Las simulaciones se realizan mediante *test-benches* que ponen a prueba distintos bloques. Un *test-bench* consiste en un código (generalmente escrito en VHDL) que representa un circuito. El circuito está compuesto por procesos que generan estímulos sobre el bloque bajo prueba y contiene conexiones en las entradas y salidas del mismo. Mediante la simulación del comportamiento del *test-bench* se pueden visualizar y almacenar los resultados intermedios así como pausar o detener la simulación. Los estímulos y los resultados se pueden leer y escribir respectivamente desde archivos en la PC, esto permite realizar pruebas con señales de tamaño relativamente grande. La validación de cada bloque, en términos generales, se puede dividir en dos grandes etapas de simulación, una primera individual y una segunda de integración al resto de los bloques. Para cualquiera de estas dos, se implementó un *test-bench* en VHDL que simula las entradas del mismo o las adquiere de un archivo en memoria y en algunos casos verifica de manera automática su salida, mientras que en otros, guarda en archivos la salida para luego ser verificada mediante *scripts* en Octave o en algunos casos de *flowgraphs* del GNU Radio Companion. La incorporación del ModelSim y de los *test-benches* como herramientas para la etapa de pruebas fue determinante para la validación de algunos bloques que se desarrollaron sobre el FPGA debido al tipo de datos con los que se trabajó (formato y extensión de señales de prueba). En el Capítulo siguiente se presentarán las simulaciones realizadas mediante estas herramientas para las pruebas y validación de la implementación en el FPGA.

Recursos disponibles para representación de señales y desarrollo de algoritmos de procesamiento. La representación de las señales de entrada al FPGA de la bladeRF está determinada por el *chip* LMS (como se recordaba anteriormente). La salida del FPGA hacia el PC está definida de la misma forma y el bloque `Osmocom Source` convierte el formato a `gr_complex` (muestras complejas en punto flotante en GNU Radio). El procesamiento a realizar sobre el FPGA debe mantener a su salida el mismo formato ya que sino sería necesario adaptar la interfaz de datos hacia la PC. Un cambio de formato podría consistir en un cambio en la representación a distinta cantidad de *bits* o pasar de punto fijo a punto flotante (ya sea por bloques o por muestras independientes). Esto podría permitir mejorar la representación, lo cual puede ser necesario de acuerdo al procesamiento a realizar sobre el FPGA (como se verá en las secciones siguientes). Para adaptar la interfaz de datos sería necesario realizar cambios en el FPGA y en la PC de forma similar a la solución de la comunicación vectorial presentada en el Capítulo 4. En particular podría ser necesario analizar la posibilidad del envío de un flujo de muestras a una cadencia de bits mayor a la original (debido al aumento de cantidad de *bits* de representación, por ejemplo si se aumentara de 16 *bits* a 32 *bits* se duplicaría la cadencia al doble).

Si se mantiene el formato original de la representación, el procesamiento debe

ser tal que su salida respete este formato. Esto supone una limitante para la precisión de las operaciones numéricas. Por otra parte la representación interna del procesamiento sobre el FPGA está condicionada a los recursos disponibles sobre el FPGA. En el párrafo siguiente se verán los tipos de operaciones numéricas que se pueden realizar sobre el FPGA y los algoritmos involucrados.

Tipos de operaciones numéricas sobre el FPGA Las operaciones numéricas que se podrán realizar en el FPGA se pueden dividir en dos grandes grupos. El primero corresponde a las operaciones de bajo nivel implementadas mediante bloques elementales como sumas, restas y multiplicaciones mientras que el segundo abarca las operaciones que son implementadas mediante el uso de tablas almacenadas en memoria. Este último tipo corresponde en general a operaciones más complejas mediante el uso de bloques específicos.

Cada tipo de operaciones tiene ventajas y desventajas según la representación de la señal y los recursos disponibles. Al momento de la implementación de alguna función específica se puede utilizar una combinación de los distintos tipos de operaciones, de forma de lograr mejores resultados de acuerdo a los requerimientos y recursos disponibles.

Las operaciones de bajo nivel implican la implementación de un algoritmo para realizar cada función. Por ejemplo, métodos numéricos para realizar funciones complejas mediante uso de funciones elementales como sumas, restas y multiplicaciones. Este caso tiene la ventaja de que se puede diseñar a la medida de los requerimientos. El FPGA dispone de bloques multiplicadores embebidos sobre el mismo que permiten implementar multiplicaciones sin utilizar otros recursos de lógica (ver sección siguiente).

En el caso del uso de tablas, éstas generalmente ocupan una cantidad considerable de memoria, en particular si se pretende tener buena precisión en las operaciones. Tienen la ventaja de que se pueden generar para cubrir cierto número de casos con buena precisión, pero el mismo está limitado por el largo de la tabla. Las mismas se pueden utilizar para el cálculo de valores de funciones arbitrarias, por ejemplo, funciones trigonométricas en determinado rango.

Finalmente, el uso de bloques específicos tiene la ventaja de que pueden implementar operaciones complejas, en algunos casos pueden ser paramétricos (por ejemplo representación, precisión, latencia, parámetros de la función). Como aspecto desfavorable se podría mencionar que los bloques disponibles pueden resultar inadecuados para los requerimientos, en particular si presentan limitaciones de funcionamiento (por ejemplo interfaz del bloque, latencia de procesamiento, precisión y representación del resultado, etc.). Estos últimos, muchas veces se encuentran disponibles en las librerías de los fabricantes (Altera, Nuand, etc.). Algunos ejemplos son: filtros, FFT, operaciones no lineales como logaritmos, exponenciales, raíces, etc.

5.1.2. Recursos del FPGA

En la Sección 3.1 se presentaron los recursos utilizados por el FPGA sin modificar. El lector los puede repasar en la Tabla 5.1 bajo la columna *Utilización Original*. En la columna *Utilización con modificación* de la misma tabla se presentan los recursos utilizados al finalizar la implementación. El lector puede observar que el recurso que más se consumió fue el de memoria (los recursos disponibles del FPGA inicialmente estaba utilizado un 10 % mientras que al finalizar la implementación se utilizó un 50 %) debido al tipo de datos y a las cantidades que se manejaron esto se profundizará en la próxima sección.

Obsérvese que si se hubiera utilizado el modelo x40 del FPGA (alternativa de bladeRF) no habría sido suficiente debido a la cantidad de bloques M9K necesarios (la versión x40 tiene tan solo 126) [1]. En cuanto a los elementos lógicos, a pesar de que habría sido suficiente, se hubiera utilizado un 63 % de los disponibles. Como se esperaba, debido a la extensión de los datos con los que se debe trabajar, el consumo de memoria fue el que creció en mayor proporción respecto a los recursos disponibles al comienzo de la implementación.

Si bien el uso luego de la implementación del resto de los recursos no fue tan grande respecto al total disponible, se puede observar que su uso aumentó en algunos casos más de doble de lo que estaba en uso (*Elementos lógicos*) y en otros se quintuplicó (*Multiplicadores*).

Recurso	**Utilización Original**	**Utilización con modificación**
Elementos lógicos	8886/114.480 (8 %)	24.844/114.480 (22 %)
M9Ks	57/432 (13 %)	217/432 (50 %)
Multiplicadores	12/532 (2 %)	52/532 (10 %)
Bits de Memoria	376.832/3.981.312 (9 %)	1.687.552/3.981.312 (42 %)
Bits de Implementación de Memoria	526.312/3.981.312 (13 %)	1.999.872/3.981.312 (50 %)

Tabla 5.1: Recursos utilizados por el FPGA luego de implementar el diseño propuesto y recursos sin el mismo.

5.1.3. Requisitos

En esta sección se enumeran los requisitos de la aplicación. Como se mencionó en la introducción del capítulo, el procesamiento sobre *hardware* (a realizarse sobre un bloque específico) se deberá comportar como el primer bloque del módulo `gr-isdbt` y tendrá las limitantes y condiciones dadas por el *hardware* y las herramientas disponibles. En este sentido, la mayoría de los requisitos se desprenderán del objetivo de imitar el comportamiento de este bloque. Por otro lado, otros requisitos se definen en función del tipo de señales a utilizar. Estos aspectos implican los siguientes puntos:

- El resto de la cadena de procesamiento presente sobre el *software* correspondiente al módulo `gr-isdbt` no podrá ser afectado por la implementación parcial en *hardware*, en particular, desde el punto de vista numérico. Esto implica que la implementación final debe permitir la correcta decodificación de las señales de vídeo por parte del resto de la cadena de procesamiento.

- El procesamiento se deberá adaptar al formato y tipo de datos ya definidos en la sección anterior. Esto implica que las muestras de la señal de entrada tendrán el formato dado por el *chip* LMS de la bladeRF y corresponden a señales transmitidas por los canales de DTV abierta de la Tabla 2.2. En este caso el largo de símbolo OFDM es de $N = 8192$ y los valores del prefijo cíclico de 1/8 o 1/16 ($L = 1024$ y $L = 512$ respectivamente).
- La salida del bloque deberá tener el tamaño de un símbolo de largo N (descartando el prefijo cíclico).
- Se deberá incluir un bloque que implemente un filtro pasa bajos para eliminar la presencia de portadoras de canales adyacentes.
- El envío de los símbolos desde el *hardware* hacia la PC se realizará respetando el protocolo definido sobre la interfaz modificada (presentada en el Capítulo 4). Esto último debe permitir incorporar bloques de procesamiento sobre la misma cadena en el FPGA, teniendo en cuenta que al final de la misma se respete la interfaz modificada.
- El procesamiento a implementar deberá realizar las dos funciones del bloque original del módulo `gr-isdbt`: la sincronización temporal y el ajuste fraccional en frecuencia. Para el caso de la sincronización temporal se utilizará el algoritmo de máxima verosimilitud (ver Sección 2.1.2). Esto lo deberá realizar, en lo posible, para todos los símbolos recibidos. En caso de no poder hacerlo con alguno, deberá contar con mecanismos de aviso para evitar generar grandes tiempos de latencia en el resto de la cadena de datos.
- De manera de procesar todos los símbolos, el tiempo de duración del procesamiento de cada símbolo deberá ser menor a lo que se demora en recibir 1 símbolo y su CP, lo que equivale a $N + L$ muestras recibidas a la frecuencia de muestreo.
- Debido al algoritmo a utilizar y el largo de símbolo, el *buffer* de entrada deberá ser suficientemente grande para poder almacenar las muestras y realizar su procesamiento. En un peor caso, la sincronización temporal requiere $2N + L$ muestras por lo tanto este será el tamaño mínimo de *buffer*. Por otro lado deberá ser suficientemente grande de manera de que no se pierdan muestras durante el procesamiento.
- El reloj de funcionamiento del bloque será el mismo que el de muestreo de la señal (reloj de $2 * f_s$, con $f_s = 512/63$ MHz $\approx 8{,}13$ MHz). Este genera una nueva muestra cada dos flancos de reloj. El algoritmo deberá ser definido para que funcione específicamente a la frecuencia de muestreo.

5.2. Solución propuesta

En esta sección se presentará la arquitectura del circuito diseñado en el FPGA 115KLE Cyclone 4 E de Altera perteneciente a la placa bladeRF que fue diseñada

para resolver el problema planteado. Se destacarán aquellos aspectos que diferencian una solución en *hardware* de una solución en *software* así como también criterios de diseño y alternativas de implementación.

5.2.1. Arquitectura e interfaz de datos

Arquitectura del FPGA La arquitectura implementada está compuesta por tres grandes bloques: el `sdr_en_fpga_filter`, el `sym_acquisition` y el `interface`. La comunicación de los mismos y la ubicación dentro de la arquitectura original se puede observar en la Figura 5.1. En la Sección 3.1 se analizó la posibilidad de incorporar procesamiento en el flujo de datos y se definió que el lugar sería entre el bloque `rx_iq_correction` y el `fifo_writer`, en azul en la figura mencionada. Se marca con una flecha el sentido que recorren las muestras en la cadena de procesamiento.

Indirectamente esto definió la interfaz básica que la arquitectura debía tener. La misma está compuesta por dos puertos de datos en la entrada de 16 *bits* para cada componente de la señal (fase y cuadratura) y dos en salida. Además tiene un puerto de dato válido en la entrada y otro análogo en la salida.

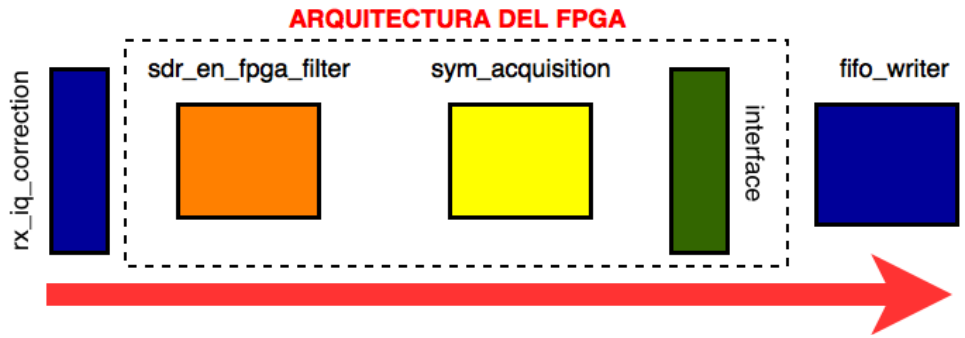


Figura 5.1: Arquitectura del FPGA implementada.

Luego de la etapa de corrección (bloque `rx_iq_correction`) comienza la arquitectura implementada. El primer bloque es el `sdr_en_fpga_filter` el cual está encargado de filtrar los datos de entrada al procesamiento para evitar superposición entre canales. El bloque `sym_acquisition` es el encargado de la primera etapa de sincronización del receptor OFDM y el bloque `interface` de entregarle la señal resultante al `fifo_writer`. Si se hubiera implementado la `FFT`, que es el próximo bloque en la cadena de demodulación OFDM, hubiera sido suficiente con respetar el protocolo de comunicaciones establecido entre el bloque `interface` y el `sym_acquisition`. En los próximos párrafos se explicarán sin entrar en detalles los bloques `sdr_en_fpga_filter` e `interface`. Por mayor información de dichos bloques referirse a la Sección E.1 y a la Sección E.2 respectivamente. El bloque `sym_acquisition` se describe en la Sección 5.2.2 y sus detalles se presentan en el Apéndice E.

Bloque `sdr_en_fpga_filter` En esta sección se expone brevemente el funcionamiento del filtro pasa bajos que implementa el bloque `sdr_en_fpga_filter` y se realizan algunos comentarios asociados a su implementación.

El mismo instancia dos bloques tipo FIR `fir_filter` de la biblioteca de Nuand, uno para la parte real de señal y otro para la parte imaginaria (esto es válido debido a la linealidad del filtro). Cada bloque `fir_filter` consiste básicamente en una cadena serial donde en cada etapa se aplica cada coeficiente de la respuesta al impulso del filtro correspondiente. Estos bloques se configuran con una respuesta al impulso mediante un parámetro genérico: $H = \begin{bmatrix} b_0 & b_1 & ... & b_{N-1} \end{bmatrix}$, que contiene N coeficientes de la respuesta al impulso del filtro diseñado (para la aplicación). Los mismos corresponden a constantes reales y por lo tanto, se pueden especificar con buena precisión. Finalmente las operaciones quedan limitadas por la cantidad de *bits* utilizada para representar las muestras.

Existen algunos bloques en las librerías que implementan estos tipos de filtros (FIR) pero no se utilizaron dado que presentan condiciones en la representación de la señal, consumo de recursos y en sus interfaces, como por ejemplo el filtro FIR provisto por Altera (*FIR Compiler*). Este caso permite crear filtros con determinados parámetros y generar *test-benches* para su emulación en Matlab (análogo a Octave). En caso del *software* alcanza con instanciar un filtro genérico y definir los parámetros que lo caracterizan (frecuencia de corte, tipo de ventana, etc.) y en general no presentan problemas numéricos. Por lo tanto, en *software* resulta mucho más sencillo el agregado de un filtro en la cadena de procesamiento.

El filtro (`sdr_en_fpga_filter`) se implementó de forma análoga al sistema original (`gr-isdbt`) en el cual se encuentra ubicado al comienzo del procesamiento (ver Figura 3.7 de la Sección 3.2 y los motivos explicados en dicha sección). A diferencia del caso de *software*, para incluir un filtro en el procesamiento sobre *hardware* fue necesario el diseño del filtro y validación de su comportamiento (tanto desde el punto de vista numérico como del filtrado de la señal). Para esto fue suficiente con el filtro FIR provisto por Nuand y la configuración de la respuesta al impulso mediante coeficientes y de esta forma se logró el filtrado de la señal sobre la cadena de procesamiento.

Bloque `interface` El bloque `interface` tiene una función de *gateway* ya que es responsable de la lectura de símbolos de los bloques de procesamiento y el pasaje de muestras hacia el bloque `fifo_writer`. Por lo tanto, el mismo debe respetar el protocolo de comunicaciones establecido con el `fifo_writer`. Se recuerda al lector que este implica que todas las muestras que se entreguen al `fifo_writer` serán válidas y corresponden a distintos símbolos (ver Sección 4.2.1).

Interfaz de Datos: repaso del Capítulo 4 Se recuerda al lector que los datos enviados hacia la PC se transmiten en mensajes de tamaño de 2 K *bytes* para el caso de USB 3.0 y no en particular, del tamaño N del símbolo OFDM. Teniendo en cuenta que la principal tarea a implementar será la detección de los símbolos OFDM, se señalizarán los mensajes enviados de manera de que cuando GNU Radio los reciba pueda detectar dónde está ubicado cada uno. Se tuvo en

cuenta que dentro de un mensaje, 16 *bytes* son de metadatos: 4 reservados, 8 de *timestamp* y 4 de banderas. Los símbolos OFDM serán transmitidos de manera continua (sin hacer *padding*) y en cada mensaje irá un metadato de 4 *bytes* (los correspondientes a las banderas) indicando si comienza un símbolo OFDM o no y en caso afirmativo dónde. Recordar que de esta forma se necesitarán enviar 16.13 mensajes para completar un símbolo OFDM. Para esto, fue necesario modificar al bloque `fifo_writer` para que señalizara los mensajes que contienen símbolos y la implementación de `messages2symbol` en GNU Radio para que a nivel de la PC se obtuvieran los símbolos nuevamente.

5.2.2. Bloque `sym_acquisition`

En la presente sección se describe el bloque principal de la arquitectura realizada: el bloque `sym_acquisition`. La función que debiera llevar a cabo el `sym_acquisition` tiene dos grandes aspectos: la sincronización temporal y el ajuste fraccional en frecuencia. Las mismas son idénticas a las que realiza el bloque `OFDM Sym Acquisition` de la solución actual (ver la Sección 3.2). Para resolver el problema planteado se definió una arquitectura típica basada en dos bloques principales: el de control `sym_acq_control` y el de procesamiento `sym_acq_process` y dos bloques de memoria: uno para almacenar datos en la entrada y otra para almacenar datos en la salida. En la Figura 5.2 se muestran los mismos y el sentido de los datos marcado con una flecha. El bloque `mem_ram_2_port_in` es el primer bloque de memoria (en amarillo), el bloque `sym_acq_control` (en naranja) es el de control, el `sym_acq_process` (en verde) es el bloque de procesamiento y por último en celeste el bloque de memoria `mem_ram_2_port_out`.

El funcionamiento del bloque será cíclico, trabajando básicamente en dos posibles estados: sincronizado o des-sincronizado. El caso sincronizado es análogo al caso régimen explicado en la Sección 2.1.2 y el des-sincronizado al transitorio. Con algunas diferencias: en ambas situaciones el `sym_acq_process` esperará a tener la cantidad de muestras necesarias para comenzar el procesamiento, procesará las mismas y entregará el resultado en sus puertos de salida de datos mediante el aviso de símbolo listo.

Para facilitar la implementación, se definió el bloque `sym_acq_control` para que realice el pasaje de muestras entre el procesamiento y las memorias. Además es responsable de las comunicaciones con el bloque aguas abajo, quien continúa con el procesamiento del símbolo. En el caso de la arquitectura planteada corresponde al bloque `interface`. Sin entrar en detalles, tanto en el caso sincronizado como en el des-sincronizado, el procesamiento constará básicamente de dos grandes pasos, el primero, en el cual se llevará a cabo la alineación temporal para lo cual se utilizará la estrategia planteada en la Sección 3.2. Y un segundo en el cual se corregirá el desfasaje fraccional en frecuencia rotando las muestras. Esto implicará acceder a los datos dos veces: la primera para el cálculo de la correlación para encontrar el prefijo cíclico (donde se deberán tener muestras que difieran en N índices) y la segunda para corregir y sacar las muestras procesadas.

Se utiliza la nomenclatura de memoria de entrada y salida haciendo referencia

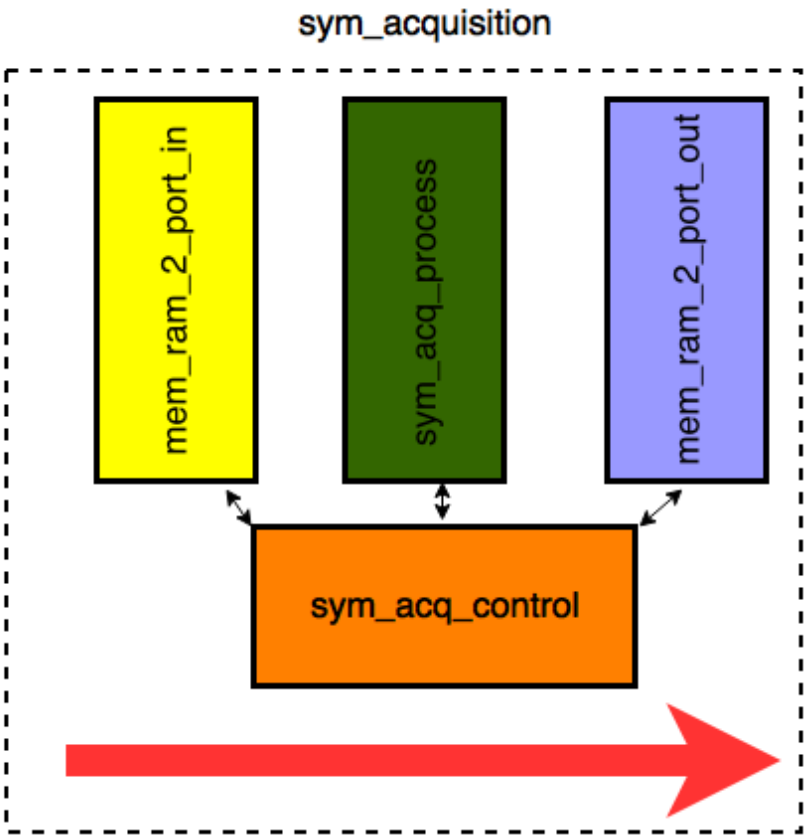


Figura 5.2: Arquitectura interna del bloque `sym_acquisition`.

a `mem_ram_2_port_in` y `mem_ram_2_port_out` respectivamente.

Las características de las memorias, esto es la cantidad de datos a almacenar, el tamaño de los mismos y con qué velocidad se pueden acceder, fueron fijados basándose en los requerimientos globales del sistema (ver la Sección 5.1.3). Como se profundizará en la Sección E.3.1, el procesamiento en algunas etapas deberá contar con $2N + L$ muestras. Este factor fue determinante para definir el tamaño mínimo de la memoria de entrada. Por otro lado, la arquitectura del `sym_acquisition` fue definida de manera que la memoria de entrada fuera utilizada tanto para escribir los datos que fueran llegando (cada dos períodos de reloj) como para acceder a los mismos cuando fuera necesario por el procesamiento. Este aspecto fue crítico ya que redujo ampliamente las posibilidades de memorias disponibles.

Al comienzo de la etapa de la definición de la arquitectura, se planteó la posibilidad de utilizar una memoria tipo *fifo* (*first in first out*) sin embargo esta fue descartada debido a cómo se llevaría adelante el procesamiento. En segunda instancia se consideró utilizar memorias ram de 4 puertos para lectura y escritura. No se pudo conseguir una implementación adecuada por lo que se optó por utilizar un conjunto de memorias ram de 2 puertos de entrada-salida de la biblioteca de Altera (`ram_2_port`). Como el procesamiento debería acceder a muestras que difieran en N índices se optó por utilizar varias unidades de memorias de N muestras.

Finalmente se utilizaron 4 memorias para contar con un margen frente al caso de $2N + L$ muestras. El análisis de tiempos para esta decisión será planteado en la Sección E.3.5. En la Sección E.3.3 se profundiza en las características y validación de este bloque.

En cuanto a la memoria de salida, su tamaño queda fijo dada la aplicación: la salida de este bloque corresponde a un símbolo. De esta forma se implementó una memoria ram de 2 puertos de N datos del mismo tipo que las unidades de la memoria de entrada. La posibilidad de contar con 2 puertos aumentó considerablemente la velocidad de escritura en la misma: redujo en la mitad su duración. Por más detalles sobre esta memoria se puede referir a la Sección E.3.6.

En este sentido, utilizando los tipos de memoria explicados anteriormente fue necesario contar con un bloque de control que se encargara de manejar los datos de manera simple y eficiente, tanto para la recepción de datos, como para su procesamiento y posterior entrega del símbolo procesado aguas abajo (hacia el bloque `interface`). Es importante que el lector entienda que el uso de memorias de dos puertos agiliza sustancialmente el acceso a las mismas pero que esto requiere de un manejo eficiente de las señales de control de dicha memoria.

En la memoria de entrada se guardan las muestras entregadas por el `sdr_en_fpga_filter` sin procesar cada dos periodos de reloj. El bloque de procesamiento hace uso de las mismas para cumplir su función. Esto implica el control, tanto de la escritura de la memoria de entrada, como de la lectura de la misma maximizando los tiempos de este último proceso. Por otro lado, el bloque `sym_acq_process` al terminar el procesamiento debe guardar el símbolo corregido en la memoria de salida, para que luego el bloque `interface` lo lea. En este sentido, el `sym_acq_control` también se encarga de la escritura y lectura de la memoria de salida.

Una de las ventajas de la arquitectura desarrollada para el bloque de control (`sym_acq_control`) es que es capaz de realizar cualquiera de las funciones anteriores de manera paralela (en caso de que sean compatibles). Esto fue posible gracias a la arquitectura de procesamientos en paralelo. En este sentido, de acuerdo a las funciones previamente descriptas el bloque está compuesto por un proceso en *vhdl* para llevar a cabo cada una de ellas. En la Sección E.3.4 se profundiza en el diseño de este bloque, uno de los aspectos presentados es el funcionamiento de cada uno de los procesos.

Consideraciones sobre el procesamiento en el FPGA El procesamiento del bloque, llevado a cabo por el bloque `sym_acq_process`, se realizó a imagen y semejanza del bloque de *software* `OFDM Sym Acquisition` presentado en la Sección 3.2 cuyo algoritmo se expuso en la Sección 2.1.2. El procesamiento se implementó en el bloque `sym_acq_process`. Para el mismo se tuvo en cuenta las limitaciones de la implementación sobre *hardware* vistas en la Sección 5.1.

Tal como se mencionó en la Sección 2.1.2, el sincronismo temporal con los símbolos se realiza mediante la correlación presente entre los prefijos cíclicos de los mismos. Para el cálculo se utilizó la misma función de correlación (Λ), correspondiente al algoritmo de máxima verosimilitud presentado en la Sección 2.1.2 (con N la cantidad de muestras en un símbolo y L la cantidad de muestras del prefijo

cíclico (CP)):

$$\Lambda(\theta, \epsilon) = |\gamma(\theta)| \cos(2\pi\epsilon + \angle\gamma(\theta)) - \rho\Phi(\theta), \tag{5.1}$$

con:

$$\gamma(m) = \sum_{k=m}^{m+L-1} r[k]r^*[k+N],$$

$$\Phi(m) = \sum_{k=m}^{m+L-1} \frac{|r[k]|^2 + |r[k+N]|^2}{2},$$

y $\rho = \sigma_s^2/(\sigma_s^2 + \sigma_n^2) = SNR/(1+SNR)$.

Si esta fuera la única función del bloque, una vez que se encuentre la ubicación de los símbolos, solo se debería entregar a la salida cada símbolo inalterado (descartando el CP).

Esta función presenta la condición de que debe ser evaluada en un rango relativamente grande ($N = 8192$) y en particular cada valor corresponde a una operación sobre un conjunto de muestras de largo $L = 2048$. La dificultad en este caso es la de realizar el cálculo en el menor tiempo posible, de forma de lograr el sincronismo con los símbolos. Este problema está presente tanto al momento de la primer búsqueda sobre todo el rango así como durante el procesamiento en régimen.

La primer búsqueda implica que la ventana de observación debe ser grande ($2N+L$) y por lo tanto la memoria de la entrada debe tener suficiente espacio para alojar las muestras de la misma y las siguientes que se reciban, hasta que termine el procesamiento (en particular, los recursos de memoria podrían ser limitados a cierto valor máximo dado por el FPGA). Durante el procesamiento en régimen, el cálculo de la correlación también deberá ser rápido debido a que se debe verificar nuevamente la presencia del CP y realizar la lectura de cada símbolo a la misma cadencia con la que se reciben.

Para solucionar este problema, se diseñó un algoritmo recursivo para lograr la sincronización mediante una función que implica el menor tiempo posible de cálculo de la correlación. El mismo permite calcular los valores de la función Λ a una cadencia igual a la frecuencia de muestreo, es decir, se calcula un nuevo valor de Λ por cada nueva muestra. El cálculo recursivo implica realizar una doble lectura de las muestras de la memoria de entrada. Esto es debido a que Λ se calcula en función de γ y Φ, los cuales corresponden a sumatorias recursivas. Cada elemento de las funciones γ y Φ debe ser hallado dos veces, ya que en un momento se utiliza como nuevo término a agregar a la suma y luego es necesario restar su valor debido a que ya no debe integrar la misma. De esta forma se evita el uso de *buffers* de gran tamaño que almacenen todos los sumandos o la alternativa de volver a calcular toda la suma cada vez. Si fuera el caso, se deberían almacenar $L = 2048$ sumandos con buena precisión generando un gran consumo de memoria dentro del bloque, el mismo sería del orden de consumo de la memoria de entrada.

Se invita al lector a repasar la Tabla 5.1 y analizar si aún quedaría espacio para la posible implementación de esta memoria.

El algoritmo requiere almacenar el resultado de la operación y algunas funciones intermedias en cadenas de registros tipo *fifo*, durante 4 períodos de reloj como máximo (por lo tanto cadenas de 4 elementos como máximo). Esto se debe a que si bien las componentes de la función Λ: γ y Φ son recursivas, para armar función Λ es necesario el valor de $|\gamma|$ junto a Φ. Por lo tanto se debe registrar el valor de Φ hasta el cálculo de $|\gamma|$ (pueden ser necesarios dos períodos de reloj para ello). Además es necesario que a ambas se le sumen los términos nuevos y se le reste los viejos de forma alterna entre distintos períodos de reloj. Por lo tanto, debido a que solo se registran señales durante pocos períodos de reloj la cantidad de registros auxiliares necesarios es poca. Es interesante hacer el ejercicio de resolver este problema en *software* en cualquier lenguaje de alto nivel. Para una computadora realizar esta cuenta es relativamente sencillo y no requiere todas estas consideraciones.

Las funciones Λ, γ y Φ se deben calcular con buena precisión, por lo tanto es necesario utilizar una gran cantidad de *bits* para la representación. Esto se debe a que es deseable mantener el error acotado dada la naturaleza del cálculo (recursivo). El único error que se genera es en el cálculo de $|\gamma|$ y el mismo no se acumula. Otra causa es que el resultado del cálculo intermedio de la correlación correspondiente a $\gamma(\theta)$ (siendo $\theta = \widehat{\theta}_{EMV}$, el índice correspondiente al máximo de Λ), será utilizado en la etapa siguiente para realizar la corrección en frecuencia.

La segunda función del bloque corresponde a la corrección en frecuencia, que a diferencia de la función de sincronismo, deberá modificar cada símbolo encontrado desde el punto de vista numérico. La misma se realizará mediante la aplicación de una rotación de fase creciente a lo largo de las muestras del símbolo (el incremento de la fase a aplicar entre muestras consecutivas es constante). El incremento de fase a acumular, se obtiene en función de las muestras del prefijo cíclico y las de su copia, lo cual corresponde al valor $-\angle\gamma(\theta)/N$. Por lo tanto, la fase de la rotación a aplicar deberá cubrir con buena precisión todo el rango de $[0, 2\pi]$, debido a que el incremento de la misma implica cubrir N valores de fase entre $[0, -\angle\gamma(\theta))$, siendo el valor de $\angle\gamma(\theta)$ arbitrario.

Para que el `sym_acq_process` pueda realizar la corrección en frecuencia, fue necesario estudiar un algoritmo que permita realizar las rotaciones de fases con poca cantidad de recursos, poca latencia y en lo posible mantener la precisión del resultado en valores aceptables. El algoritmo utilizado se llama CORDIC (*COordinate Rotation DIgital Computer*)[28], en la Sección E.3.5.3 se presentará en detalle su funcionamiento e implementación sobre el FPGA.

La corrección es realizada por el bloque `sym_acq_derote`. El mismo aguarda por el valor de $\gamma(\theta)$ y calcula su fase para poder realizar la corrección, dando el aviso cuando se encuentre pronto para ello. Ver la Sección E.3.5.3 por una descripción detallada del `sym_acq_derote`. Luego el bloque halla el valor de la fase de $\gamma(\theta)$ para determinar la rotación a aplicar sobre cada muestra. El bloque instancia dos bloques `cordic`, los cuales se encargan de realizar la rotación de las muestras de cada puerto (ver la Sección E.3.5.3). Los mismos pertenecen a la librería de Nuand del proyecto bladeRF.

En la Sección E.3.5 se profundiza en el diseño del bloque `sym_acq_process`.

Validación de los algoritmos implementados Los algoritmos de correlación y de rotación de fases que se implementaron en *hardware* fueron analizados en detalle desde el punto de vista numérico mediante el uso de la herramienta de cálculo Octave, durante las etapas de diseño, implementación y validación. En ambos casos se utilizaron señales de referencia como entrada en simulaciones mediante *test-benchs*. Los resultados correspondientes fueron comparados contra el procesamiento del bloque de *software* y los análogos implementados sobre Octave. En el caso particular de Octave, generalmente se realizó la comparación contra un algoritmo ideal desde el punto de vista numérico (únicamente limitado por la herramienta) y uno que emula el comportamiento numérico de procesamiento sobre el *hardware*. Esta comparación permite analizar qué etapa del algoritmo sobre *hardware* es la limitante de su comportamiento numérico y determinar posibles mejoras. La validación de los algoritmos se realizó en función del análisis mediante las comparaciones descritas.

Parámetros de la solución propuesta La aplicación cuenta con ciertos parámetros ya que se debe contemplar la opción de elegir los distintos largos del prefijo cíclico, modos y SNR para los bloques internos. Para esto es necesario incluir puertos de entrada para tales efectos asociados a dichas funciones. Las tres entradas mencionadas provendrán en última instancia de la PC.

En el cuadro 5.2 se muestran los posibles valores que puede tomar el tamaño del Prefijo Cíclico (CP) de acuerdo al parámetro `cp`. El valor 2 corresponde a un CP de 1/4, el 3 a un CP de 1/8 y así sucesivamente ($CP = \frac{1}{2^{cp}}$). Este dato será obtenido de la PC. Para ello se modificó el `gr-osmosdr` como se expuso en la Sección 4.3.1. La salida de un *bit* del bus GPIO del NIOS la cual permite elegir el CP en 1/4 o 1/8, luego con el dato del CP se conectó al puerto de entrada del bloque con el mismo nombre.

CP			
2	3	4	5
1/4	1/8	1/16	1/32

Tabla 5.2: Prefijo cíclico (CP)

El modo de funcionamiento puede ser de $N = 2$K, 4K u 8K muestras. El mismo se dejó fija para la implementación ya que para Uruguay solo se utiliza el modo 8K.

Por último, el SNR (relación señal a ruido) se definió fijo también.

5.2.3. Ventajas y limitantes de la solución

Se implementó de manera exitosa una solución al problema planteado en el FPGA de la bladeRF la cual presenta algunas ventajas y limitaciones que se describirán a continuación.

Como primer ventaja de la solución se destaca el conocimiento generado durante toda la experiencia haciendo énfasis en que se resolvió un problema de índole numérico en un FPGA.

En segundo lugar, la arquitectura diseñada permite la posterior implementación de nuevos bloques siempre y cuando los recursos lo permitan y se respete el protocolo definido. En particular, la implementación de una FFT (siguiente bloque en la demodulación) sería posible ya que se estima un consumo menor al de memoria restante disponible.

Otra ventaja de la solución propuesta es que el procesamiento de la señal de DTV continúa de manera transparente para el resto de los bloques en *software* (`gr-isdbt`).

Además se implementó una solución que cuenta con la posibilidad de variar el CP de la señal recibida mientras que el N (cantidad de muestras en el símbolo) está limitado al valor 8192 ya que es el único que se utiliza en Uruguay así como también el SNR.

Mediante el diseño realizado sobre el FPGA se concluye que para realizar operaciones sobre el mismo que requieran buena precisión en los resultados, no solo es suficiente con la elección del algoritmo y un adecuado dimensionamiento del mismo sino que hay que tener en cuenta los recursos disponibles en el FPGA y limitar la implementación.

En otras palabras, por más que se utilicen varios *bits* para la representación o un gran número de pasos en los algoritmos iterativos, se pueden requerir ciertos recursos, como memoria para la implementación de bloques elementales o de lógica que pueden llegar a superar la disponibilidad del FPGA.

Por otra parte, al haber trabajado en un proyecto ya armado, existen ciertas limitaciones referentes a las interfaces entre las distintas partes de la cadena de datos que pueden limitar la representación y la cadencia de salida de los bloques de procesamiento. Por ejemplo la cantidad de *bits* de cada muestra (16 para I y 16 para Q). Se podría, por ejemplo, considerar la posibilidad de enviar muestras de 32 *bits* para I y 32 *bits* para Q. Para esto habría que hacer un análisis más profundo de la interfaz hacia la PC de forma de modificarla para implementar esta mejora.

Se debe tener en cuenta que el desarrollo de este proyecto está limitado al lenguaje *vhdl* y al FPGA presente en la placa y por lo tanto a la plataforma de desarrollo definida por Nuand, en este caso Quartus 15.0. También se considera que el desarrollo del diseño está condicionado a otras herramientas desarrollo utilizadas como el simulador (ModelSim) y el *software* necesario para pruebas numéricas (Octave).

Finalmente, a pesar de estos elementos, se implementó una solución adecuada, en la cual aún quedan recursos disponibles para otros bloques y permite transmitir las muestras generadas en el FPGA hacia GNU Radio sin inconvenientes, a pesar de que sea a una cadencia diferente a la frecuencia de muestreo.

5.3. Conclusiones

En este capítulo se presentó el diseño realizado sobre el FPGA del SDR bladeRF. El mismo consistió en la definición de una arquitectura sobre el FPGA de forma de poder realizar la primer etapa de sincronización de un receptor OFDM de Televisión Digital según el estándar ISDB-T. En este sentido, las señales OFDM procesadas corresponden a esta norma. Se definió trabajar en el FPGA a la frecuencia de muestreo definida por la misma correspondiente al valor $f_s = 512/63$ MHz $\approx 8{,}13$ MHz. Por lo tanto, el procesamiento de la señal fue realizado a dicha frecuencia para utilizar el mismo reloj de muestreo.

La arquitectura consistió en una cadena de procesamiento de tres bloques ubicados entre los bloques `rx_iq_correction` y el `fifo_writer`, los cuales definieron el formato de entrada y salida de datos. El primer bloque, denominado `sdr_en_fpga_filter`, se implementó para el filtrado de la señal de entrada de forma de atenuar la señal en los canales adyacentes. El segundo, el `sym_acquisition`, se encarga de llevar a cabo la sincronización temporal con los símbolos en la entrada y de la corrección fraccional en frecuencia de los mismos. Para la primer tarea, utiliza un algoritmo recursivo correspondiente a la función de correlación. Este algoritmo implicó la definición del mínimo necesario de memoria para la recepción de muestras en la entrada y de los requerimientos de lecturas sobre la misma. El mismo permite la sincronización con todos los símbolos recibidos. Para llevar a cabo la segunda función se implementó un algoritmo de rotación de las muestras de cada símbolo. La corrección se realizó aplicando una rotación a las muestras de cada símbolo mediante una fase creciente, la cual se realiza mediante el algoritmo iterativo CORDIC. El diseño del mismo implicó la especificación de una arquitectura interna al bloque basada en funcionalidades. La misma consiste en bloques de memoria de entrada y salida, en un bloque de control y en uno de procesamiento. El tercer bloque, `interface`, escribe el resultado del procesamiento en el `fifo_writer` sobre el cual continúa la cadena.

Para la implementación de la solución se tuvo en cuenta las limitantes intrínsecas de la plataforma de desarrollo: recursos y representación numérica limitados. Esto tuvo como consecuencia que se definiera un algoritmo lo más eficiente posible para utilizar la menor cantidad de recursos (de memoria sobre todo) y para que el resultado numérico fuera lo más exacto posible. Además, el proyecto tuvo como limitante la herramienta de desarrollo en *vhdl*: Quartus definida directamente por Altera, fabricante del FPGA utilizado, y cuya versión fue fijada por Nuand, fabricante de la bladeRF.

Por otro lado, para el diseño, desarrollo y la validación se tuvo más libertades optando por ModelSim que facilitó fuertemente la prueba de los bloques mediante el uso de *test-benches* y Octave que ayudó en los aspectos numéricos. En la primer sección del capítulo se discuten en mayor profundidad los recursos disponibles, las limitantes y los requisitos propuestos. En base a los puntos mencionados anteriormente se concluye que el mecanismo de diseño e implementación sobre el FPGA permitió resolver el problema propuesto de forma exitosa.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Pruebas y Validación

La validación del bloque `sym_acquisition` corresponde a la verificación del funcionamiento del mismo tanto en simulación como en *hardware*. Se le recuerda al lector que las funciones del bloque corresponden a la sincronización temporal y a la corrección fraccional en frecuencia de símbolos OFDM correspondientes a la norma ISDB-T de DTV. En este sentido, la prueba se dividió en dos etapas: la primera corresponde a la validación de la sincronización temporal y la segunda al sistema completo, incluyendo la corrección fraccional en frecuencia. Para la primera, se consideró el cálculo de la función de correlación Λ que permite la sincronización con los símbolos de la señal.

En las simulaciones se utilizaron *test-benchs* en VHDL ejecutados por la herramienta ModelSim. Los mismos permitieron realizar lecturas de señales en el disco, la simulación y la captura de los resultados finales e intermedios deseados. Las pruebas en *hardware* se realizaron mediante la recepción de señales generadas en la placa bladeRF.

Las entradas utilizadas en simulación corresponden a capturas de referencia almacenadas en la PC y leídas mediante el *test-bench*. En el caso de las pruebas en *hardware*, se utilizaron dos tipos de señales. El primer tipo corresponde a señales generadas mediante la lectura cíclica de una memoria ram conteniendo parte de una señal de referencia. Y el segundo, a capturas desde la interfaz de aire.

La verificación en cada caso consistió en la comparación de los resultados obtenidos con los esperados mediante el uso de *scripts* de Octave y *flowgraphs* de GNU Radio. En general, se verificó la integridad de los símbolos de salida, la integridad de la función Λ correspondiente a la correlación y los resultados del procesamiento de la salida del resto de la cadena del receptor `gr-isdbt`. En este último punto se verificó en particular la constelación recibida para señales relativamente cortas (menores a 1000 símbolos), el chequeo de mensajes TMCC realizado por el módulo y la correcta decodificación del vídeo correspondiente para señales de mayor tamaño (al menos 10000 símbolos).

Para la validación de los resultados obtenidos se definieron ciertos criterios de éxito explicados a continuación. En el caso de la verificación de Λ, se comparó muestra a muestra (la salida de cierta prueba y la esperada obtenida de un *script*) y se definió como criterio de éxito que fueran idénticas. Para la verificación de

símbolos (sin la rotación) el criterio fue el mismo. En el caso de la verificación mediante la constelación se tomó como criterio de éxito que se pudieran identificar las portadoras de la misma. Para la validación mediante vídeo, se tomó como criterio de éxito que tuviera secciones de vídeo y de audio continuas de por lo menos 5 segundos. Este criterio se definió de esta forma ya que no siempre se contó con una señal de aire de buena calidad. Para los casos en que se contara con una buena señal se considera 1 minuto de vídeo y audio.

En este sentido, en esta etapa se verificaron los bloques internos de la arquitectura: el `sym_acq_control`, `mem_ram_2_port_in`, `mem_ram_2_port_out` y `sym_acq_process` como estaba previsto.

Señales de entrada de referencia Las señales de entrada para las pruebas fueron obtenidas mediante el sistema original realizando capturas con la placa bladeRF y guardando la salida del bloque `Osmocom Source` luego del filtrado pasa bajos. Las capturas se realizaron en las mejores condiciones de recepción posibles para cada canal de DTV de Montevideo, Uruguay. De esta forma para cada una de ellas se consiguió una señal que una vez procesada por el módulo `gr-isdbt` sea posible observar la constelación y finalmente obtener la señal de vídeo. Las señales fueron almacenadas en el formato `gr_complex` utilizado en GNU Radio.

Luego, para la obtención de señales de prueba de referencia, se utilizaron las anteriores para extraer ventanas de distinto largo. Los largos utilizados fueron los siguientes: $2N + L$ muestras correspondientes a un ventana de observación que incluye un símbolo y su CP, ventanas de largo 2^{15} muestras (incluyendo 6 símbolos) y señales del orden 1000 y 10000 símbolos (incluyendo su CP).

Debido a que estas últimas se utilizarían como entrada de *test-benchs* y como contenido inicial de una memoria ram, fue necesario realizar una conversión de formato, en particular se multiplicaron las mismas por el factor 2048 (para deshacer la normalización que realiza el bloque `Osmocom Source`, ver la Sección 3.4.2). En el primer caso se convirtieron al formato de texto, representando las mismas en números enteros. En el segundo también a texto pero al formato *MIF* (*Memory Initialization File*) para inicializar la memoria ram. Las salidas de los *test-benchs*, en algunos casos se tuvieron que convertir al formato `gr_complex` para utilizarse como entradas de GNU Radio. Estas conversiones fueron realizadas mediante *scripts* de Octave, los cuales permitieron intercambiar fácilmente entre los distintos formatos. Los *scripts* tuvieron la limitante que para los casos de las señales de mayor tamaño implicaron un tiempo considerable para realizar la conversión (del orden de varias horas).

6.1. Pruebas de sincronización temporal

En la presente prueba, como se explicó en la introducción, se verificó la correcta implementación de la sincronización temporal. En este sentido, la corrección fraccional en frecuencia no fue parte de esta prueba. La validación se llevó a cabo mediante la verificación de Λ y de los símbolos de salida.

6.1.1. Simulación

Se definió un *test-bench* con una instancia del `sym_acquisition` (sin la implementación de la corrección en frecuencia) y un bloque generador de datos llamado `generator` implementado para esta prueba.

La misma consistió en la validación del sincronismo temporal mediante una entrada constante que se repetía indefinidamente. Para ello, se implementó el bloque `generator` que está compuesto por una memoria ram de dos puertos de largo 32768 de muestras. La misma se configuró para inicializarse con una señal de referencia. Para generar la etapa de transitorio, en una primera instancia, se definió que entregara muestras correspondientes a un remanente de datos, un prefijo cíclico y un símbolo y en segunda instancia, únicamente el símbolo y el prefijo cíclico. Este último comportamiento se repitió indefinidamente.

Con el objetivo de clarificarle el entorno de validación al lector, se aclara que en este punto no se utilizó el filtro en la entrada ni la señalización de los datos ya que estos se guardaron directamente de la salida del `sym_acquisition`.

Resultados Se verificó el funcionamiento de la sincronización temporal validando la integridad de la salida de los símbolos y de la función Λ utilizando los criterios definidos en la introducción. Mediante los *scripts* se verificó que la señal de salida del entorno de prueba era idéntica a la esperada.

6.1.2. Pruebas en Hardware

Se implementó en el FPGA la misma arquitectura definida para la simulación además del bloque `interface` y el `fifo_writer` (se incluyeron obviamente el resto de los bloques correspondientes a la arquitectura original). Se obtuvieron los datos del `Osmocom Source` y mediante los mismos *scripts* que en el caso anterior, se verificaron. Se pudo corroborar la correcta sincronización temporal en la implementación en *hardware*.

6.2. Pruebas del sistema completo

En esta sección se describen las pruebas realizadas del sistema completo. En primer lugar, se presentan las simulaciones realizadas de la arquitectura completa mediante un *test-bench* que incluye los siguientes bloques: `sym_acquisition`, `interface` y `fifo_writer` (sin utilizar el filtro). En segundo lugar, las pruebas del sistema completo sobre *hardware*, conteniendo la arquitectura completa, los bloques explicados anteriormente y el filtro `sdr_en_fpga_filter`.

6.2.1. Simulación

Se modificó el *test-bench* de la Sección 6.1.1, instanciando el `sym_acquisition` con la corrección fraccional en frecuencia. La entrada de la simulación consistió en una señal de 1000 símbolos. Los mismos fueron procesados por la arquitectura.

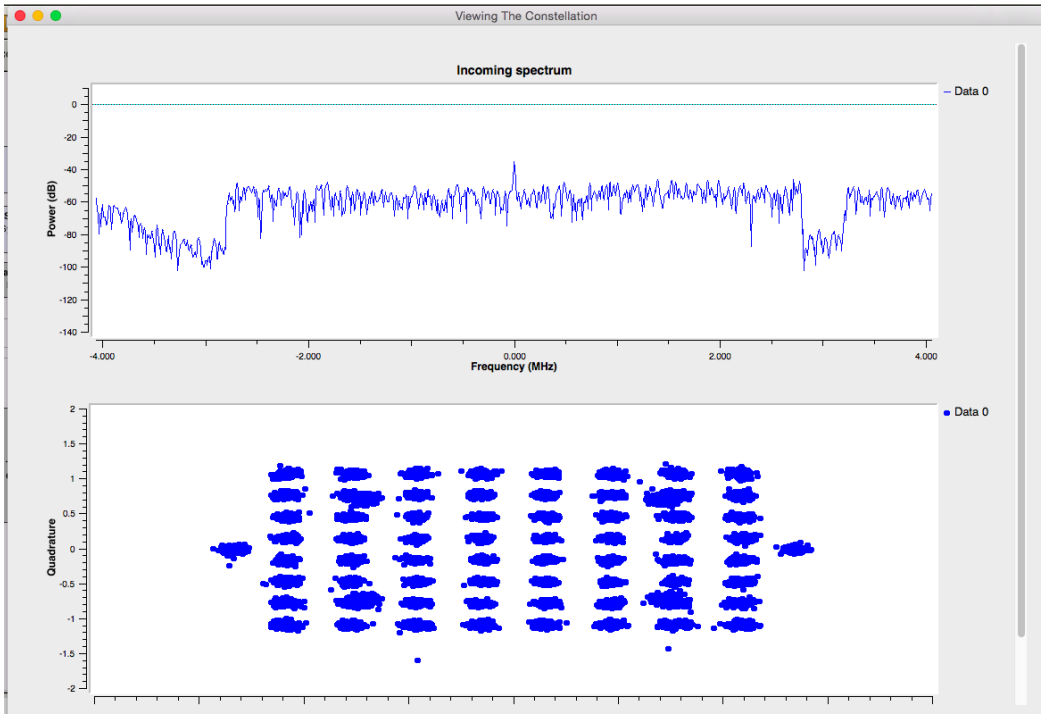


Figura 6.1: Constelación de GNU Radio Companion, obtenida de la simulación del sistema completo (sin filtro).

En esta prueba no se utilizó el filtro pasa bajos para que las entradas al bloque `sym_acquisition` sean inalteradas.

Resultados En primera instancia, se verificó nuevamente el funcionamiento de la sincronización temporal validando la integridad de los símbolos previo a la rotación de los mismos y de la función Λ, utilizando los criterios definidos en la introducción. Luego, se guardó la señal conteniendo los símbolos rotados correspondientes a la salida bloque `interface`. Esta fue convertida al formato `gr_complex` y se utilizó como entrada en GNU Radio. En la Figura 6.1 se observa el contenido en frecuencia y la constelación asociada. Como se observa en la misma, en la constelación obtenida se visualiza claramente los esquemas de modulación utilizados verificando así que el procesamiento del bloque `sym_acquisition` fue correcto.

6.2.2. Pruebas en Hardware

Se implementó el sistema definido en la sección anterior en el FPGA de la bladeRF. Un elemento crítico de esta etapa fue la obtención de una buena señal de DTV en la entrada de la placa. Una vez superada esta etapa, se realizaron múltiples capturas de todos los canales de televisión digital de Montevideo, Uruguay verificando el correcto funcionamiento del sistema según los criterios establecidos en la introducción.

En este sentido, se validó el sistema íntegro: el filtro `sdr_en_fpga_filter`, el bloque de procesamiento `sym_acquisition`, el bloque de interfaz `interface`, el `fifo_writer` modificado. En GNU Radio, se validó el `gr-osmosdr` y el bloque `messages2symbol`. Del primero se validó la posibilidad de configurar el prefijo cíclico.

En la Figura 6.2 se presenta un *flowgraph* del sistema original, idéntico al explicado en la Sección 3.2. Se pretende que el lector pueda verificar la diferencia de los dos sistemas, en la Figura 6.3 se presenta el *flowgraph* modificado para la implementación de esta prueba y el que deberá utilizarse para el uso de esta implementación.

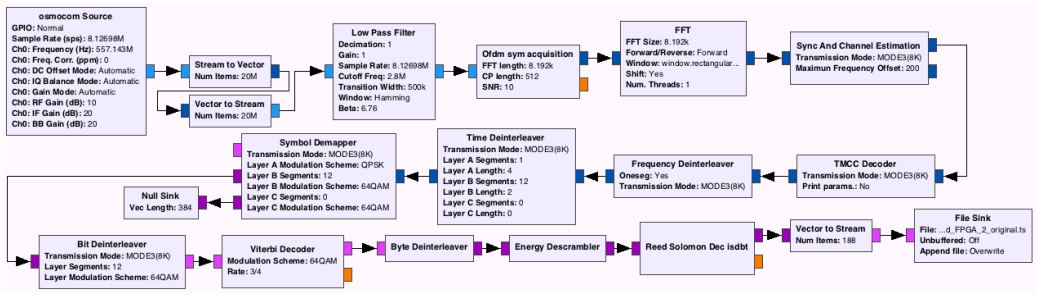


Figura 6.2: *Flowgraph* de GNU Radio Companion del receptor `gr-isdbt` sin modificar.

6.3. Conclusiones de Pruebas del Sistema

Para la validación del sistema implementado se definieron dos etapas, una primera en la cual se verificó la sincronización temporal y una final del sistema completo. Para cada etapa, se validó en primer lugar el sistema mediante el uso de *test-benchs* en el ModelSim y en segundo lugar en el FPGA.

Fue necesario contar con varias señales de referencia que supiéramos tenían un comportamiento casi ideal frente al sistema `gr-isdbt`. Además se definieron varios *scripts* en Octave para la conversión de las mismas entre las herramientas de prueba utilizadas: Octave, GNU Radio Companion y ModelSim.

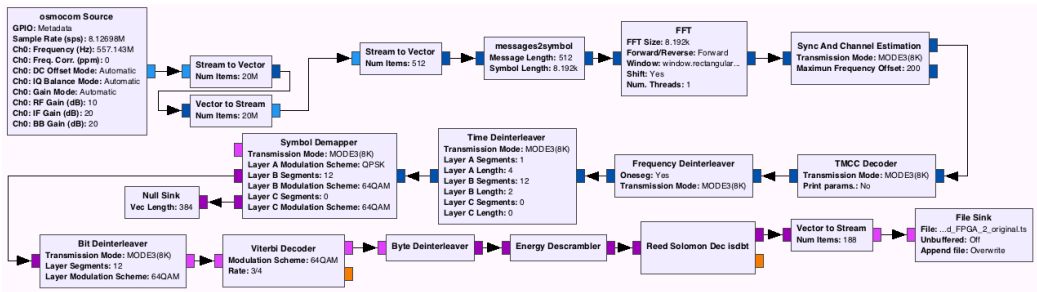


Figura 6.3: *Flowgraph* de GNU Radio Companion del receptor `gr-isdbt` con funcionamiento implementado en *hardware*.

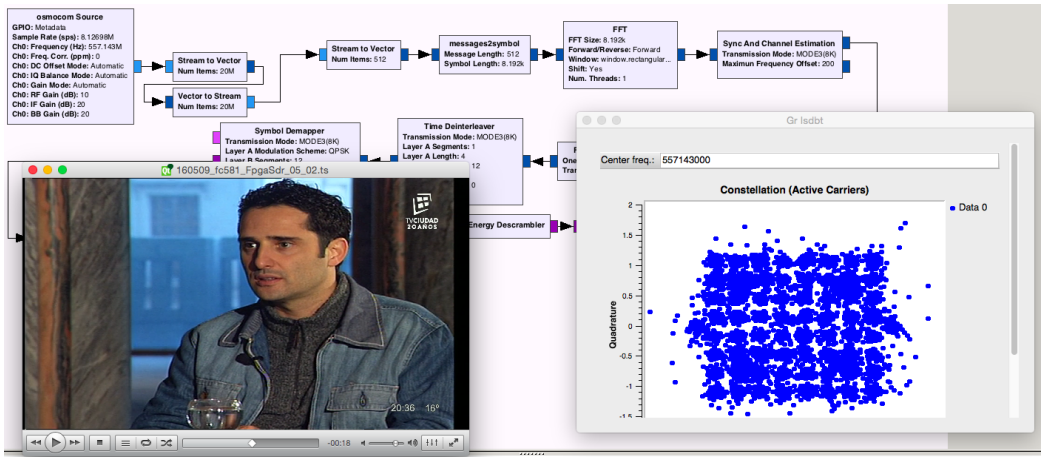


Figura 6.4: *Flowgraph* de GNU Radio Companion del receptor `gr-isdbt` con funcionamiento implementado en *hardware*, constelación y vídeo.

Por otro lado, fue necesaria la implementación de varios *scripts* en Octave que simulaban la respuesta esperada los cuales se validaron con el sistema `gr-isdbt` y sirvieron como referencia para la presente etapa.

En la primer etapa de validación, se implementó un bloque generador de datos que entregaba la misma señal de forma indefinida. Se verificó comparando contra la salida de un *script* la integridad de los símbolos y la función de correlación Λ se obtuvo un resultado exitoso tanto para la simulación como para la prueba en el *hardware*.

En la segunda etapa, se probó el sistema completo. Esto incluye tanto el filtro en la entrada del `sym_acquisition` como el bloque de interfaz y el `fifo_writer` modificado para la simulación y todos los bloques para la prueba en el FPGA. Se verificó que la constelación cumplía con los criterios definidos en la introducción al igual que la señal de vídeo. En la Figura 6.4 se presenta una captura a modo de referencia del entorno para esta prueba.

Capítulo 7

Conclusiones finales

7.1. Síntesis

El proyecto consistió en la implementación en un FPGA de la primer etapa de sincronismo de un receptor OFDM para la recepción de señales de Televisión Digital según la norma ISDB-T sobre un sistema de radio definido por *software*. Esto implicó también la definición e implementación de un protocolo de comunicaciones para el envío de símbolos OFDM del FPGA hacia la PC, en la cual corre el software del sistema y se realiza el resto del procesamiento.

El documento se dividió en siete capítulos: el primero introductorio, el segundo donde se presentaron los elementos teóricos necesarios para la implementación: el estándar ISDB-T y los sistemas SDR. En el tercer capítulo se expuso la solución actual para la recepción de DTV mediante la placa bladeRF y GNU Radio, en particular el módulo `gr-isdbt` y la interfaz bladeRF - GNU Radio. En los capítulos cuarto y quinto se presentó la solución actual, en el primero lo correspondiente a las comunicaciones y en el otro a la implementación en *hardware*. En el capítulo sexto se expusieron las pruebas llevadas a cabo para la validación de la solución propuesta. Finalmente, en el presente capítulo se realizan las conclusiones finales del proyecto.

Marco teórico En el capítulo Capítulo 2 se presentó el estándar ISDB-T y las características de la etapa de sincronismo en primer lugar. En particular, se focalizó en aquellos aspectos importantes para la implementación en *hardware*. En este sentido se presentaron algunos de los parámetros y características de la modulación OFDM utilizada por la norma: ancho de banda, frecuencia de muestreo y prefijo cíclico. También se presentaron los parámetros que utilizan los canales de DTV de Uruguay, los cuales fueron utilizados para configurar la recepción en las etapas posteriores.

Luego, en segundo lugar, se introdujo el concepto de radios definidas por *software* (SDR por su sigla en inglés). Los mismos son sistemas de comunicación por radio implementados en *software*. En este sentido, tienen por objetivo implementar la mayor cantidad de elementos que típicamente estarían en *hardware*, en *software*.

Se presentó GNU Radio, la plataforma de desarrollo de los SDR en la PC.

En el capítulo Capítulo 3 se presentó la solución actual, la cual incluye la placa bladeRF y los módulos `gr-isdbt` y `gr-osmosdr` de GNU Radio. En primer lugar se realizó la descripción de la plataforma de *hardware* utilizada, la placa bladeRF de Nuand. Es una plataforma abierta, de costo medio (alrededor de 650 USD), con un activo foro de consultas e intercambio, diseñada con el objetivo de permitirle a estudiantes, radioaficionados o profesionales del área de radio frecuencia explorar el mundo de las comunicaciones inalámbricas proporcionando una plataforma de desarrollo versátil. La misma cumple con los requerimientos necesarios para la aplicación: frecuencia de muestreo y ancho de banda. Cuenta con un FPGA responsable de la configuración de los integrados de la placa y que permite la implementación *on-board* de procesamiento.

Se estudió el módulo `gr-isdbt` de GNU Radio que implementa un receptor *full-seg* de Televisión Digital del estándar ISDB-T implementado en la Facultad de Ingeniería de la Universidad de la República del Uruguay. Se estudió el módulo `gr-isdbt`, elemento de partida para la implementación en *hardware*. Además, fue el sistema utilizado para el procesamiento posterior al bloque en *hardware* correspondiente a las siguientes etapas de procesamiento, en particular de la decodificación de la señal de vídeo. Se presentaron los bloques del mismo, profundizando en el primero cuya función es la de sincronización ya que fue el que se implementó en *hardware*.

Luego se presentó la interfaz bladeRF - GNU Radio correspondiente a la solución actual. En primer lugar se vio la interfaz por defecto y se destacó la limitante que presenta para la aplicación a desarrollar: envío de muestras en paquetes de tamaño fijo diferente al tamaño de los símbolos. Se estudió el flujo de datos sobre la interfaz y el uso de metadatos sobre los paquetes que envía la placa hacia la PC.

Solución Propuesta: Comunicaciones La solución propuesta para las comunicaciones incluyó la implementación de un protocolo que señalice símbolos y de un bloque en GNU Radio que los decodifique.

Para llevar a cabo esto, fue necesario modificar un bloque en la cadena de recepción de datos en el FPGA: el `fifo_writer`. Se validó el bloque de forma individual mediante el uso de la herramienta de simulación del Quartus y la realización de distintas pruebas en el *hardware*. En este sentido se concluyó que la herramienta de simulación de Quartus presentaba varias limitaciones, como el largo de la simulación y por lo tanto dificultó la etapa de verificación.

Se modificó el módulo `gr-osmosdr` para poder configurar funciones sobre el FPGA de la placa bladeRF, en particular, el uso de metadatos. Se verificó que esta función hubiera quedado implementada realizando distintas capturas. A su vez, se corroboró, que no se hubiera afectado el comportamiento integral del mismo. Esta modificación fue utilizada posteriormente para configurar otras funciones. Por ejemplo, la asignación de señales en el FPGA correspondientes a parámetros del procesamiento desde GNU Radio mediante el bloque `Osmocom Source`.

Se implementó el bloque `messages2symbol` sobre GNU Radio para interpretar los metadatos y armar los vectores que se generan en el *hardware*. Este bloque tam-

bién permite realizar otras funciones asociadas a los metadatos, como por ejemplo la verificación de *timestamps*. Se validó el funcionamiento del mismo generando distintas entradas con diversas condiciones para poder cubrir la mayor cantidad de situaciones posibles.

En esta etapa se realizaron pruebas de integración de la modificación de la interfaz junto al módulo `gr-isdbt`. Se verificó que la implementación realizada funciona en condiciones similares a la aplicación y en particular, que la misma resulta transparente para el procesamiento del módulo.

Finalmente, se propusieron otras posibilidades del uso de metadatos, como el envío de información extra asociada al procesamiento o a la representación.

Solución Propuesta: *hardware* En el quinto capítulo se presentó el diseño realizado en el FPGA de la placa bladeRF que resuelve el problema planteado.

Se comenzó por definir las limitantes del *hardware* es decir las características del *hardware* que condicionaron el diseño. Se mencionó la limitante de la interfaz de radio provocada por la dependencia con el transmisor de la señal de DTV y las condiciones de recepción, que entre otras razones depende de la placa bladeRF. Además se vio la condición consecuente de la interfaz *hardware*-PC ya solucionada en el Capítulo 3 mediante la implementación de un protocolo de comunicaciones. Se vio que la pérdida de muestras sería otro problema a resolver así como también la adaptación a la arquitectura interna del FPGA ya existente en el proyecto original que entre otros elementos fijaría una ubicación para la implementación así como también una cantidad limitada de recursos, entre ellos de memoria.

El capítulo continuó definiendo una serie de requisitos que debiera cumplir la solución propuesta los cuales se consideraron durante el diseño. Luego se presentó la arquitectura implementada en el FPGA haciendo énfasis en los desafíos y en las soluciones encontradas por el trabajo en una plataforma de *hardwre*. La arquitectura propiamente del FPGA consistió en una cadena de procesamiento de tres bloques ubicados entre los bloques `rx_iq_correction` y el `fifo_writer` de la arquitectura original del sistema. El primer bloque, denominado `sdr_en_fpga_filter`, se implementó para el filtrado de la señal de entrada. El segundo, el `sym_acquisition`, realiza el procesamiento correspondiente a la sincronización. El tercer bloque, `interface`, escribe el resultado del procesamiento en el `fifo_writer` sobre el cual continúa la cadena.

El bloque `sdr_en_fpga_filter` implementa un filtro FIR pasa bajos, necesario para el filtrado de los canales adyacentes. Esto fue necesario ya que el ancho de banda utilizado (8,13 MHz), que se encuentra ligado a la frecuencia de muestreo, es mayor al mínimo necesario para un canal de DTV (6,0 MHz). Este filtro fue implementado mediante el diseño de la respuesta al impulso e instanciación de un bloque de la librería de Nuand.

El bloque `interface` permite la lectura de los símbolos de salida del `sym_acquisition` y la escritura en el bloque `fifo_writer`. En este sentido, si se implementara por ejemplo la FFT en el FPGA, para continuar con la etapa de sincronismo, la salida de esta debería cumplir con el protocolo definido entre el `sym_acquisition` y el `interface` agregándose así a la cadena de procesamiento.

El principal bloque de la implementación, bloque de procesamiento sym_acquisition, corresponde a la versión en el FPGA del bloque de GNU Radio `OFDM Sym Acquisition` del módulo de `gr-isdbt`. El diseño del mismo implicó la especificación de una arquitectura interna al bloque basada en funciones. La misma consiste en bloques de memoria de entrada y salida, en un bloque de control y en uno de procesamiento. El sym_acquisition se encarga de llevar a cabo la sincronización temporal con los símbolos en la entrada y de la corrección fraccional en frecuencia de los mismos. Para la primer tarea, utiliza un algoritmo recursivo correspondiente a la función de correlación. Este algoritmo implicó la definición del mínimo necesario de memoria para la recepción de muestras en la entrada y de los requerimientos de lecturas sobre la misma. El mismo permite la sincronización con todos los símbolos recibidos. Para llevar a cabo la segunda función se implementó un algoritmo de rotación de las muestras de cada símbolo. La corrección se realizó aplicando una rotación a las muestras de cada símbolo mediante una fase creciente, la cual se realiza mediante el algoritmo iterativo CORDIC.

Los recursos consumidos para la implementación llegaron a un 50 % de memoria del tipo M9K y de *bits* genéricos. En cuanto a los elementos lógicos se utilizó tan solo un 22 % de los disponibles. Estos valores incluyen el consumo de la arquitectura original.

Validación de la Implementación Se definieron distintos entornos de prueba, tanto para simular como para realizar pruebas en *hardware*, que fueron imprescindibles para la validación de la implementación. La incorporación del uso de *test-benchs* en la mitad del proyecto fue un punto de inflexión en el área de validación. Se utilizaron diversas herramientas: ModelSim, Octave y GNU Radio Companion para definir ambientes de depuración y validación, además de que se generó una amplia biblioteca de señales de referencia también esenciales para la validación del sistema.

Se implementaron varios *scripts* en Octave que permitieron convertir las señales capturadas al formato de datos requerido por cada herramienta. Además, se definieron numerosos *scripts* para simular las funciones implementadas en *hardware* (en particular para emular su comportamiento numérico) y así contar con resultados ideales esperados. Los mismos se validaron contra la salida del `gr-isdbt`, bloque de referencia para la validación en todas sus etapas.

Se realizó una etapa intermedia de validación en la cual sólo se verificó la correcta sincronización temporal. Para ello, se utilizó una memoria interna en el FPGA con datos conocidos que generaba indefinidamente la señal de entrada. Se verificó mediante el uso de *scripts* que la señal obtenida era idéntica a la esperada. La misma prueba se realizó en *hardware* obteniendo el mismo resultado.

En última instancia, se validó el sistema completo. En primer lugar se implementó un *test-bench* que incluía la arquitectura completa, esto es: sdr_en_fpga_filter, sym_acquisition, interface y fifo_writer. Teniendo en cuenta la limitante temporal de las simulaciones, se verificó para alrededor de 1000 símbolos. Para ello, se guardaron los datos en la salida de la simulación y luego de ser convertidos al formato de GNU Radio se procesaron hasta la etapa de sincronismo

(`Sync And Channel Estimation`) obteniendo una constelación sin ruido.

Finalmente, se implementó el sistema completo en el FPGA y se realizaron múltiples capturas de todos los canales uruguayos obteniendo con éxito el vídeo en la salida del procesamiento del GNU Radio Companion.

7.2. Conclusiones

Conclusiones Finales y Trabajos Futuros En este proyecto se logró la implementación de un receptor *full-seg* de DTV según el estándar ISDB-T con un bloque implementado en un FPGA. Esto implicó no sólo la solución del manejo de algoritmos que requieren cierta precisión numérica en *hardware* sino que también la solución del pasaje de datos vectoriales por un sistema que originalmente enviaba muestras independientes.

No se pudo conseguir la implementación de la `FFT` ni del `Sync And Channel Estimation` como estaba planteado en un comienzo, sin embargo, creemos que todo el camino recorrido hasta este punto hará que la implementación de estos dos últimos sea más rápida que la del primero.

Mediante el diseño realizado sobre el FPGA se concluye que para realizar operaciones sobre el mismo que requieran buena precisión en los resultados, no solo es suficiente con la elección del algoritmo y un adecuado dimensionamiento del mismo sino que hay que tener en cuenta los recursos disponibles en el FPGA ya que pueden llegar a limitar la implementación.

Debido a que se trabajó sobre una placa ya diseñada, existen ciertas limitaciones referentes a la interfaz de salida que pueden limitar la representación y la cadencia de salida de los bloques de procesamiento. También existieron condiciones sobre las plataformas en las que se realizó el diseño como el Quartus y ModelSim así como de Octave y GNU Radio.

Finalmente, a pesar de estos elementos, se implementó una solución adecuada mediante un mecanismo de diseño, implementación y prueba en base a las técnicas presentadas: diseño en VHDL, simulaciones y pruebas en el *hardware*. La solución permite transmitir las muestras generadas en el FPGA hacia GNU Radio sin inconvenientes, a pesar de que sea a una cadencia diferente a la frecuencia de muestreo y ser vectoriales y aún quedan recursos disponibles para agregar otros bloques de procesamiento.

7.3. Trabajos Futuros

En este sentido, planteamos como primer trabajo futuro la implementación en el FPGA de los dos bloques mencionados anteriormente. En principio, suponemos que será una tarea realizable pero que tiene dos elementos críticos (al igual que los tuvo el bloque que se implementó) la limitación de los recursos del FPGA y la resolución numérica del algoritmo en *hardware*.

Ante la posibilidad de que exista una limitante en el caso anterior, proponemos como otro posible trabajo futuro, la implementación del receptor completo pero

para el sistema 1-*seg*. Al trabajar con menos datos, simplifica notoriamente el consumo de recursos del FPGA en el primer bloque del sistema.

En tercer lugar, creemos que a partir de las soluciones encontradas, se podría implementar otro tipo de receptor que requiriera de igual manera, el pasaje de señales vectoriales o bien de información de metadatos hacia el *software*.

Por último, consideramos que se debería llevar a cabo una comparación de la *performance* del sistema con un bloque en *hardware* y el sistema implementado completamente en *software*. En particular, del bloque implementado en el FPGA con su análogo en el sistema `gr-isdbt`.

Apéndice A

Esquemático de la bladeRF

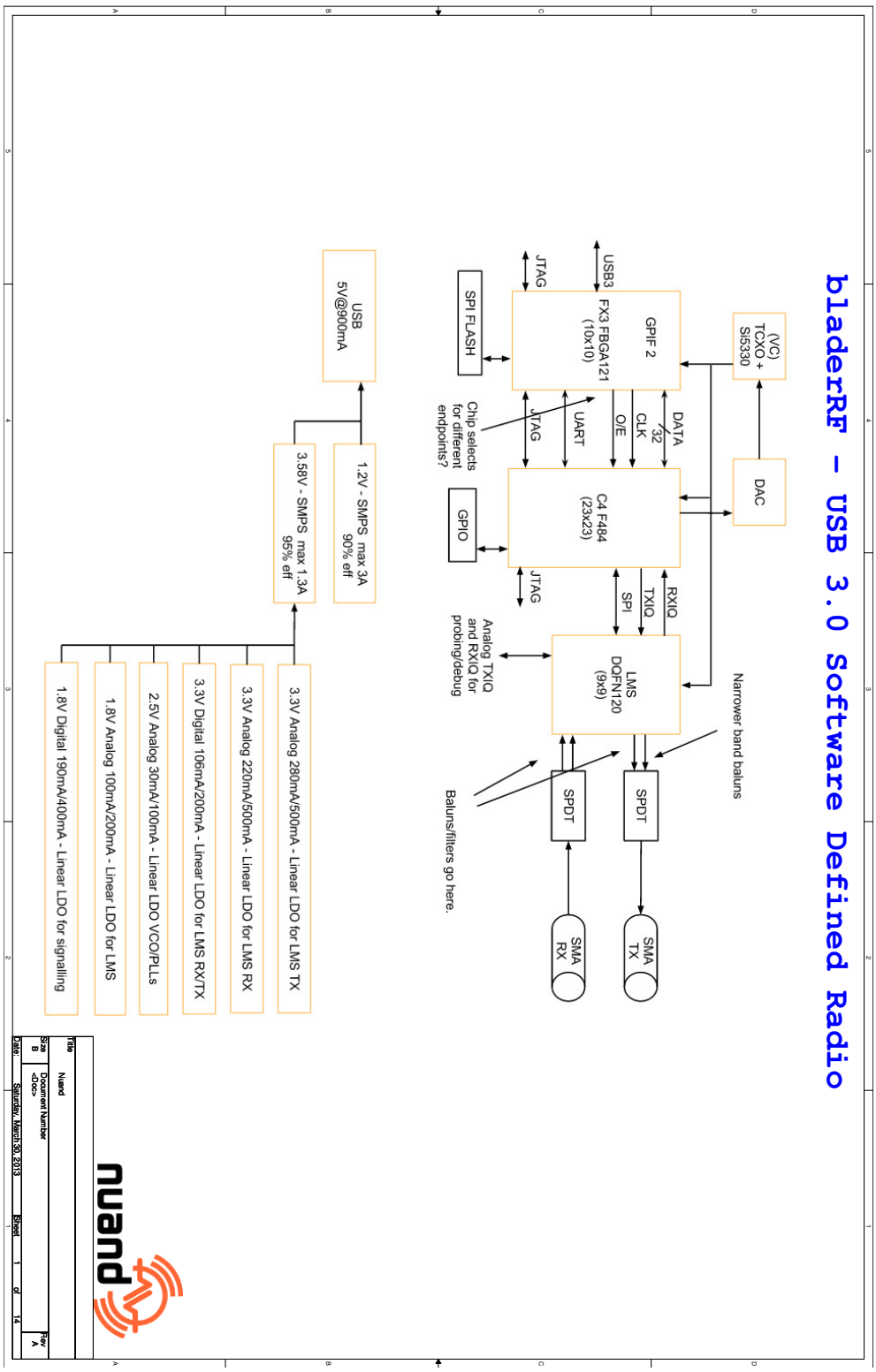


Figura A.1: Página 1 de 14 - Esquemático bladeRF

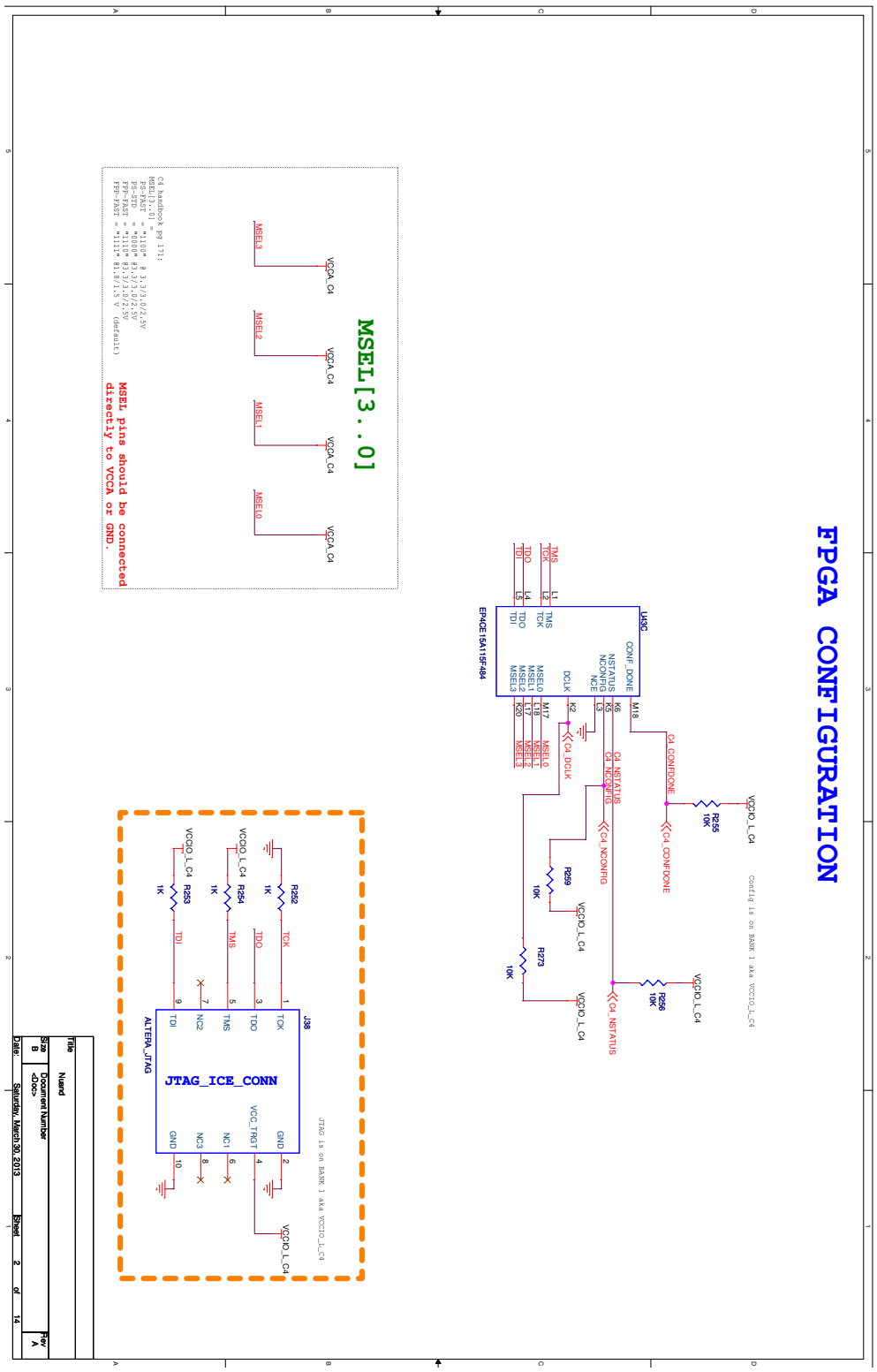


Figura A.2: Página 2 de 14 - Esquemático bladeRF

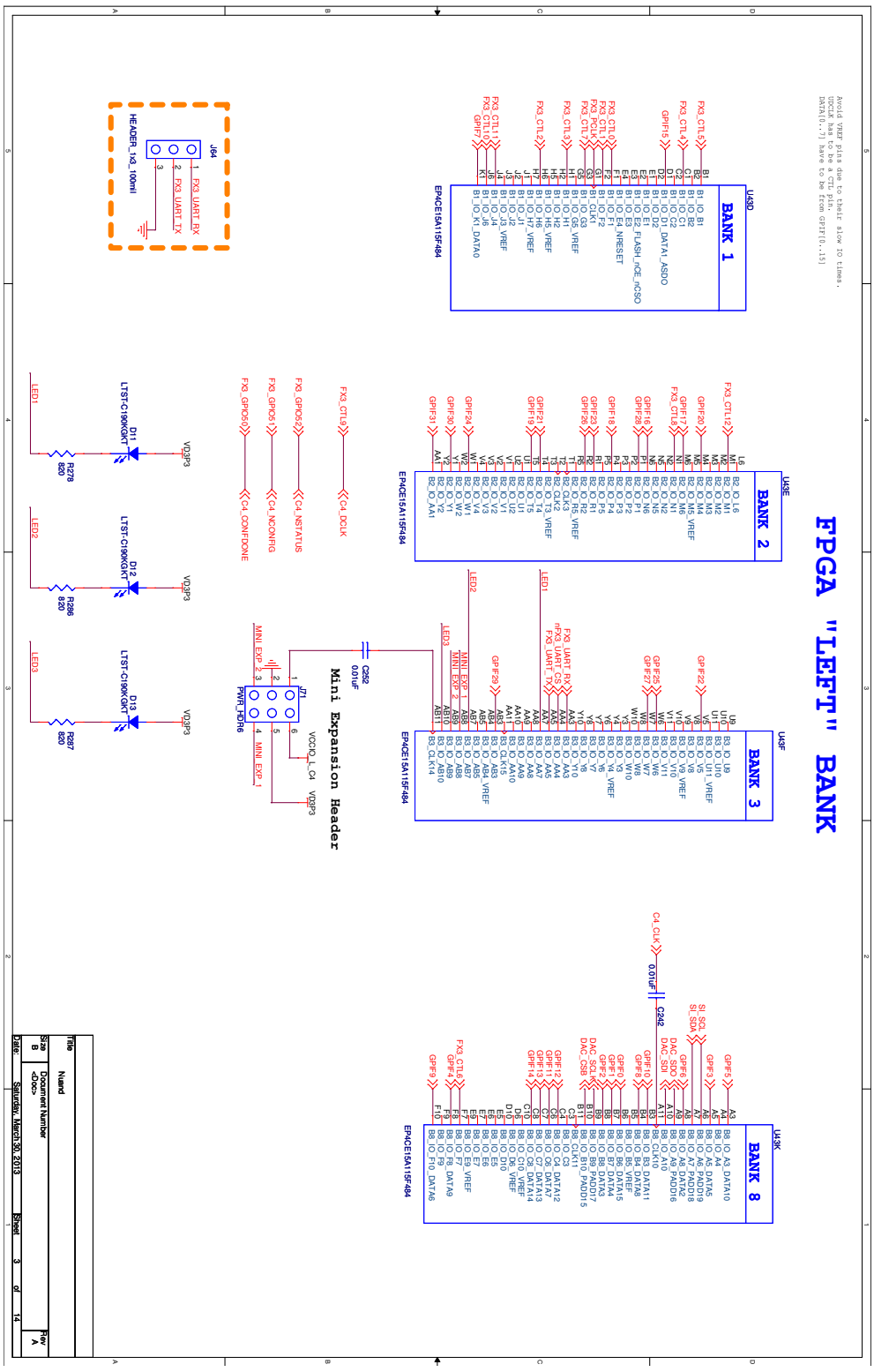


Figura A.3: Página 3 de 14 - Esquemático bladeRF

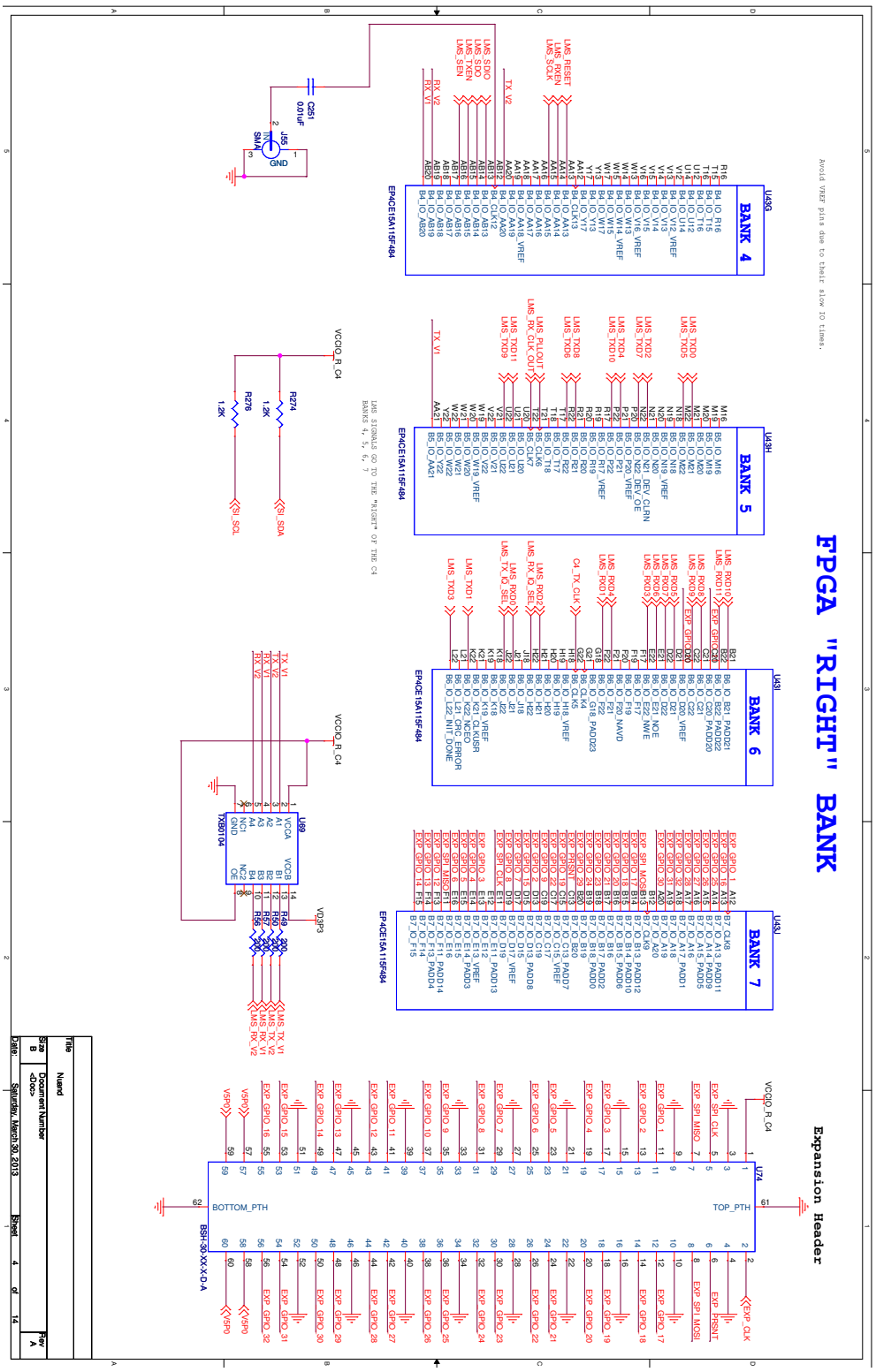


Figura A.4: Página 4 de 14 - Esquemático bladeRF

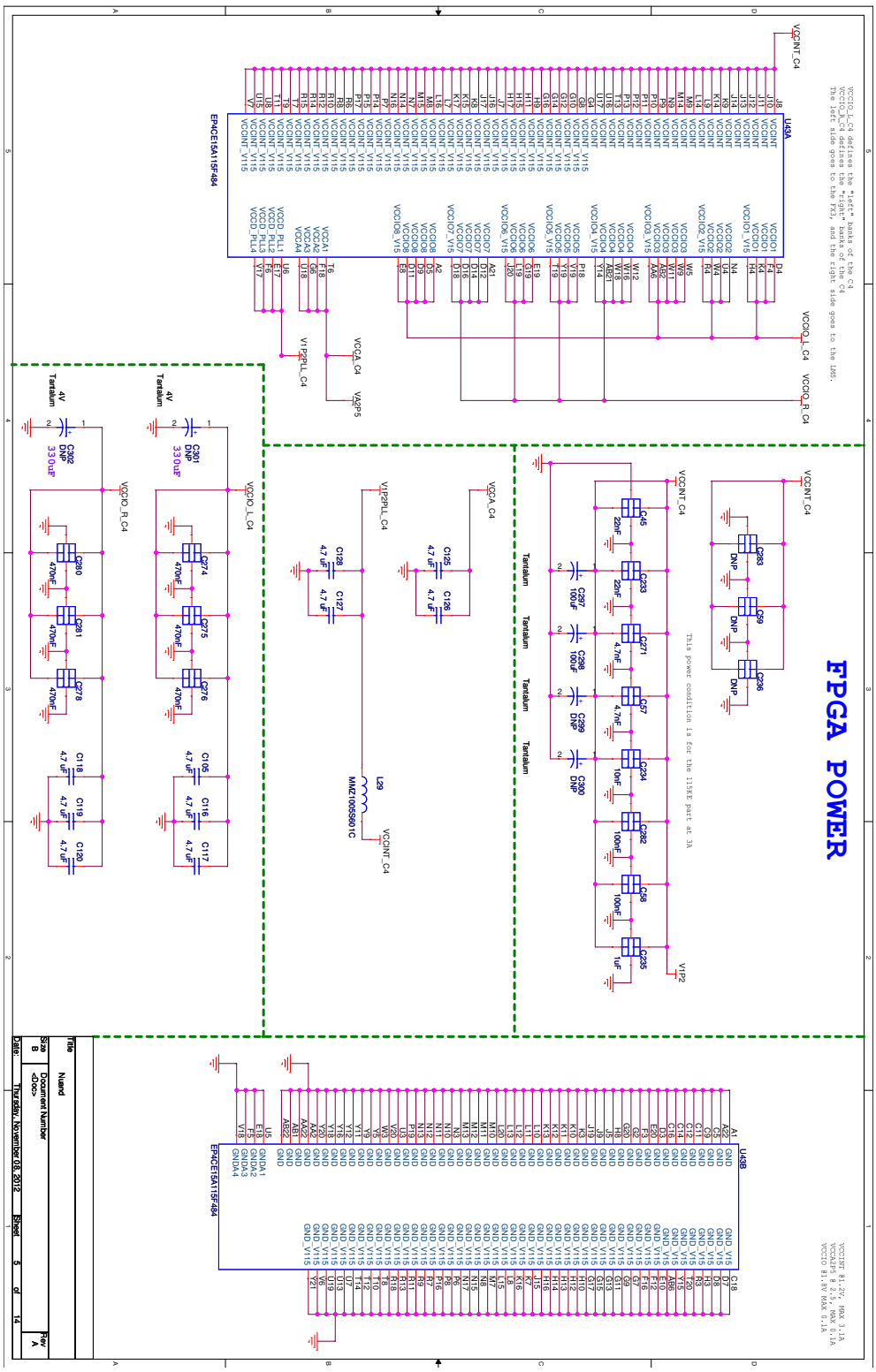


Figura A.5: Página 5 de 14 - Esquemático bladeRF

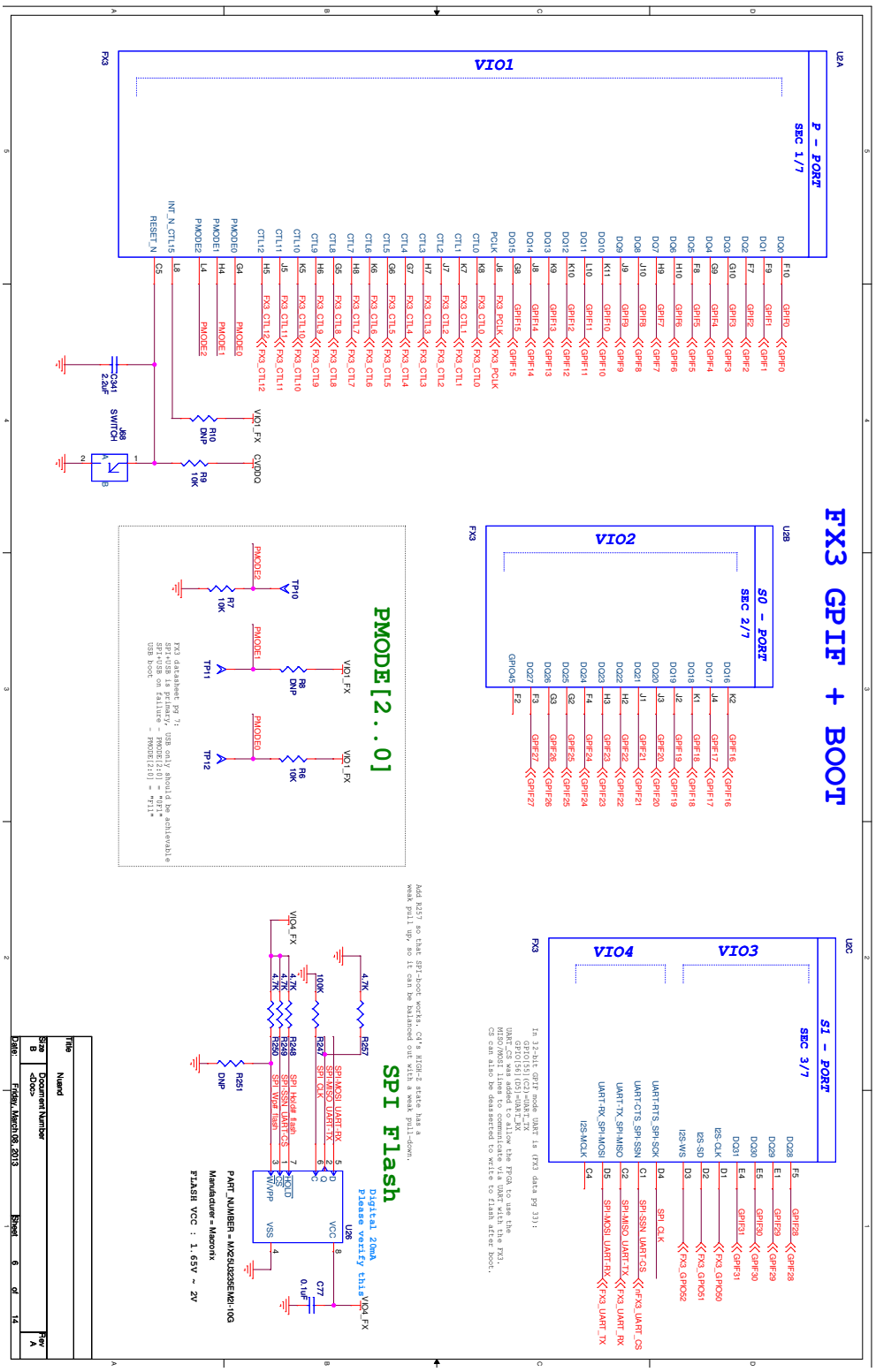


Figura A.6: Página 6 de 14 - Esquemático bladeRF

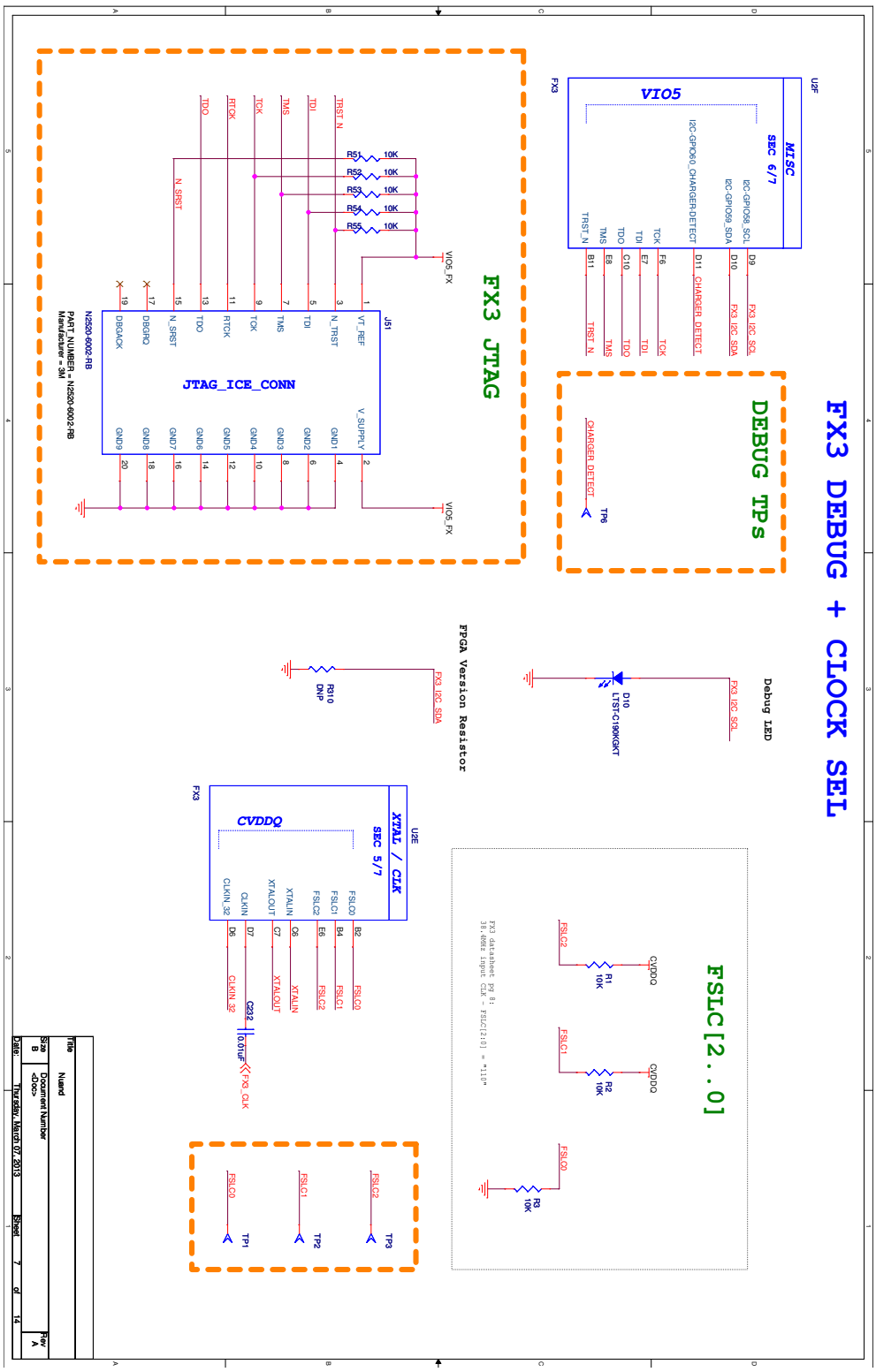


Figura A.7: Página 7 de 14 - Esquemático bladeRF

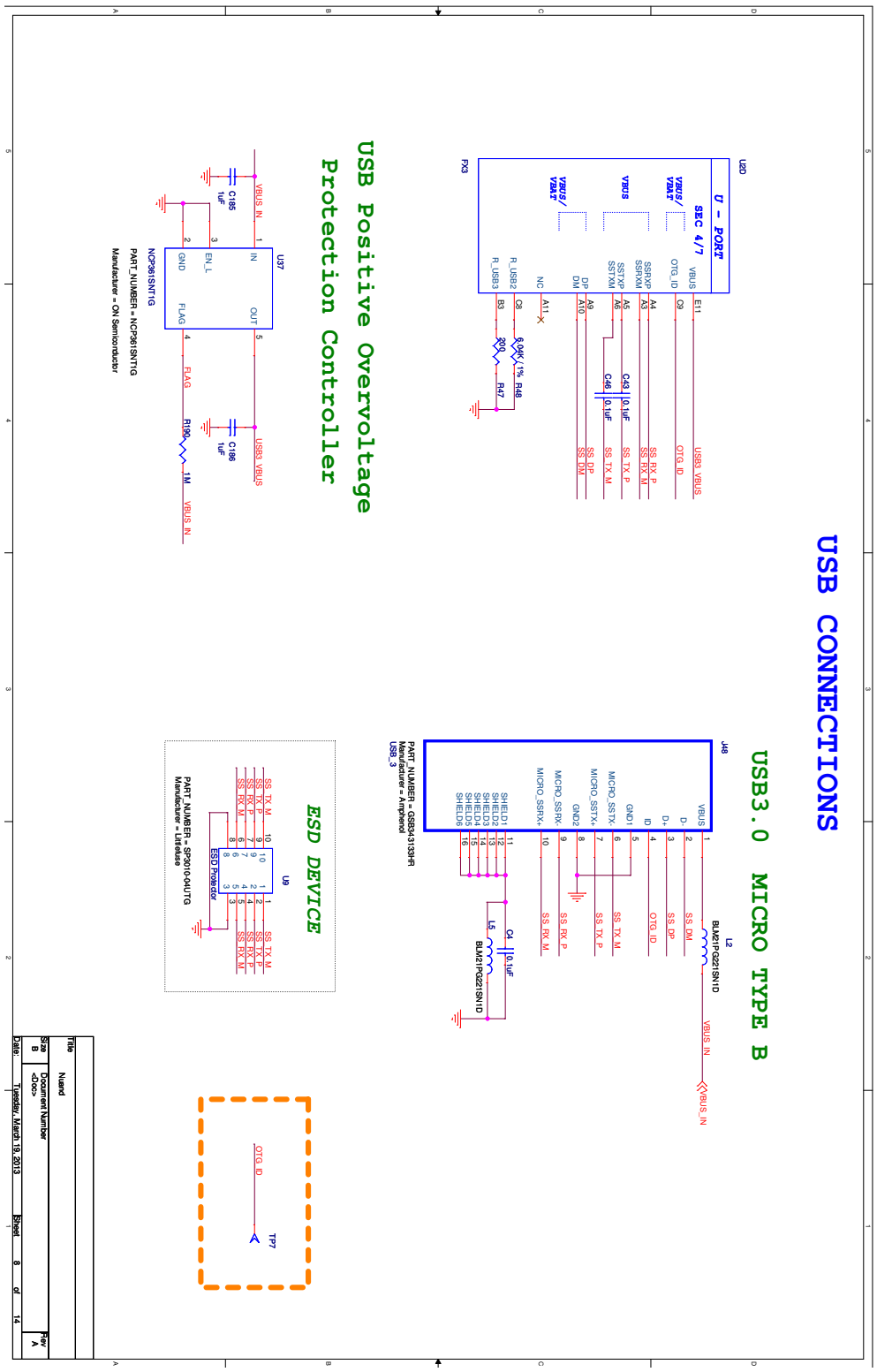


Figura A.8: Página 8 de 14 - Esquemático bladeRF

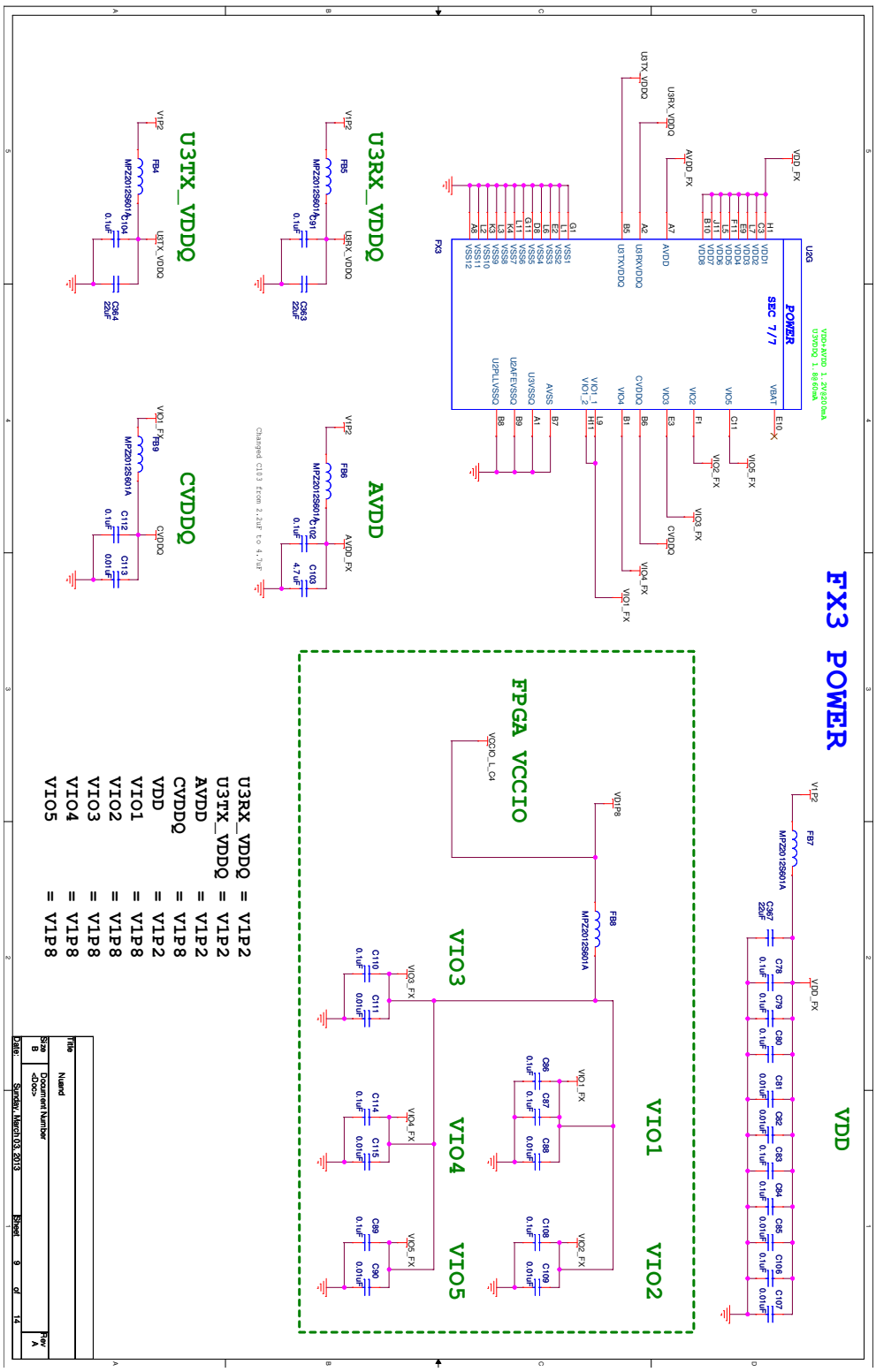


Figura A.9: Página 9 de 14 - Esquemático bladeRF

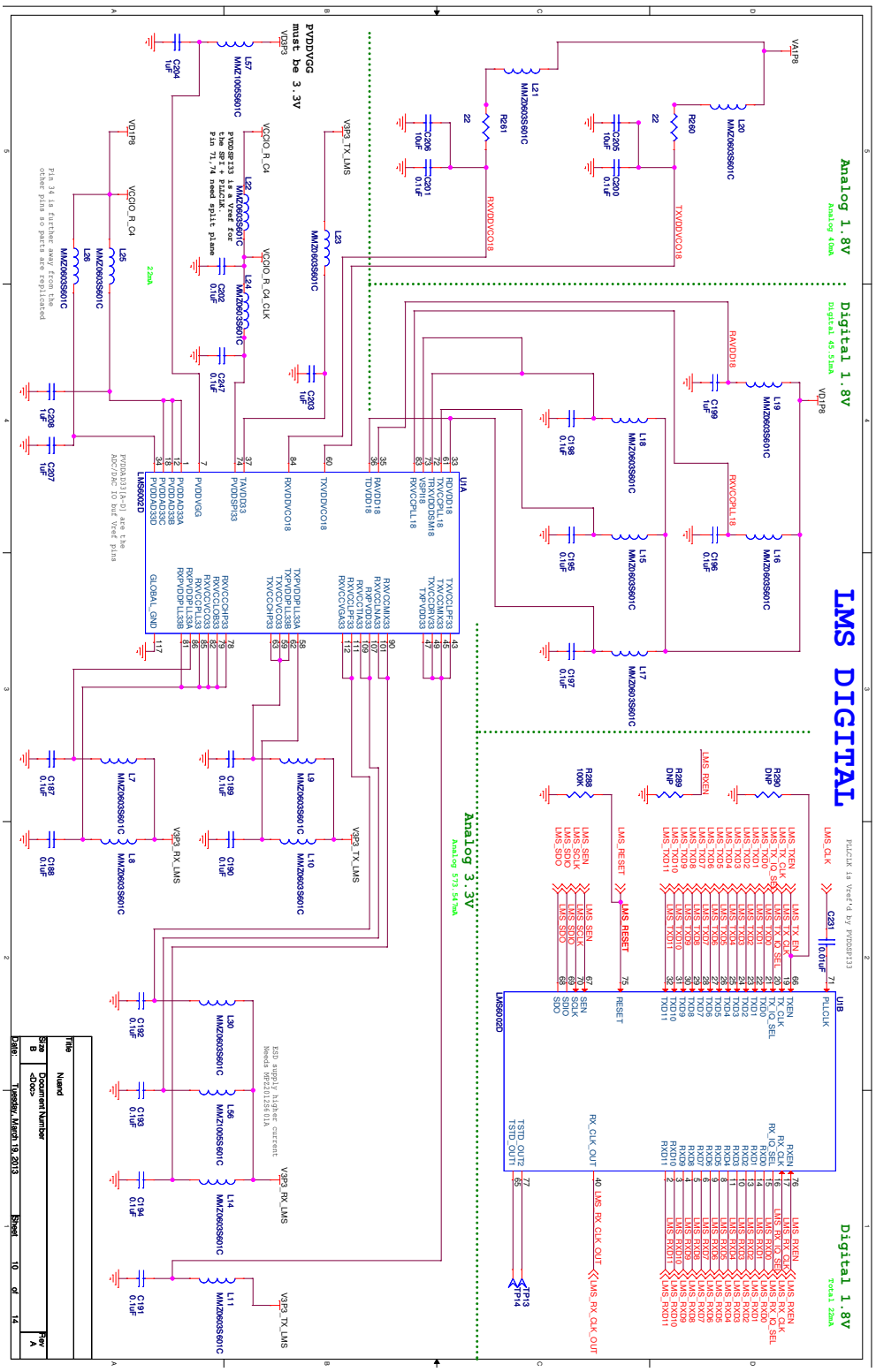


Figura A.10: Página 10 de 14 - Esquemático bladeRF

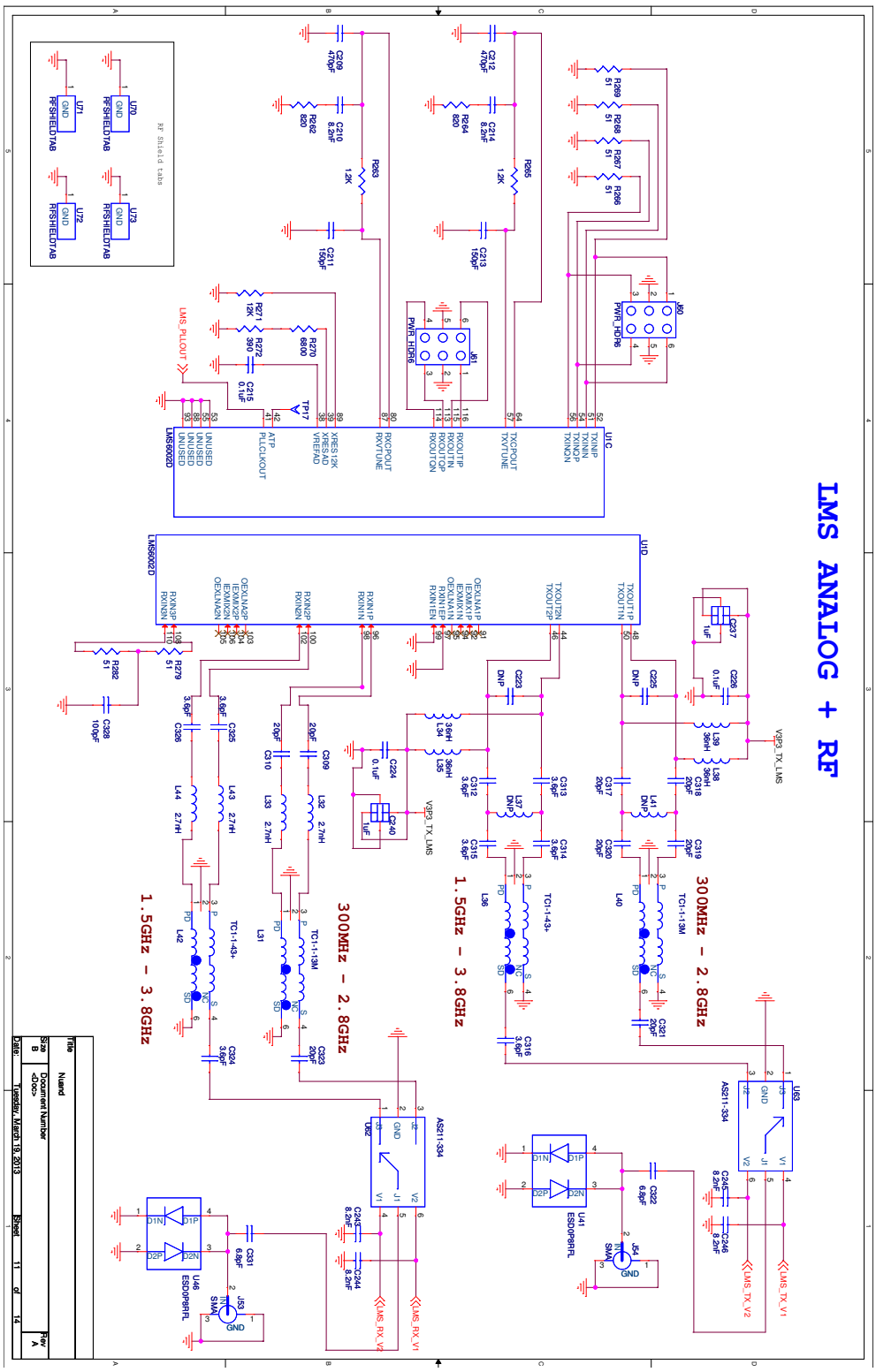


Figura A.11: Página 11 de 14 - Esquemático bladeRF

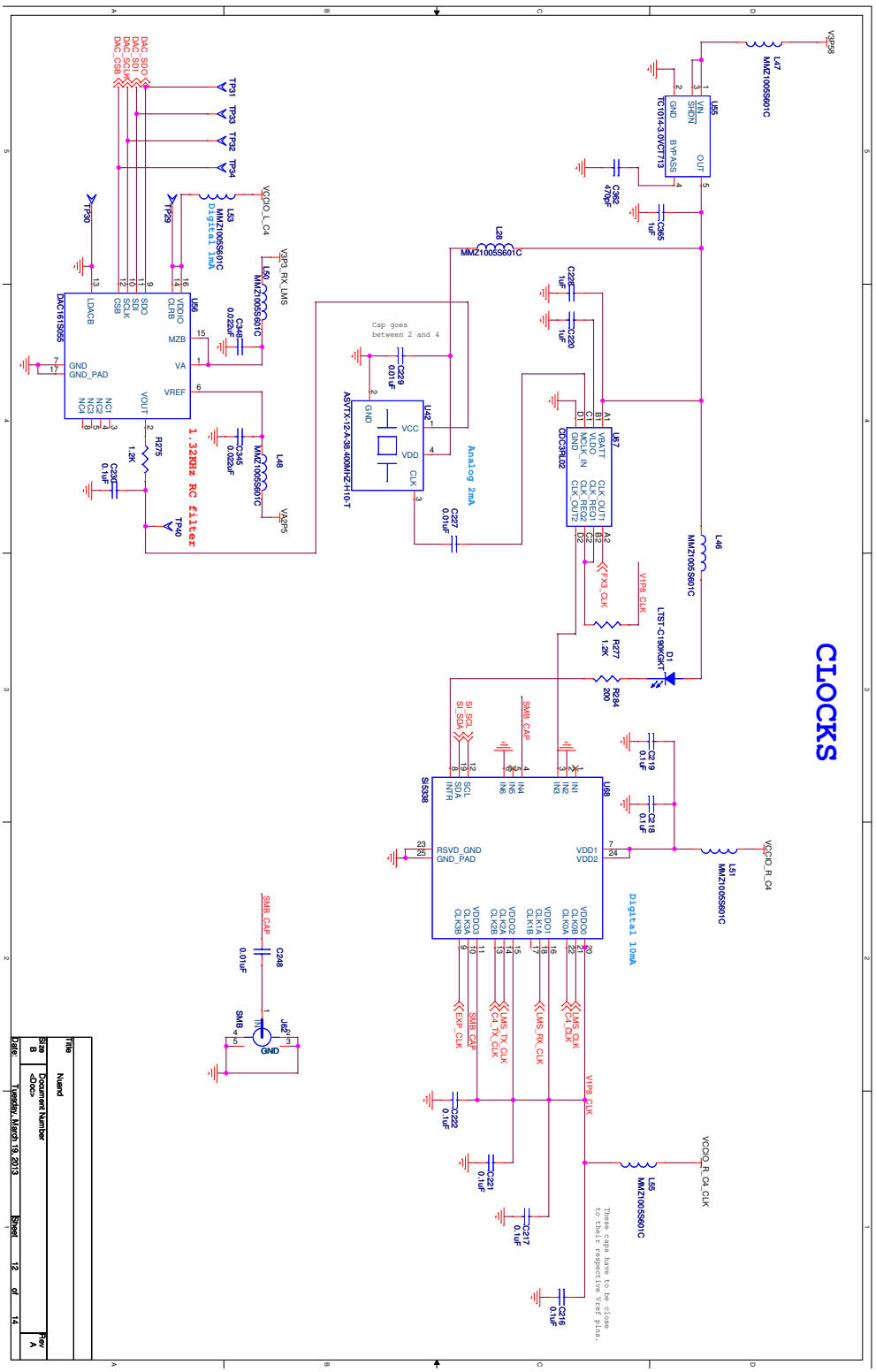


Figura A.12: Página 12 de 14 - Esquemático bladeRF

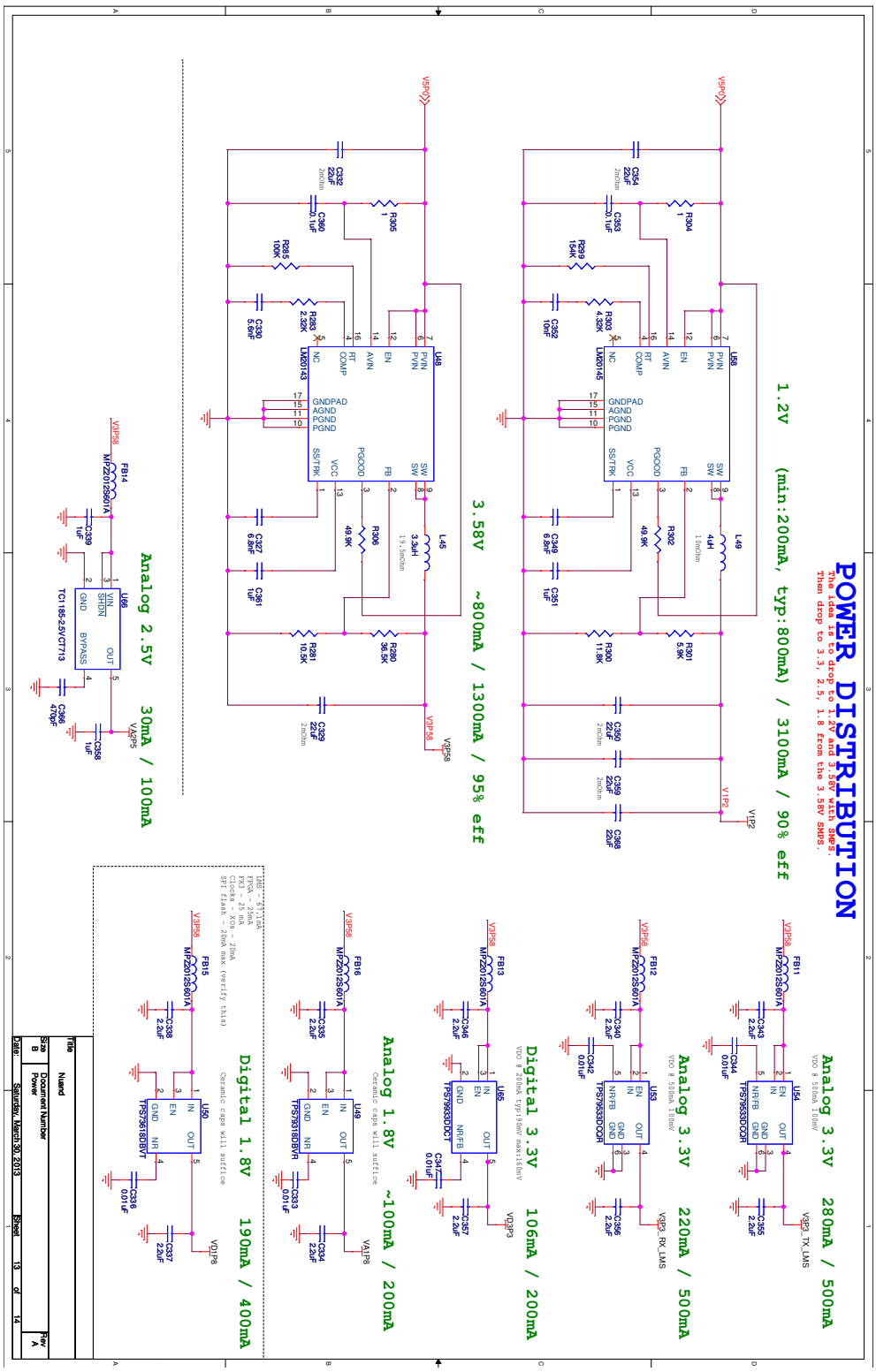


Figura A.13: Página 13 de 14 - Esquemático bladeRF

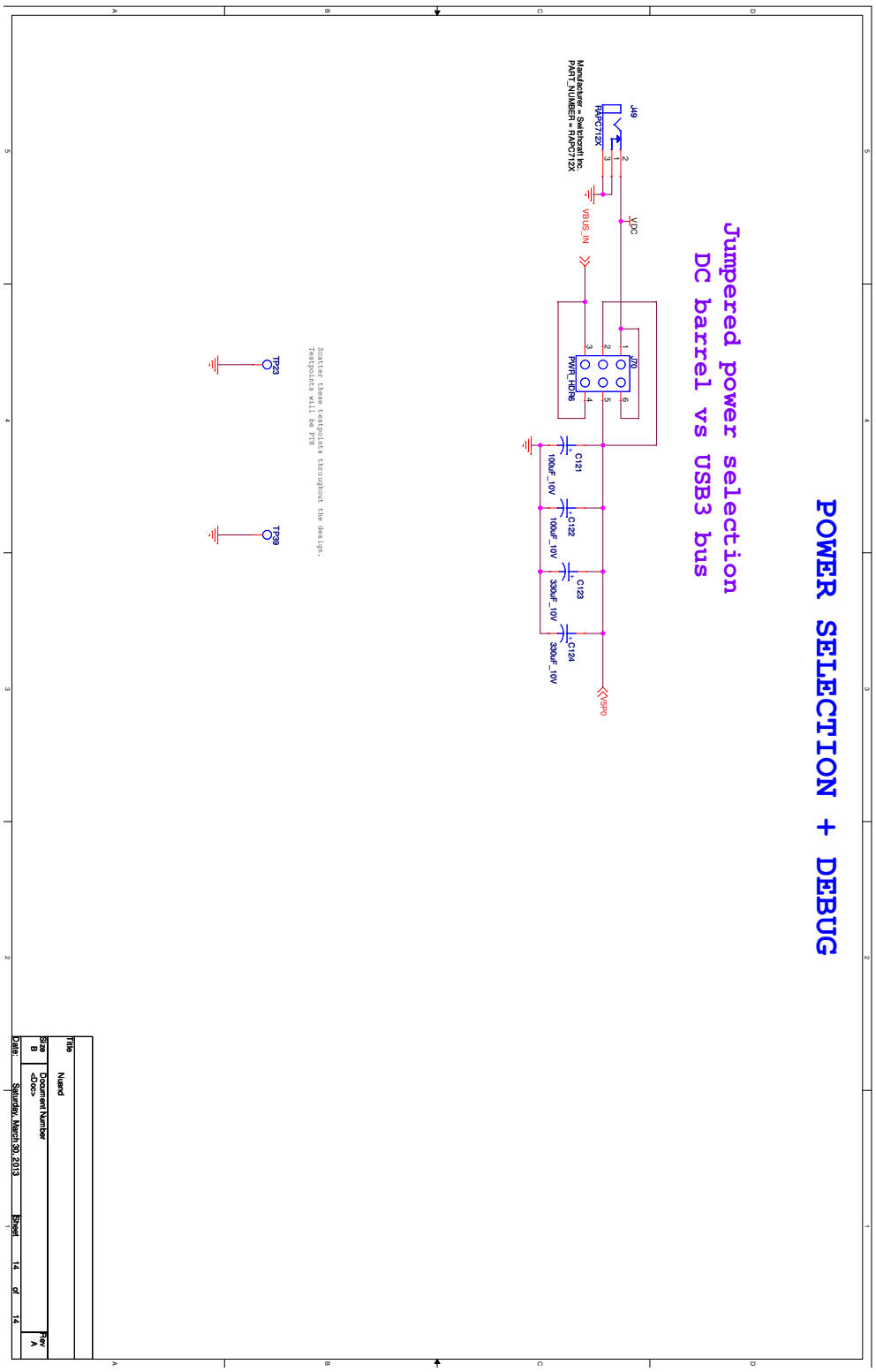


Figura A.14: Página 14 de 14 - Esquemático bladeRF

Esta página ha sido intencionalmente dejada en blanco.

Apéndice B

Detalles de la Arquitectura del FPGA de la placa bladeRF

B.1. Estudio `fifo_writer` sin modificar

A continuación se presenta el estudio del bloque `fifo_writer` sin modificar. En la Figura B.1 se presenta el diagrama de entrada - salida del mismo. Los puertos `clock`, `reset` y `enable` son entradas de control generales del bloque. Los puertos `usb_speed`, `meta_en` y `timestamp[63..0]` contienen información de configuración (en el caso de los dos primeras) o de información a enviar (en caso del último)[1]. Los puertos de entrada `in_i[DATA_WIDTH-1..0]` e `in_q[DATA_WIDTH-1..0]` contienen el valor I, Q de las muestras y el puerto `in_valid` indica en qué flancos los mismos son válidos. Los puertos `fifo_usedw`, `fifo_clear`, `fifo_write`, `fifo_full` y `fifo_data` corresponden al control y escritura en el *fifo* de datos `rx_sample_fifo` mientras que las análogas corresponden al *fifo* de metadatos `rx_meta_fifo`, estas últimas comienzan con el prefijo `meta_fifo_`. Por último, hay tres puertos relacionados con el control de *overflow*: `overflow_led`, `overflow_count` y `overflow_duration`.

Se presentan los detalles de los puertos y las señales de interés para el desarrollo de la interfaz bladeRF - GNU Radio.

1. De acuerdo al tipo de USB el puerto `usb_speed` define una señal llamada `dma_buf_sz` que toma el valor del doble de la cantidad de muestras por mensaje ya que las muestras son válidas cada dos flancos de reloj. `usb_speed` = 0 corresponde a USB 3.0 siendo 508 muestras por mensaje (`dma_buf_sz` = 1015) mientras que `usb_speed` = 1 a USB 2.0 con 252 muestras por mensaje (`dma_buf_sz` = 503). La señal `dm_buf_sz` se utiliza en el proceso descripto en el siguiente punto.

```
dma_buf_sz <= to_signed(1015, dma_buf_sz'length) when
    usb_speed = '0' else to_signed(503, dma_buf_sz'length);
```

[1] Los *timestamps* son enviados únicamente si están habilitados los metadatos

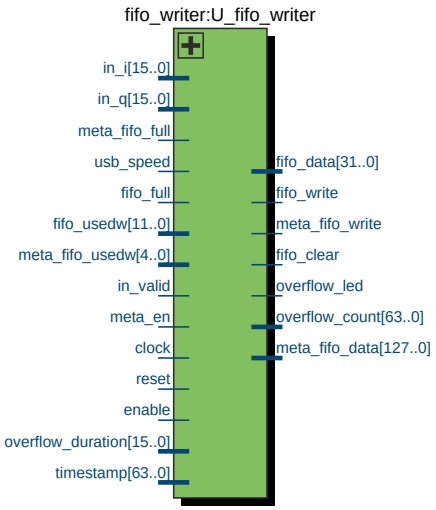


Figura B.1: Diagrama de Entrada-Salida del bloque `fifo_writer` original.

2. El primer proceso del código (no tiene nombre) se encarga de manejar las señales de control de escritura de metadatos: `dma_downcount` y `meta_written`.

3. Señal de control `buf_enough`

   Esta señal es una de las que permite que se active la señal de escritura en el *fifo* de metadatos (`meta_fifo_write`). Esta toma el valor de 1 cuando `dma_downcount` comienza. Observar en las simulaciones siguientes el comportamiento de (`meta_fifo_write`).

   ```
   buf_enough <= '1' when (dma_downcount = (dma_buf_sz)) else '0';
   ```

En la Figura B.2 se representa un diagrama de estados del proceso en cuestión. En cada flanco de reloj (el reloj que entra al bloque es el de la frecuencia de muestreo f_s) en caso de que el bloque y los metadatos estén habilitados: `dma_downcount` se decrementa en 1. Cuando llega a -1 si se cumple que el *fifo* de datos no está lleno y tiene suficiente espacio para seguir aceptando tantos datos como `dma_buf_sz` si `(meta_written or in_valid)` *resetea* el contador `dma_downcount` llevándolo a su valor por defecto (`dma_buf_sz`).

De esta forma se manejan las señales de control que influyen en la escritura de metadatos en el *fifo*:

```
buf_enough <= '1' when ( dma_downcount = (dma_buf_sz)) else '0';

meta_fifo_write <= '1' when (enable = '1' and meta_en = '1'
                             and buf_enough = '1') else '0';

meta_fifo_data <= x"FFFFFFFF" & std_logic_vector(timestamp)
```

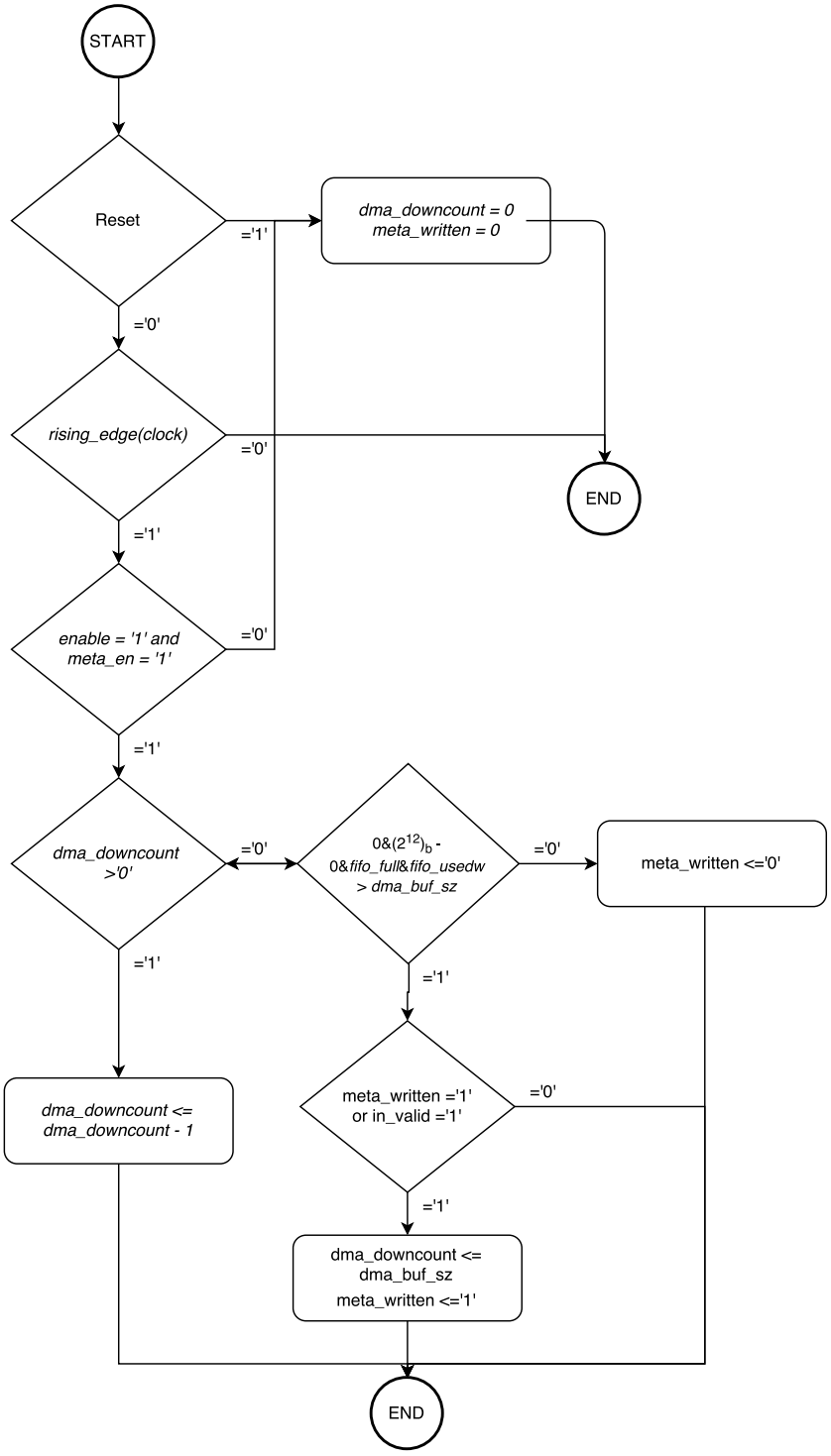


Figura B.2: Diagrama de estados del primer proceso del bloque fifo_writer original

```
                    & x"12344321";

meta_written_reg <= '0' when reset = '1' else
                    meta_written when rising_edge(clock) ;
```

Por último, se presentan dos simulaciones del bloque en estudio. En la Figura B.3 se presenta la simulación al comienzo en donde queremos destacar el comportamiento de la señal `dma_downcount` y `meta_fifo_write` y en la Figura B.4 en la transición de dos mensajes.

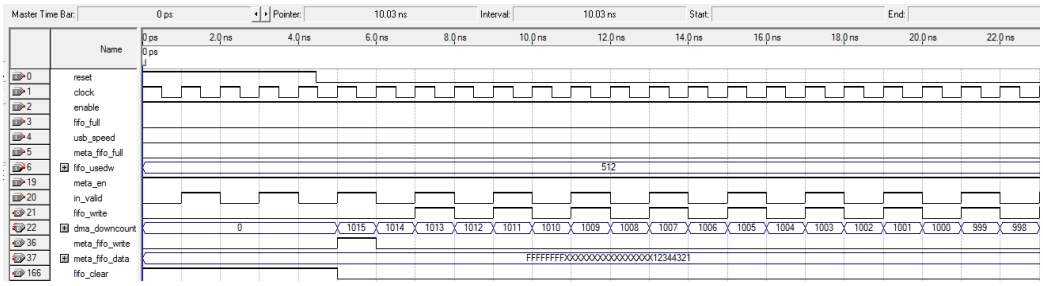


Figura B.3: Simulación del bloque `fifo_writer` sin modificaciones al comienzo.

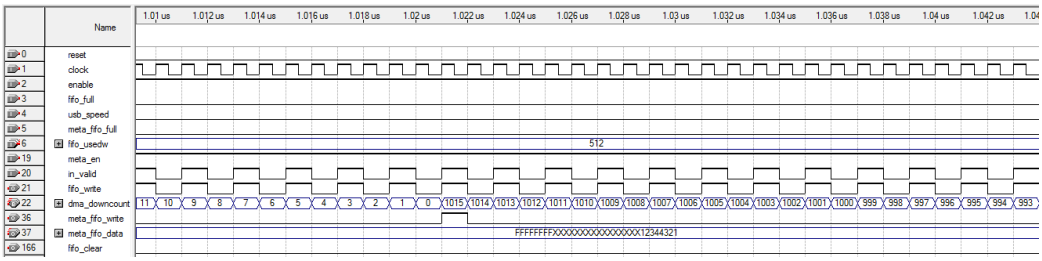
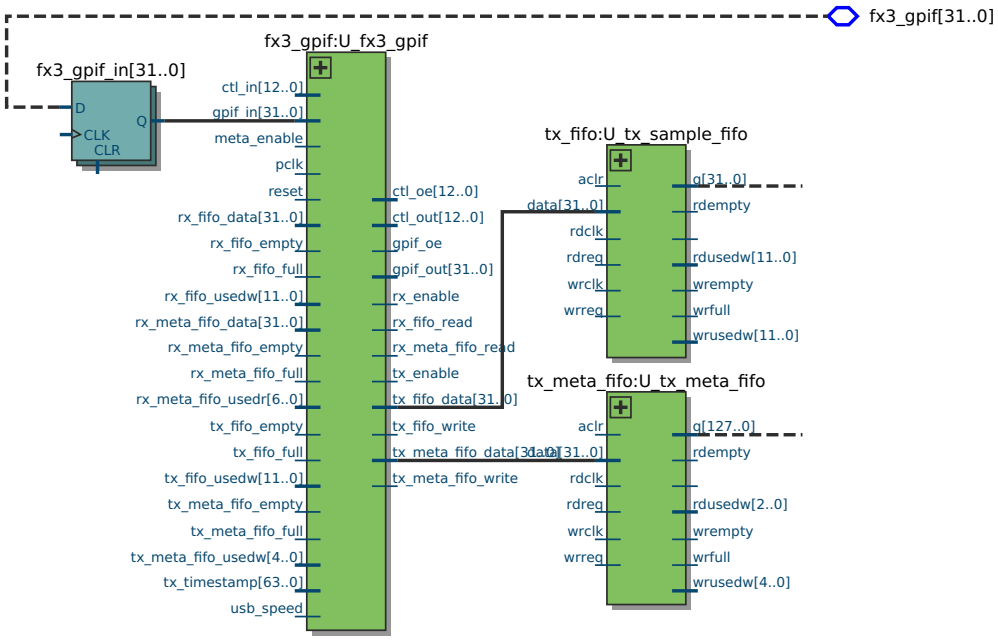


Figura B.4: Simulación del bloque `fifo_writer` sin modificaciones, fin de un mensaje comienzo de otro.

Nota: el bloque `fifo_writer` tiene otros procesos que no son de importancia para el desarrollo. Los mismos generan señales de *overflow* que idealmente nunca llegarán a activarse pero podrían ser útiles en una etapa posterior.

B.2. Bloques pertenecientes al flujo de datos en caso transmisión

En la Figura B.5 y en la Figura B.6 se presenta el RTL de los bloques que participan en el camino de transmisión de datos.



(a) Bloques: `fx3_gpif`, `tx_sample_fifo` y `tx_meta_fifo`.

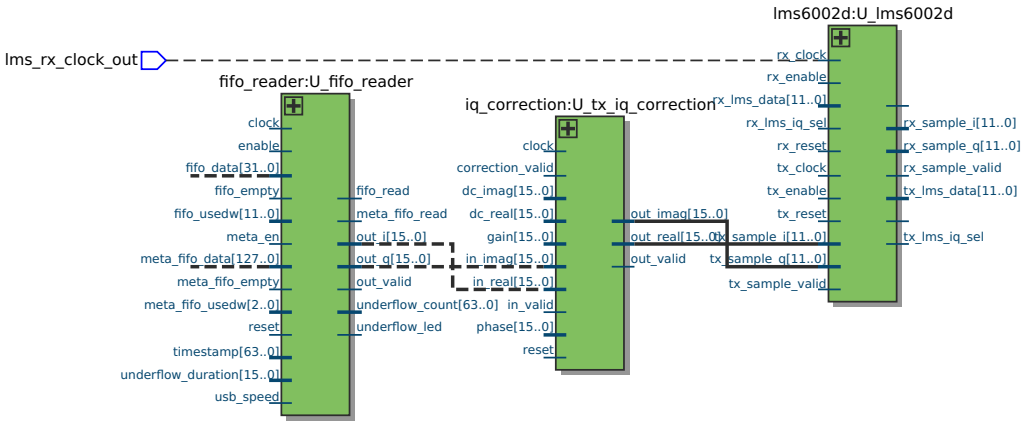
Figura B.5: Bloques pertenecientes al flujo de datos para el camino de transmisión.

B.3. Interfaz GPIO del `nios_system`

En la Figura B.7 se presenta la interfaz GPIO del `nios_system`. En particular, se dejaron únicamente las conexiones relacionadas con la configuración del usb y de los metadatos para la recepción.

B.4. Alternativas Fuentes de datos

En la Figura B.8 y en la Figura B.9 se presentan distintas alternativas de fuentes de datos. Las mismas son: recepción de muestras por aire provenientes del



(a) Bloques: `fifo_reader`, `iq_tx_correction` y `lms6002d`.

Figura B.6: Bloques pertenecientes al flujo de datos para el camino de transmisión.

LMS, contador interno de 32 *bits* y realimentación (*digital loopback*).

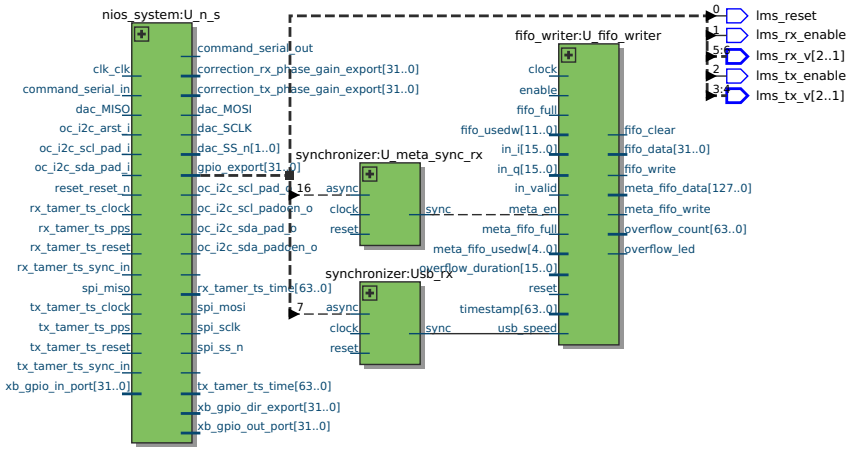
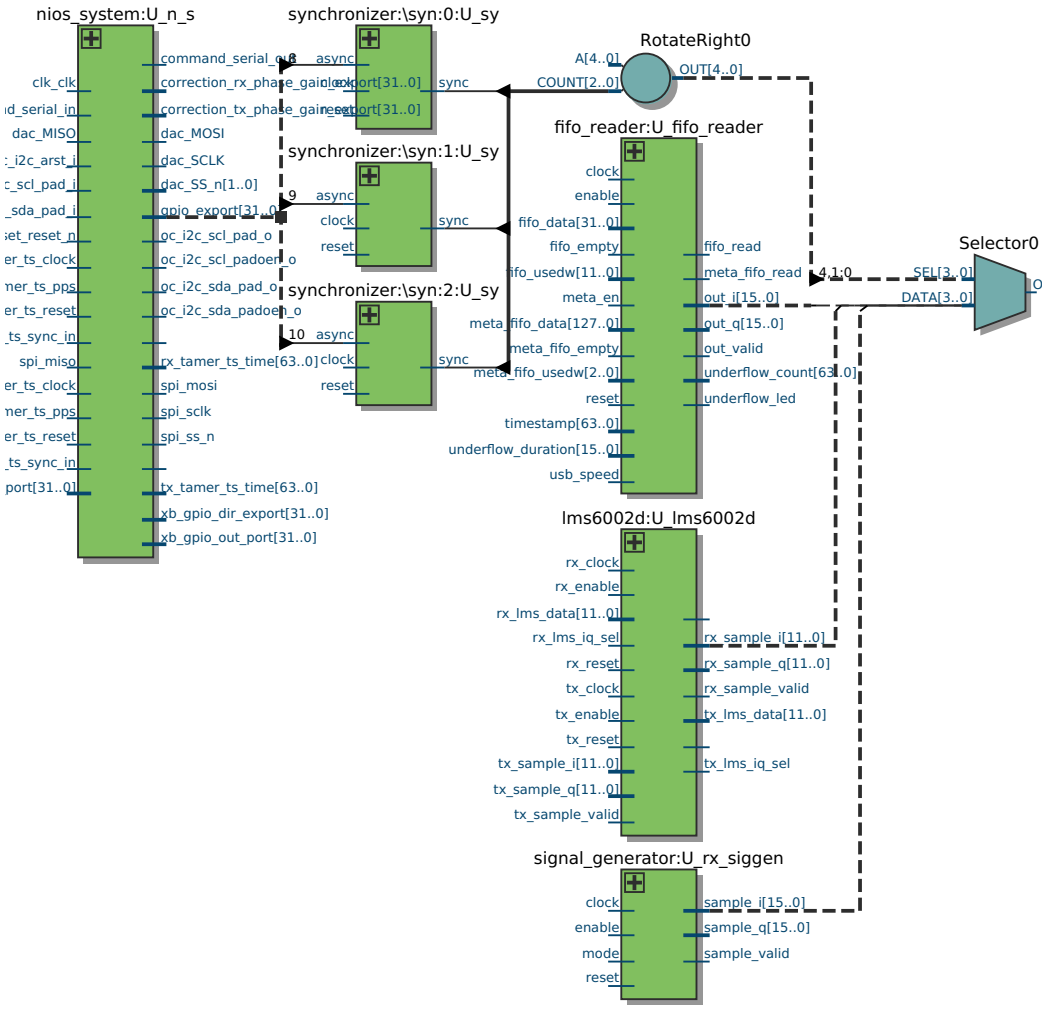
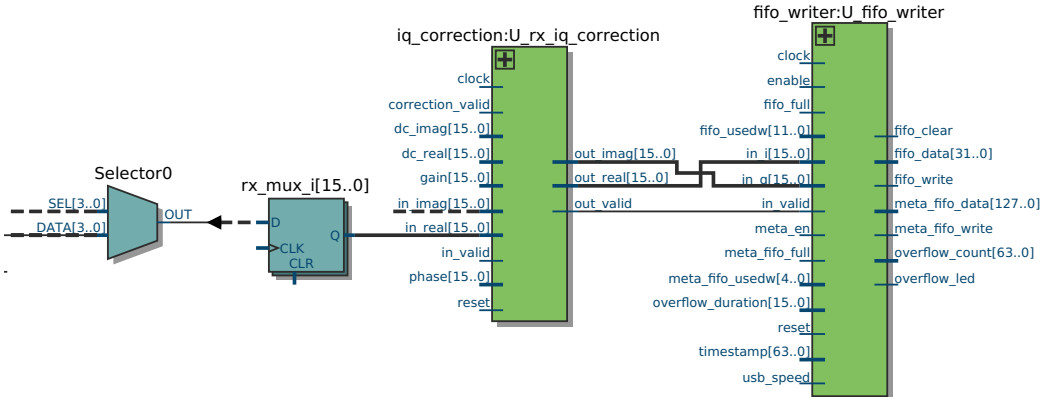


Figura B.7: Interfaz GPIO del `nios_system`: se muestra únicamente conexión relacionada a la configuración del usb y de los metadatos para la recepción.



(a) Parte 1 - Alternativas Fuentes de datos. Bloques: `nios_system`, `synchronizer (3 veces)`, `fifo_reader`, `lms6002d` y `signal_generator`.

Figura B.8: Distintas alternativas de fuente de datos para la recepción de los mismos en la PC definida por los valores de los *bits* 8, 9 y 10 del gpio del `nios_system`.



(a) Parte 2 - Alternativas Fuentes de datos. Bloques `rx_iq_correction` y `fifo_writer`.

Figura B.9: Distintas alternativas de fuente de datos para la recepción de los mismos en la PC definida por los valores de los *bits* 8, 9 y 10 del gpio del `nios_system`.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice C

Herramientas utilizadas

C.1. Git

Git es un sistema de control de versiones que es ampliamente utilizado para el desarrollo de *software* y otro tipo de actividades que requieren control de versiones [10].

Utilizamos este sistema para el desarrollo de la arquitectura en el FPGA, para ello se partió del repositorio de Nuand [22]. También se utilizó para la implementación del bloque `messages2symbol` y de las modificaciones en el `gr-osmosdr`.

C.2. Gnuradio

GNU Radio [30] es una herramienta de desarrollo de sistemas de radio definidos por *software* de código abierto, libre y gratuito. Provee una amplia librería de bloques con diversas funciones que van desde filtrado, detectores de pico, operadores matemáticos hasta moduladores OFDM y decodificadores y otras funciones genéricas de procesamiento.

C.2.1. Funciones de GNU Radio Utilizadas

A continuación se describen algunas funcioes de GNU Radio utilizadas para el desarrollo del bloque `messages2symbol`

Para realizar el procesamiento del bloque `messages2symbol` de forma correcta (ver especificación del bloque Sección 4.3.2), garantizando el control de flujo, se tuvo en cuenta el uso correcto de las siguientes funciones de GNU Radio (ver API [16]):

- `messages2symbol_impl::messages2symbol_impl`

  Constructor. Permite obtener parámetros de entrada genéricos, inicializar variables globales y algunas funciones. Se puede utilizar la función gr::block

para describir el bloque (por ejemplo tamaño y cantidad de entradas y salidas). La variable global *n_read_samples* lleva la cuenta de la cantidad de muestras leídas que forman un símbolo que se arma en la salida.

- `set_relative_rate`

  Permite indicar la relación aproximada entrada-salida para que el programa genere las entradas necesarias que indique el forecast, se seteó la relación en 1/(d_n_messages) (1/17). Se utiliza en el constructor. Está función fue necesaria para tamaño de símbolo 8k, en general no es necesaria.

- `messages2symbol_impl::forecast(int noutput_items, gr_vector_int &ninput_items_required)`

  Permite indicar la cantidad necesaria de items en la entrada (ninput_items _required) para generar cierto numero de salidas (noutput_items). Se configuró para que sean requeridos 17 mensajes en la entrada para cada símbolo ($d\_n\_messages = symbol\_length/message\_length + 1$) a generar en la salida (generalmente son necesarios 17 mensajes por cada símbolo o en el peor caso 18 y este último queda contemplado en el caso general de 17 mensajes requeridos).

- `messages2symbol_impl::general_work (int noutput_items, gr_vector_const_void_star &input_items, gr_vector_void_star &output_items)`

  Es la función donde se realiza el procesamiento, lee la entrada y genera la salida. Utiliza los siguientes parámetros:

  - noutput_items: Cantidad de salidas a generar
  - &ninput_items: Puntero que indica cantidad de entradas
  - &input_items: Puntero a señal de entrada
  - &output_items: Puntero a señal de salida

  Generalmente se realiza un *for* para iterar *noutput_items* veces para generar dicha cantidad de salidas. Por ejemplo:

```
// Make noutput_items
for(int i = 0; i < (noutput_items); i++) {
    ...
    out[i] = function(in[i]);
    ...
}// end for
```

  La función utiliza varias variables locales, algunas en particular se acumulan. La función retorna la cantidad de salidas generadas por cada pasada, la cual se indica mediante:

```
return n_outs
```

Siendo *n_outs* la cantidad de salidas generadas, en este momento se garantiza que se generaron *n_outs* salidas que contienen datos válidos.

- `consume_each`

  La función permite indicar la cantidad de entradas procesadas y de esta forma actualizar el puntero a nuevos valores de entradas.

C.3. Quartus

Se utilizó la herramienta *Quartus II Web edition* para el desarrollo en VHDL del FPGA [5]. En la Figura C.1 se observa una impresión de pantalla del ambiente de desarrollo.

La versión utilizada fue:
Versión: 15.0.0
Build: 145
Fecha: 22/04/2015
SJ Web Edition
SO: Ubuntu 14.04 LTS

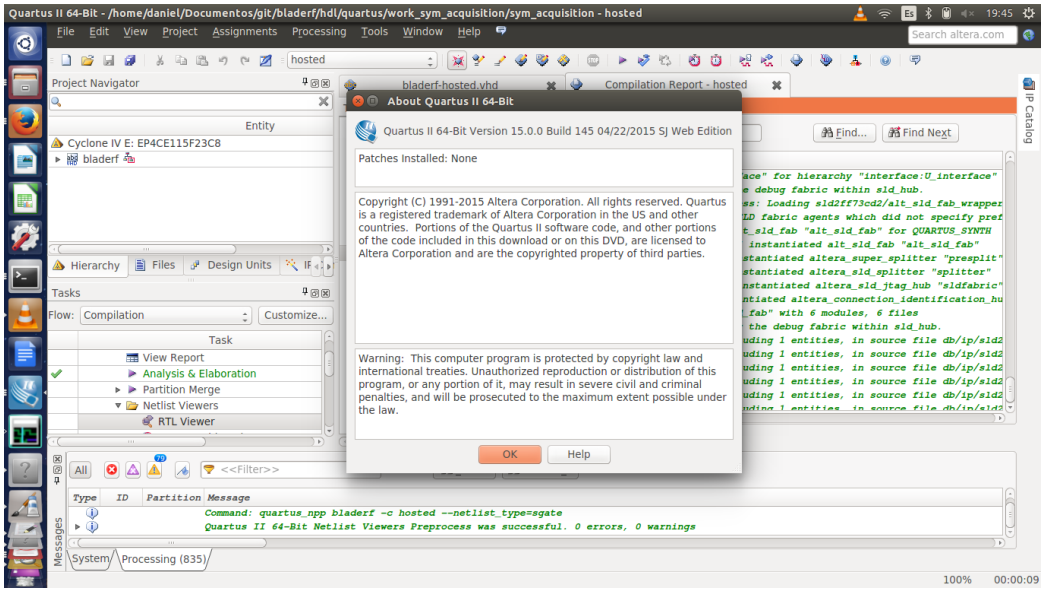


Figura C.1: Ambiente de desarrollo de Quartus V. 15.0.0

C.4. ModelSim

Se utilizó la herramienta *ModelSim Altera* para la simulación de la mayoría de los diseños realizados en VHDL [4]. En la Figura C.2 se observa una impresión de

pantalla del ambiente de desarrollo.

La versión utilizada fue:
ModelSim Altera Starter Edition 10.3d
Revisión: 2014.10
Fecha: 07/10/2014
SO: Ubuntu 14.04 LTS

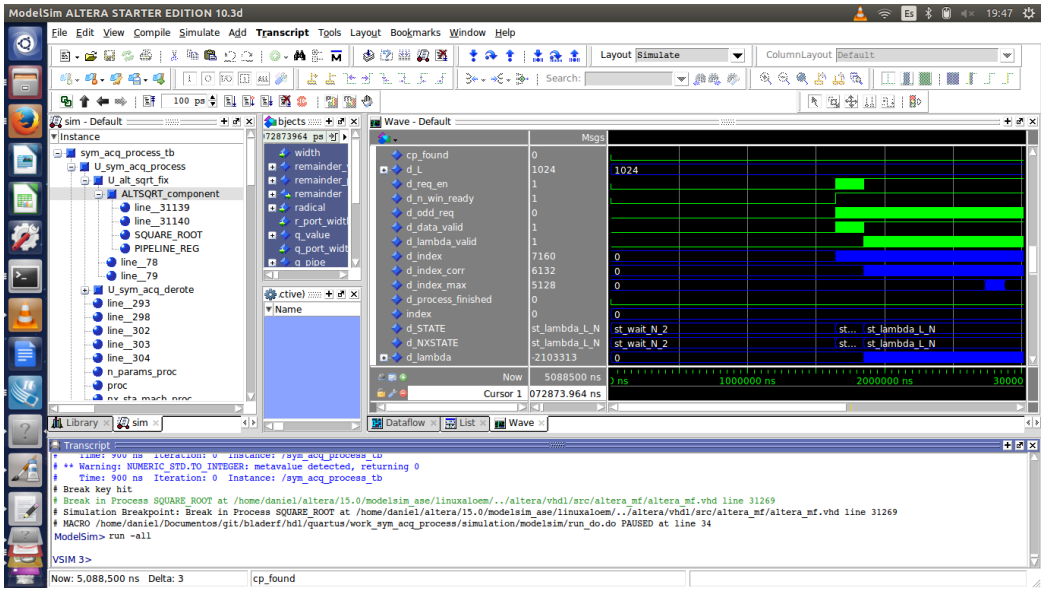


Figura C.2: Ambiente de desarrollo del ModelSim Altera Starter Edition

C.5. Octave

Herramienta de alto nivel para el cálculo numérico computacional [26]. Se utilizó la versión 3.8.1 en Ubuntu 14.04 LTS.

Apéndice D

Sync And Channel Estimation

Una vez obtenidas las 8192 muestras correspondientes a un símbolo corregidas en frecuencia, se calcula la FFT y por ende se pasa al dominio de la frecuencia. El bloque `Sync And Channel Estimation` es el encargado de corregir el *offset* entero en frecuencia de la señal recibida y de realizar una estimación lineal del canal para luego compensar en frecuencia las portadoras.

Para la estimación del *offset* entero en frecuencia [18] (de ahora en más Δf_I) se utilizan las portadoras TMCC que como ya se mencionó, tienen una ubicación fija $T[i] : i \, \epsilon \, \{0, \cdots, M-1\}$ (para el modo 3, $M = 52$). Para cada símbolo OFDM, el valor de estas es $w(T[i])$, el cual varía de acuerdo a una secuencia pseudo aleatoria conocida y a la modulación diferencial DBPSK. Esto significa que, dependiendo de la ubicación en cada símbolo toman el valor $4/3$ o $-4/3$. En este sentido, la siguiente correlación es maximizada cuando $m = \Delta f_I$.

$$\Gamma[m] = \sum_{i=0}^{M-2} w(T[i])Y[T[i]+m].w(T[i+i])Y^*[T[i+1]+m], \qquad \text{(D.1)}$$

con Y[k] la salida de la FFT.

Una vez identificadas las portadoras, se estima el canal y se realiza la ecualización. Esto es, si $X[i]$ es el símbolo transmitido e $Y[i]$ la señal recibida, se tiene que $Y[i] = X[i]H[i]$, siendo $H[i]$ la ganancia del canal para la portadora i-ésima. Si se logra estimar H[i] para todas las portadoras, la corrección se convierte en una trivial división. Para estimar el canal, se utilizan los *scattered pilots* (SP) ya que tienen valores y ubicaciones predefinidas. Un algoritmo similar al presentado para la búsqueda de los TMCCs es implementado para los SP, donde hay que tener en cuenta que los mismos pudeen variar entre cuatro posibilidades cíclicas. El valor para el resto es estimado por interpolación lineal o cuadrática.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice E

Diseño en el FPGA

E.1. Diseño del bloque `sdr_en_fpga_filter`

E.1.1. Interfaz del bloque `sdr_en_fpga_filter`

En la Figura E.1 se observa la interfaz del bloque `sdr_en_fpga_filter`.

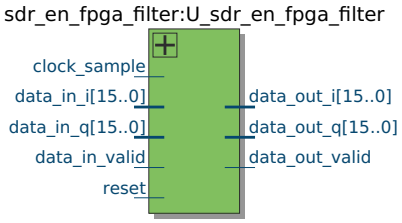


Figura E.1: Interfaz del bloque `sdr_en_fpga_filter`.

Control

- `clock_sample`:
  Reloj de entrada del bloque, corresponde al reloj de muestreo de la señal.
- `reset`:
  Reset del bloque, activo por nivel alto.

Puertos de entrada y salida

- `data_in_i[DATA_WIDTH-1..0]`:
  Entrada de la muestra I.
- `data_in_q[DATA_WIDTH-1..0]`:
  Ídem muestra Q.
- `data_in_valid`:
  Señal que indica muestras válidas en las entradas.

- `data_out_valid`:
  Ídem para las muestras de salida.
- `data_out_i[DATA_WIDTH-1..0]`:
  Salida de la muestra I.
- `data_out_q[DATA_WIDTH-1..0]`:
  Ídem muestra Q.

E.1.2. Diseño de los parámetros del filtro

Para el diseño del filtro pasa bajos requerido se determinaron los coeficientes mediante el uso de una herramienta de diseño de filtros, indicando el *ripple* de la banda pasante y el de la banda atenuante. Para el mismo se utilizaron 17 coeficientes. El filtro fue implementado para atenuar el contenido de los canales adyacentes, este se debe a que se utilizó una frecuencia de muestreo ($f_s = 512/63$ MHz $\approx$ 8,13 MHz) mayor a la mínima necesaria. En la Figura E.2 se observa la respuesta en frecuencia del filtro diseñado, marcando los *ripples* en cada banda. La banda pasante de determinó que sea de límite $f_p = 2{,}4$ MHz y la banda atenuante $f_t = 3{,}0$ MHz , aún teniendo en cuenta que se podía atenuar parte de la señal útil.

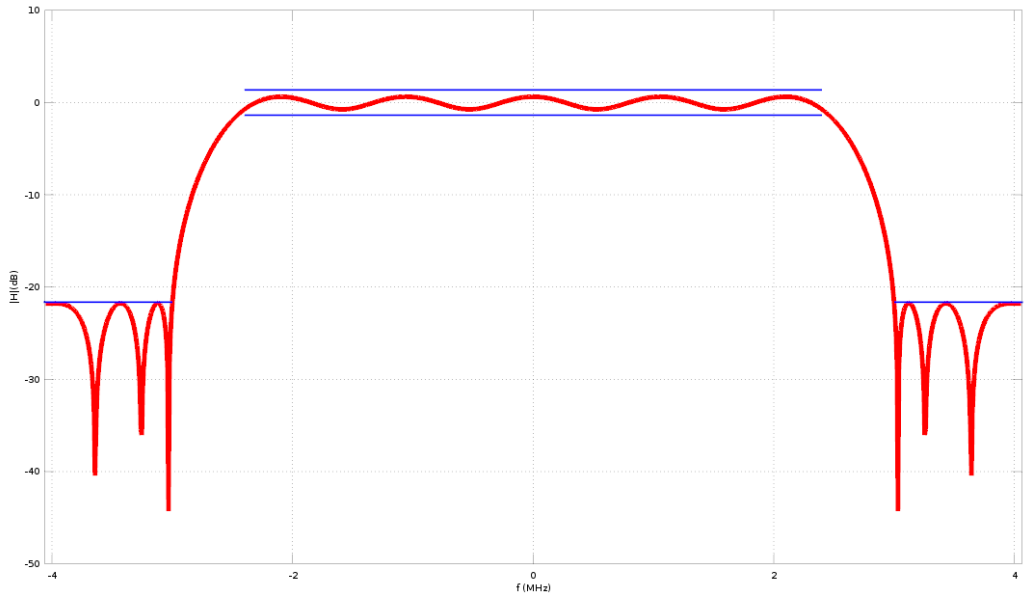


Figura E.2: Respuesta en frecuencia del filtro pasa bajos `sdr_en_fpga_filter`.

E.1.3. Validación

El funcionamiento del bloque fue verificado mediante el filtrado de una señal de referencia. La señal filtrada se utilizó como entrada de un *flowgraph* de GNU Radio y se verificó que la señal resultante estaba filtrada. Luego el bloque fue incluido a

la arquitectura del FPGA. Las pruebas en hardware del bloque fueron realizadas junto al bloque sym_acquisition. Las mismas se presentan en el Capítulo 4.

E.2. Diseño del bloque interface

En el bloque interface el largo de los símbolos está determinado por la entrada mode correspondiente al modo de transmisión de los mismos.

El bloque cuenta con un puerto para entrada de datos y otro para la salida. Estos se encuentran conectados directamente. En el estado de reposo, el mismo aguarda por el aviso de que se encuentra disponible un símbolo para ser leído. Este está indicado por la entrada symbol_ready. Inmediatamente el bloque interface comienza con la solicitud de lectura de muestras del símbolo. Mediante la salida symb_reading_out avisa que se encuentra leyendo las muestras del símbolo.

La solicitud de lectura la realiza mediante la salida data_out_re que será activa en todos los flancos de reloj, de forma de realizar la lectura en el menor tiempo posible. Luego, cada muestra válida que ingrese en respuesta a la solicitud será entregada a la salida. En caso de haber errores durante la copia de algún símbolo, se indicará mediante la entrada out_of_sync de forma que pueda *resetear* la señalización del fifo_writer mediante el puerto reset_fifo.

E.2.1. Interfaz del bloque interface

En la Figura E.3 se observa la interfaz del bloque.

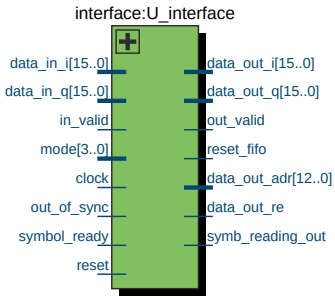


Figura E.3: Interfaz del bloque interface.

A continuación se describe cada una de las entradas y salidas del bloque, agrupadas según su función:

Control

- clock:
  Reloj de entrada del bloque, corresponde al reloj de muestreo de la señal.
- reset:
  Reset del bloque, activo por nivel alto.

Interfaces de entrada y salida

- `data_in_i[DATA_WIDTH-1..0]`:
  Entrada de la muestra I.
- `data_in_q[DATA_WIDTH-1..0]`:
  Ídem muestra Q.
- `in_valid`:
  Señal que indica muestras válidas en las entradas.
- `out_valid`:
  Ídem para las muestras de salida.
- `data_out_i[DATA_WIDTH-1..0]`:
  Salida de la muestra I.
- `data_out_q[DATA_WIDTH-1..0]`:
  Ídem muestra Q.

Puertos de control de lectura y escritura

- `mode[3..0]`:
  Señal de entrada que indica el modo de transmisión de los símbolos en notación entera sin signo.
- `symbol_ready`:
  Señal que indica si hay un símbolo pronto para ser leído en la entrada.
- `data_out_re`:
  Señal para solicitud de lectura de muestras del símbolos.
- `data_out_adr[ADDR_WIDTH-1..0]`:
  Señal de salida para solicitar determinada muestra del símbolo.
- `out_of_sync`:
  Señal que indica si la transmisión de muestras está des-sincronizada.
- `reset_fifo`:
  Puerto de salida para indicar reseteo de la señalización del `fifo_writer`.

E.2.2. Validación

Para la validación del bloque se realizó un *test-bench* que contiene un bloque generador de símbolos de forma que el `interface` lea el contenido de los mismos y los copie al bloque `fifo_writer`. Los datos de los símbolos corresponden a un contador almacenado en una memoria.

Luego, el mismo diseño fue incluido en la arquitectura del FPGA con el `fifo_writer` modificado. Se realizaron pruebas de recepción de los símbolos generados en el FPGA correspondientes al contenido de una memoria (contador). Las señales

recibidas fueron comparadas contra el contenido esperado verificando que las mismas contenían los valores correspondientes al contador. Para ello se utilizó el bloque de GNU Radio `messages2symbol` para extraer las muestras de los mensajes y verificar mediante un *script* de Octave que los datos eran correctos.

En el marco de estas pruebas se verificó que el tiempo máximo de pausas en el envío de muestras desde el FPGA hacia la PC era del orden de 2 segundos. También se encontraron problemas con pérdidas de muestras en el camino hacia la PC. Esto fue posible gracias al `messages2symbol` que cuenta con la posibilidad de verificar los saltos entre los *timestamps*. En la sección siguiente se explicarán las pruebas extras realizadas para encontrar el problema de pérdidas de muestras.

E.3. Diseño del bloque sym_acquisition

En la Figura E.4 se muestran los bloques internos del sym_acquisition y algunas de sus conexiones.

E.3.1. Especificación y Requerimientos del sym_acquisition

Reloj de muestreo y procesamiento

- El reloj de funcionamiento del bloque será el mismo que el de muestreo de la señal (en la aplicación se utilizará un reloj de $2 * f_s$, con $f_s = 512/63$ MHz $\approx$ 8,13 MHz). Este genera una nueva muestra cada dos flancos de reloj. En la etapa de definición se consideró utilizar un reloj más rápido de 38,4 MHz utilizado por el nios_system sin embargo esto fue descartado debido al algoritmo de procesamiento definido. El algoritmo fue definido para que funcione específicamente a la frecuencia de muestreo.

Objetivos del bloque

- Objetivo intermedio: sincronización temporal mediante el algoritmo de máxima verosimilitud que utiliza la función de correlación (Λ) para hallar el CP duplicado de cada símbolo, dentro del rango de datos procesados.

- Objetivo final: corrección de desfasaje fraccional en frecuencia y salida de forma serial del símbolo, esto es, se corrige y entrega cada muestra a la salida. Para ello, se debe hallar el valor de $\angle\gamma(\theta)$ ($\gamma(\theta)$ es un resultado intermedio del punto anterior y en particular del cálculo de la función Λ) y aplicarle el fasor de corrección para cada muestra hallado a partir del $\angle\gamma(\theta)$.

Algoritmo para la sincronización temporal: presentación general

- Se creó un algoritmo recursivo para lograr la sincronización mediante una función cuyo objetivo principal es que lleve el menor tiempo posible de cálculo. Se recuerda al lector que se partió del análisis de la Sección 3.2.

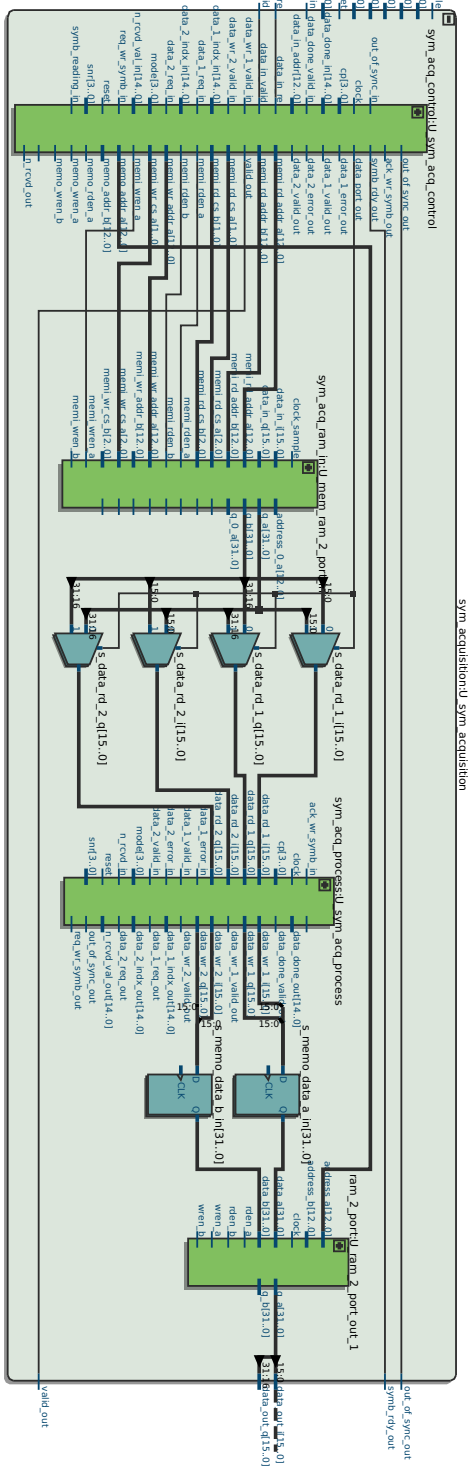


Figura E.4: Arquitectura interna del bloque `sym_acquisition`.

- El algoritmo tiene como objetivo calcular los valores de la función Λ, esto lo hace de forma recursiva lo que implica realizar una doble lectura de las muestras de la memoria de entrada a una cadencia igual a la frecuencia de muestreo.
- Λ se calcula en función de γ y Φ, los cuales corresponden a sumatorias recursivas. Cada elemento de las funciones γ y Φ debe ser hallado dos veces, ya que en un momento se utiliza como nuevo término a agregar a la suma y luego es necesario restar su valor debido a que ya no debe integrar la misma. Esto se explicará en más detalle en la Sección E.3.5.3. De esta forma se evita el uso de memorias de gran tamaño que almacenen todos los sumandos para cada sumatoria. Si fuera el caso, se deberían almacenar $L = 2048$ sumandos con buena precisión (correspondientes a los sumandos de γ y Φ) generando un gran consumo de memoria dentro del bloque.
- Estas cuentas se realizarán en dos rangos dependiendo del estado del bloque: des-sincronizado (N valores de Λ en una ventana de $2N + L$ muestras) o sincronizado (16 valores de Λ en una ventana de $N + L + 16$ muestras).

Memoria de Entrada

- Se definió un bloque de memoria compuesto por cuatro memorias ram 2-port de Altera de largo 8192 y ancho 32 *bits*.

Memoria de Salida

- Se utilizó una memoria ram 2-port de Altera de 8192 y ancho 32 *bits*.

Procesamiento - Tiempos

- Se debe sincronizar con la mayor cantidad de símbolos de la entrada, es decir, se debe detectar la mayor cantidad de símbolos presentes sobre la señal. En caso de no poder hacerlo con alguno, deberá contar con mecanismos de aviso para evitar generar grandes tiempos de latencia en la cadena de datos.

Interfaces de entrada/salida

- Las interfaces de entrada y salida del bloque cuentan con dos puertos para las cada componente de las muestras (I y Q) y un puerto de validación.
- Cada muestra I o Q está compuesta por la cantidad de *bits* indicados por el parámetro `DATA_WIDTH` el cual se podrá configurar de acuerdo a la señal de entrada (en la implementación se utilizarán 16 *bits*). Las mismas están representadas en complemento a dos.

Parámetros del procesamiento

- Los parámetros: modo, CP y SNR, serán indicados mediante los puertos `mode`, `cp` y `snr`, respectivamente.

E.3.2. Interfaz del bloque `sym_acquisition`

En la Figura E.5 se observa la interfaz del bloque `sym_acquisition`. A continuación se describe cada una de las entradas y salidas del bloque, agrupadas según su función:

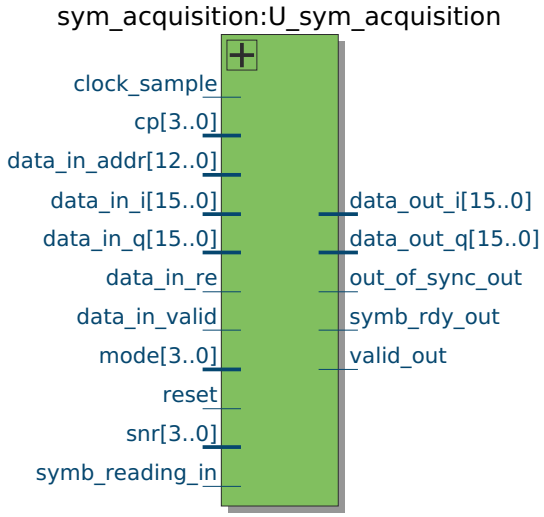


Figura E.5: Interfaz del bloque `sym_acquisition`.

Control

- `clock_sample`:
  Reloj de entrada del bloque, corresponde al reloj de muestreo de la señal.

- `reset`:
  Reset del bloque activo por nivel alto.

Puertos de comunicación con el bloque `sdr_en_fpga_filter`

- `data_in_i[DATA_WIDTH-1..0]`:
  Puerto de entrada con el valor I de la muestra actual, representado en complemento a dos.

- `data_in_q[DATA_WIDTH-1..0]`:
  Puerto de entrada con el valor Q de la muestra actual, representado en complemento a dos.

- `data_in_valid`:
  Señal de entrada que indica si las muestras en los dos puertos anteriores son válidas.

Puertos de comunicación con el bloque interface

- data_out_i[DATA_WIDTH-1..0]:
  Puerto de salida con el valor I de la muestra a enviar aguas abajo, representado en complemento a dos.

- data_out_q[DATA_WIDTH-1..0]:
  Puerto de salida con el valor Q de la muestra a enviar aguas abajo, representado en complemento a dos.

- valid_out:
  Puerto de salida que indica si los datos de los puertos anteriores son válidos.

- out_of_sync_out:
  Puerto de salida que indica que el bloque sym_acquisition se des-sincronizó requiriendo una acción aguas abajo.

- symb_rdy_out:
  Puerto de salida que indica que hay un símbolo pronto.

- symb_reading_in:
  Puerto de entrada que indica que se está leyendo un símbolo.

- data_in_addr[ADDR_WIDTH-1..0]:
  Puerto de entrada que indica qué índice de los datos se quiere leer del bloque sym_acquisition.

- data_in_re:
  Puerto de entrada que al tomar el valor lógico 1 solicita una muestra.

Parámetros del procesamiento

- mode[3..0]:
  Señal de entrada que indica el modo de transmisión de los símbolos en notación entera sin signo.

- cp[3..0]:
  Señal de entrada que indica el largo del CP. El valor satisface la siguiente ecuación: $CP = \frac{1}{2^{\text{cp}}}$, con cp representado en entero sin signo.

- snr[3..0]:
  Señal de entrada que indica el valor de SNR de la señal en dB, el mismo está representado en entero sin signo.

Como se mencionó en el Capítulo 5, la arquitectura interna del sym_acquisition es del tipo estructural (ver Figura 5.2). A continuación se describen en detalle los bloques internos del sym_acquisition.

E.3.3. Diseño del bloque `mem_ram_2_port_in`

El bloque `mem_ram_2_port_in` fue diseñado con el objetivo de contar con un *buffer* para las muestras de entrada que fuera de un tamaño adecuado para los requerimientos del sistema y suficientemente versátil como para que se pudiera leer y escribir a la vez.

La primera solución planteada para estos requisitos fue una memoria ram de 4 puertos, esto implica que pueda ser leída o escrita por cuatro puertos a la vez. No se pudo lograr una implementación de estas características. Se buscaron alternativas y se terminó optando por una memoria compuesta por 4 memorias ram de 2 puertos de 8192 muestras de largo y 32 *bits* de ancho.

Para poder manejar de manera eficiente estas cuatro memorias se implementaron una serie de bloques auxiliares. Los mismos consistieron en multiplexores y decodificadores de 1 *bit* o de vectores de *bits*.

Desde un punto de vista externo, el bloque `mem_ram_2_port_in` se puede ver como una memoria de 2 puertos de 32 K *bytes* de 32 *bits* de datos que cuenta con puertos de entrada de datos para I y Q y la señal de reloj, puertos de direcciones para la lectura de los puertos A y B y para el sub-bloque de memoria, además de la señal de habilitación para la lectura y para la escritura.

E.3.3.1. Especificación y Requerimientos

Reloj

- El reloj de funcionamiento del bloque será el mismo que el de muestreo de la señal (en la aplicación se utilizará un reloj de $2 * f_s$, con $f_s = 512/63$ MHz $\approx$ 8,13 MHz). El mismo genera una nueva muestra cada dos flancos de reloj. Para el caso de lectura la misma se puede realizar a la cadencia del reloj de muestreo.

Tamaño - cantidad de muestras a almacenar

- El bloque deberá ser suficientemente grande como para almacenar las muestras necesarias para que el procesamiento funcione de manera continua.

Arquitectura eficiente

- Deberá permitir el acceso a memoria de escritura y lectura a la vez. En este sentido, deberá contemplar el peor caso: escritura por el puerto A y lectura por el puerto A de otra memoria interna o *chip*.
- Solo será necesario que cuente con la escritura de datos por un solo puerto (el A).
- En caso de que se requiera leer y escribir por el mismo puerto se priorizará la escritura.
- Siempre se llevará a cabo una escritura en la memoria interna que corresponda. La misma será de manera cíclica.

- La lectura podrá ser de dos muestras consecutivas o muchas veces podrá ser de muestras que difieran N.
- En el caso de lectura de dos muestras consecutivas en una misma memoria interna sólo podrá realizarse si no se está escribiendo en la misma.

E.3.3.2. Interfaz

En la presente sección se describe la interfaz de la memoria de entrada (ver Figura E.6).

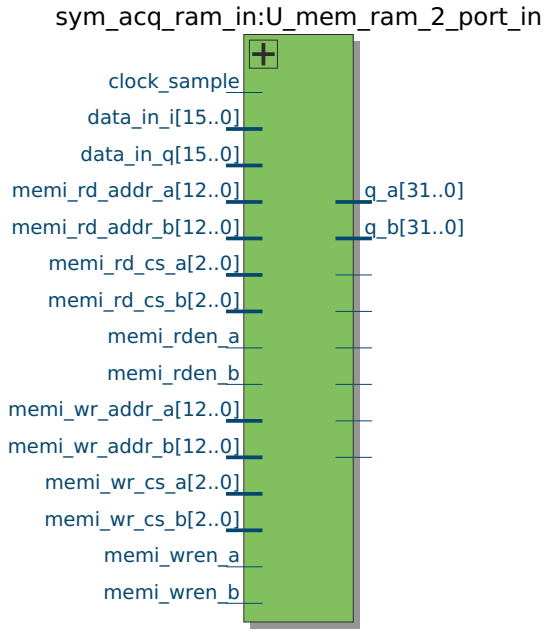


Figura E.6: Interfaz del bloque mem_ram_2_port_in.

Reloj

- clock_sample:
  Reloj de entrada del bloque, corresponde al reloj de muestreo de la señal.

Puertos de entrada de datos

- data_in_i[DATA_WIDTH-1..0]:
  Puerto de entrada con el valor I de la muestra a guardar, representado en complemento a dos.
- data_in_q[DATA_WIDTH-1..0]:
  Puerto de entrada con el valor Q de la muestra a guardar, representado en complemento a dos.

Puertos de salida de datos

- q_a[DATA_WIDTH*2-1..0]:
  Puerto de salida A con el valor Q en la parte alta e I en la parte baja de la muestra leída, representado en complemento a dos.
- q_b[DATA_WIDTH*2-1..0]:
  Análogo al puerto A.

Lectura de datos por puerto A

- memi_rd_addr_a[ADDR_WIDTH-1..0]:
  Puerto de entrada con dirección de memoria para lectura de datos por el puerto A.
- memi_rd_cs_a[CS_WIDTH-1..0]:
  Puerto de entrada con dirección de la memoria o *chip* para lectura de datos por el puerto A.
- memi_rden_a:
  Entrada de habilitación de la lectura de datos por el puerto A.

Lectura de datos por puerto B

Análogo al puerto A.

- memi_rd_addr_b[ADDR_WIDTH-1..0]
- memi_rd_cs_b[CS_WIDTH-1..0]
- memi_rden_b

Escritura de datos por puerto A

- memi_wr_addr_a[ADDR_WIDTH-1..0]:
  Puerto de entrada con dirección de memoria para escritura de datos por el puerto A.
- memi_wr_cs_a[CS_WIDTH-1..0]:
  Puerto de entrada con dirección de la memoria o *chip* para escritura de datos por el puerto A.
- memi_wren_a:
  Entrada de habilitación de la escritura de datos por el puerto A.

Escritura de datos por puerto B

Análogo al puerto A.

- memi_wr_addr_b[ADDR_WIDTH-1..0]
- memi_wr_cs_b[CS_WIDTH-1..0]
- memi_wren_b

E.3.3.3. Funcionamiento

En la Figura E.7 se presenta la arquitectura interna del bloque mem_ram_2_port_in.

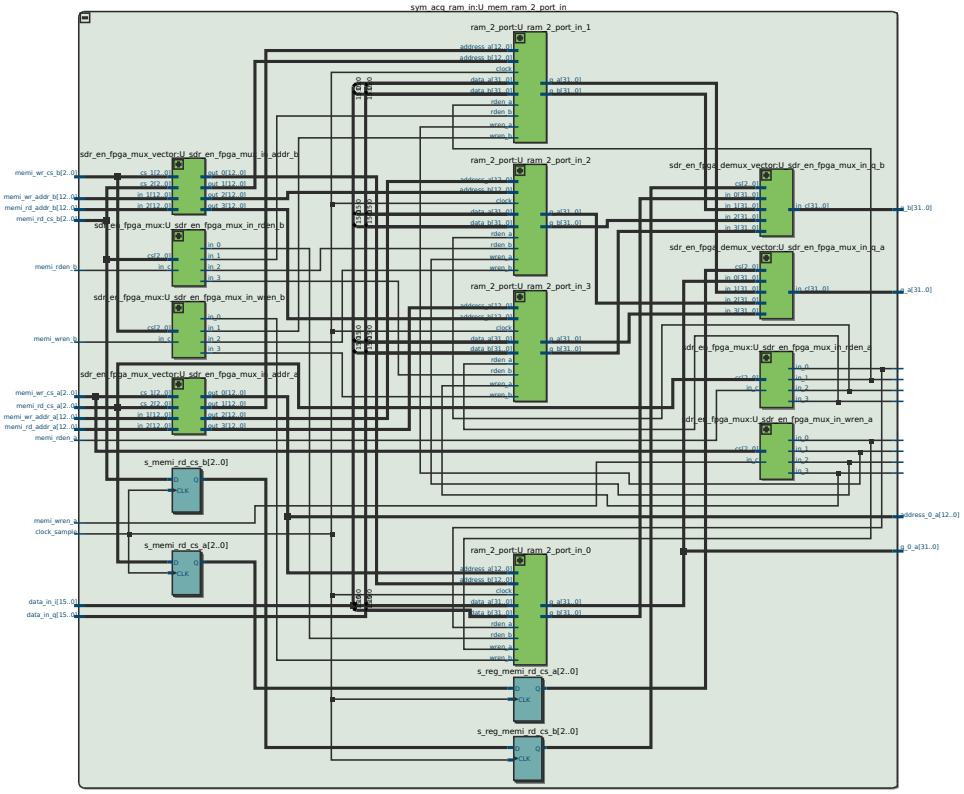


Figura E.7: Arquitectura interna del bloque mem_ram_2_port_in.

El bloque mem_ram_2_port_in se comporta como una memoria grande de largo 32768 datos. El valor que toma la señal cs indica a qué memoria interna se escribirá o se leerá. En definitiva, esta señal puede interpretarse como la parte alta de las direcciones de memoria.

En caso de solicitarse lectura y escritura del mismo puerto a la vez, se prioriza la escritura.

En los siguientes párrafos se explica cada uno de los bloques internos.

ram_2_port_in Se utilizaron cuatro instancias de la memoria ram de 2 puertos de Altera [2]. La misma consta de dos puertos de lectura y dos de escritura. Cada uno identificado como A o B. Podrán realizarse dos lecturas a la vez, dos escrituras o una lectura y una escritura por puertos distintos.

sdr_en_fpga_mux Este bloque es un multiplexor, según el valor de cs[CS_WIDTH-1..0] mapeará a la salida correspondiente, la entrada en in_c. Se utilizan cuatro instancias. Dos para las señales de habilitación para escritura y lectura del puerto A y otras dos análogas para el puerto B. La señal de selección en cada caso es la de cs de cada puerto correspondiente a lectura o escritura según corresponda.

sdr_en_fpga_mux_vector Este bloque es un multiplexor, que según el valor de sus puertos cs_1[CS_WIDTH-1..0] y cs_2[CS_WIDTH-1..0] mapea a su salida las señales vectoriales en su entrada. Se utilizarán dos instancias para la selección de las direcciones del puerto A y B tanto para lectura como para escritura.

sdr_en_fpga_demux_vector Este bloque es un decodificador. Copia a la salida la entrada según la señal de selección cs[CS_WIDTH-1..0]. Se utilizan dos instancias para la salida de las memorias q en los puertos A y B.

E.3.3.4. Validación

Simulaciones Se verificó mediante una simulación el comportamiento esperado del bloque. Debido a la simplicidad de mismo no se agregarán capturas de pantalla de esto.

Pruebas en *Hardware* Las pruebas en *hardware* se realizaron junto al resto de los bloques del sym_acquisition y se presentan en el Capítulo 6.

E.3.4. Diseño del bloque sym_acq_control

E.3.4.1. Especificación y Requerimientos

Control

- El reloj de funcionamiento del bloque será el mismo que el de muestreo de la señal (en la aplicación se utilizará un reloj de $2 * f_s$, con $f_s = 512/63$ MHz $\approx 8{,}13$ MHz). El mismo genera una nueva muestra cada dos flancos de reloj.
- El bloque deberá contar con un *reset* activo por nivel alto para llevarlo a un estado inicial conocido.

Parámetros del procesamiento

- Los parámetros: modo, CP y SNR, serán indicados mediante los puertos mode, cp y snr, respectivamente.

Recepción de muestras

- El bloque será responsable de la recepción de muestras del sym_acquisition. Para ello generará las señales de control adecuadas para la escritura en la memoria de entrada mem_ram_2_port_in.

Lectura de muestras recibidas para su procesamiento

- Frente a una solicitud de disponibilidad de cierta cantidad de muestras n, deberá avisar cuando estén disponibles.
- El bloque de control deberá manejar la memoria de entrada para poder generar las señales de control de manera eficiente para la lectura de muestras del bloque mem_ram_2_port_in.
- Deberá tener en cuenta que la mayoría de las lecturas diferirán en N muestras.
- En caso de que se soliciten dos muestras de una memoria que está siendo escrita, deberá avisar adecuadamente que una de las lecturas no podrá llevarse a cabo.

Muestras útiles

- Será responsable de la actualización de los punteros a memoria (bajo y alto) que indicarán el comienzo y el fin de muestras útiles en la memoria de entrada.
- Además ajustará continuamente la cantidad de muestras útiles lo cual dependerá de las muestras que entran a la memoria y las que son descartadas. Este último dato será una entrada del bloque.

Comunicaciones con bloques aguas abajo

- El bloque deberá llevar a cabo la función de comunicación con el bloque aguas abajo que continúe el procesamiento. Para ello se implementará un protocolo de comunicaciones que indique que hay un símbolo pronto y genere las señales pertinentes para el envío de datos.
- Deberá manejar las señales de control de lectura de la memoria de salida de manera adecuada para cumplir con esta función.

Fuera de sincronismo

- Generará la señal de fuera de sincronismo enviando directamente la recibida por parte del bloque de procesamiento.

E.3.4.2. Interfaz

A continuación se exponen los puertos de entrada y salida del sym_acq_control agrupados según sus funciones.

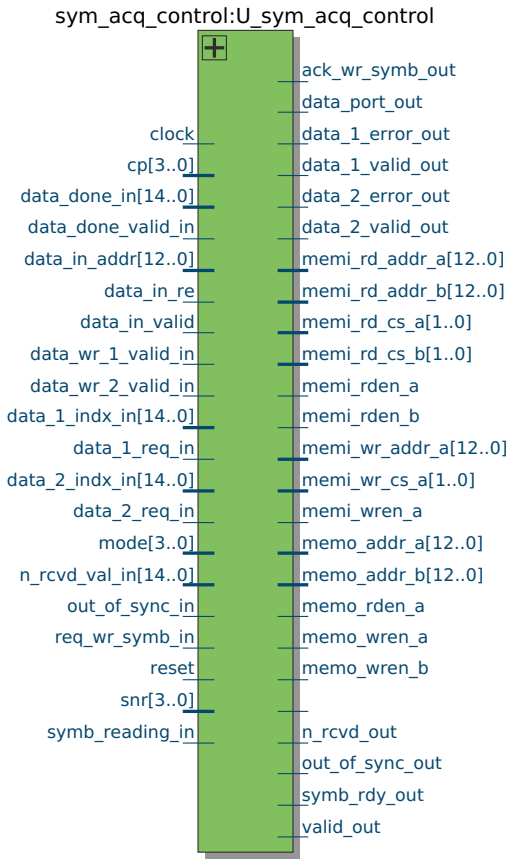


Figura E.8: Interfaz del bloque `sym_acq_control`.

Control

- `clock`:
  Reloj de entrada del bloque, corresponde al reloj de muestreo de la señal.
- `reset`:
  Reset del bloque, activo por nivel alto.
- `data_in_valid`:
  Puerto de entrada que indica muestra válida en la entrada del bloque `sym_acquisition`. Este toma el valor lógico de 1 cada dos períodos de reloj.

Control de escritura en la memoria de entrada

- `memi_wr_cs_a[CS_WIDTH-1..0]`:
  Puerto de salida que indica a qué memoria se va a direccionar la escritura por el puerto A. Toma valores del tipo entero sin signo.
- `memi_wren_a`:
  Puerto de salida que indica habilitación de la escritura por el puerto A.

- memi_wr_addr_a[ADDR_WIDTH-1..0]:
  Puerto de salida que indica en qué dirección se va a realizar la escritura en el puerto A. Toma valores del tipo entero sin signo.

Control de lectura de la memoria de entrada - hacia memoria

- memi_rd_cs_a[CS_WIDTH-1..0]:
  Puerto de salida que indica a qué memoria interna se va a direccionar la lectura por el puerto A. Toma valores del tipo entero sin signo.
- memi_rden_a:
  Puerto de salida que indica habilitación de lectura por el puerto A.
- memi_rd_addr_a[ADDR_WIDTH-1..0]:
  Puerto de salida que indica de qué dirección se va a realizar la lectura en el puerto A. Toma valores del tipo entero sin signo.
- memi_rd_cs_b[CS_WIDTH-1..0]:
  Análogo al puerto A.
- memi_rden_b:
  Análogo al puerto A.
- memi_rd_addr_b[ADDR_WIDTH-1..0]:
  Análogo al puerto A.

Control de lectura de la memoria de entrada - hacia bloque de procesamiento

- n_rcvd_val_in[ADDR_TOTAL_WIDTH-1..0]:
  Puerto de entrada que indica la cantidad de muestras solicitadas para ser leídas. El valor se encuentra representado en entero sin signo.
- n_rcvd_out:
  Salida de reconocimiento (confirmación) que indica si la cantidad de muestras solicitadas por el puerto n_rcvd_val_in están disponibles.
- data_1_req_in:
  Puerto de entrada para indicar solicitud de lectura.
- data_1_indx_in[ADDR_TOTAL_WIDTH-1..0]:
  Entrada que indica qué índice de las muestras solicitadas por el puerto n_rcvd_val_in se desea leer. Toma valores del tipo entero sin signo.
- data_1_error_out:
  Puerto de salida que indica error en caso de que no se hubiera podido realizar la lectura solicitada.
- data_1_valid_out:
  Salida de validación de la muestra entregada por el puerto 1.

- data_2_req_in:
  Análogo al puerto 1.
- data_2_indx_in[ADDR_TOTAL_WIDTH-1..0]:
  Análogo al puerto 1.
- data_2_error_out:
  Análogo al puerto 1.
- data_2_valid_out:
  Análogo al puerto 1.
- data_port_out:
  Puerto de salida para selección de puerto 1 o 2 para el caso de lectura de memoria de entrada con puerto A en uso.

Control de escritura de la memoria de salida - desde bloque de procesamiento

- req_wr_symb_in:
  Puerto de entrada que indica solicitud de escritura de muestras del símbolo procesado.
- ack_wr_symb_out:
  Salida para indicar que se puede comenzar a escribir la memoria de salida.
- data_wr_1_valid_in:
  Entrada para indicar escritura de una muestra por el puerto 1.
- data_wr_2_valid_in:
  Análogo al puerto 1.

Control de escritura de la memoria de salida - hacia memorias

- memo_wren_a:
  Puerto de salida para indicar escritura en memoria por el puerto A.
- memo_wren_b:
  Análogo al puerto A.

Control de lectura de la memoria de salida - hacia bloque interfaz

- symb_rdy_out:
  Puerto de salida que indica que hay un símbolo procesado para enviar aguas abajo.
- symb_reading_in:
  Puerto de entrada de reconocimiento de la señal symb_rdy_out.
- data_in_re:
  Puerto de entrada que indica solicitud de lectura de muestra del símbolo pronto.

- valid_out:
  Señal de validación de muestras en la salida del bloque sym_acquisition.

- data_in_addr[ADDR_WIDTH-1..0]:
  Puerto de entrada que indica el índice de lectura del símbolo pronto. Toma valores del tipo entero sin signo.

Control de lectura de la memoria de salida - hacia memorias

- memo_rden_a:
  Puerto de salida para indicar lectura en memoria por el puerto A.

Control de lectura y escritura de la memoria de salida

- memo_addr_a[ADDR_WIDTH-1..0]:
  Puerto de salida para indicar la dirección a leer en memoria por el puerto A. Toma valores del tipo entero sin signo.

- memo_addr_b[ADDR_WIDTH-1..0]:
  Análogo al puerto A.

Control de datos útiles en memoria de entrada

- data_done_in[ADDR_TOTAL_WIDTH-1..0]:
  Puerto de entrada que indica la cantidad de muestras a descartar de la memoria de entrada. Toma valores del tipo entero sin signo.

- data_done_in_valid:
  Puerto de entrada que indica si los datos del puerto data_done_in son válidos.

Control de sincronismo

- out_of_sync_in:
  Puerto de entrada que indica si el bloque sym_acquisition se encuentra des-sincronizado.

- out_of_sync_out:
  Puerto de salida que indica si el bloque sym_acquisition se encuentra des-sincronizado.

Parámetros del procesamiento

- mode[3..0]:
  Señal de entrada que indica el modo de transmisión de los símbolos en notación entera sin signo.

- `cp[3..0]`:
  Señal de entrada que indica el largo del CP. El valor satisface la siguiente ecuación: $CP = \frac{1}{2^{cp}}$, con `cp` representado en entero sin signo.

- `snr[3..0]`:
  Señal de entrada que indica el valor de SNR de la señal en dB, el mismo está representado en entero sin signo.

E.3.4.3. Funcionamiento

De acuerdo a la especificación y los requerimientos descriptos anteriormente se implementaron varios procesos que llevan a cabo las funciones especificadas.

Escritura de la memoria de entrada La escritura en la memoria de entrada implica el manejo de las señales de habilitación, de dirección dentro de cada *chip* de memoria y de dirección a cada uno. Además requiere la correcta actualización de los punteros a memoria que indica cuáles son las muestras útiles y el cálculo continuo de esta variable.

Contador de muestras Se implementó un contador para generar las direcciones de escritura a memoria. El mismo recibe en su señal de habilitación la señal `data_in_valid`.

Proceso `write_memory_input` En este proceso se generan las señales de control para la escritura de las muestras en la entrada en el bloque de memoria `mem_ram_2_port_in`. La señal de habilitación, `memi_wren_a`, toma el valor de `data_in_valid` registrado. La dirección de la memoria a escribir es la salida del contador mencionado anteriormente. El valor de `memi_wr_cs_a` que indica en qué memoria interna o *chip* se va a escribir es calculado de acuerdo a una señal auxiliar generada por el contador que indica el fin de la cuenta.

El proceso `write_memory_input` es sensible al reloj y tiene máxima prioridad. Esto implica que el mismo funcionará indefinidamente luego de un *reset*. La escritura a memoria se realiza cíclicamente por lo tanto, una vez que se haya completado se comienza a escribir nuevamente desde la dirección 0.

Además, en este proceso, se actualizan los valores de las señales auxiliares (`high_point.chip`, `high_point.addr`) para llevar la cuenta de señales útiles disponibles. Esto es explicado en el siguiente párrafo.

Proceso `math_useful_samples` Este proceso calcula para cada período de reloj la cantidad de muestras útiles disponibles en la memoria. Para ello, cuenta con dos punteros a memoria definidos mediante un tipo nuevo en VHDL: `low_point` y `high_point`. Ambos punteros tienen un campo que hace referencia al *chip* y otro a la dirección, indicados por `.chip` y `.addr` respectivamente. La señal `high_point` es actualizada en el proceso de escritura en memoria como se explicó anteriormente y la señal `low_point` en el presente proceso.

En caso de existir una solicitud de descarte de muestras (data_done_valid_in en 1), la señal low_point es actualizada según el valor de data_done_in.

Para cada escritura en memoria indicada mediante la señal s_reg_data_in_valid, en caso de no existir solicitud de descarte de muestras, se calcula la señal s_useful_samples a partir de los punteros mencionados anteriormente.

Lectura de la memoria de entrada

Para la lectura de la memoria de entrada, se definió un protocolo en el cual mediante el puerto n_rcvd_val_in se solicita cierta cantidad de muestras y mediante el puerto n_rcvd_out se responde en caso afirmativo. Una vez pasada esta etapa, el bloque correspondiente podrá solicitar muestras a través de los puertos 1 y 2 para tal propósito. Para diferenciar la nomenclatura de las memorias internas referente a los puertos A y B se optó por usar 1 y 2 para los puertos de datos del bloque de procesamiento.

Proceso let_process_know_n_received_val En cada flanco de reloj, en caso de existir una solicitud de lectura de la memoria de entrada (n_rcvd_val_in $>$ 0), se verifica si se puede cumplir, chequeando la cantidad de muestras disponibles contra las solicitadas. En caso afirmativo, se indica mediante n_rcvd_out.

Proceso read_memory_input Se procesa la solicitud a lectura de memoria. El punto crítico del mismo es si se puede cumplir o no con la solicitud recibida. Esto se debe a que como las memorias internas están siendo escritas continuamente podría llegarse a generar una situación en la que se piden dos muestras de un mismo *chip* que está siendo escrito. En tal caso se prioriza la solicitud del puerto 1 y se da un aviso de error en el puerto 2. Para estos casos se utiliza una señal auxiliar data_port_out para direccionar adecuadamente los datos leídos de la memoria hacia el bloque que los solicitó (ver Figura E.4).

Escritura memoria de salida

Proceso write_memory_output La escritura en la memoria de salida se lleva a cabo cuando se recibe una solicitud en el puerto req_wr_symb_in. En caso de que no se esté realizando una lectura en memoria es aceptada. Inmediatamente se comienza el envío de datos hacia la memoria de salida. Para ello se cuenta con dos puertos llamados 1 y 2 que contienen las señales de habilitación.

Lectura memoria de salida

Proceso read_memory_output Una vez finalizada la escritura de un símbolo procesado en la memoria de salida se habilita la señal symb_rdy_out. En caso de que el bloque aguas abajo que la recibe se encuentre preparado, lo indica mediante symb_reading_in. Luego se realiza la lectura de la memoria de salida de manera análoga a la lectura de la memoria de entrada.

E.3.4.4. Validación

Para la validación del bloque, en primera instancia se implementó un *test-bench* que simulaba distintas situaciones y verificaba para algunas de las funciones si la salida del bloque era la adecuada. Algunas verificaciones se hicieron visualmente en el visor de ondas del ModelSim. En una segunda instancia se verificó su funcionamiento mediante un *test-bench* que instanciaba el resto de los bloques de la arquitectura del `sym_acquisition`. En ultima instancia se validó en el *hardware*. Estos últimos dos puntos de la validación se presentan en el Capítulo 4.

Simulaciones En las siguientes secciones se presentan las simulaciones de las funciones definidas anteriormente. A pesar de que las mismas fueron utilizadas para la validación del bloque aquí se presentan con el fin de clarificar el comportamiento de las señales internas del mismo. Se marca mediante una línea amarilla el comienzo de la función que se desea mostrar en cada caso o algún elemento de la misma a destacar.

Escritura memoria de entrada En la Figura E.9 se presenta la simulación para la escritura en la memoria de entrada. Se puede observar el comportamiento de las tres señales que manejan el puerto de escritura A: `memi_wren_a`, `memi_wr_cs_a` y `memi_wr_addr_a`.

Obsérvese que se marcó en amarillo cuando se termina de escribir en la memoria interna 0 y se comienza en la 1 marcado por la señal de `memi_wr_cs_a`. La señal `memi_wren_a` es `data_in_valid` registrada, los datos I y Q se mantienen válidos durante este tiempo.

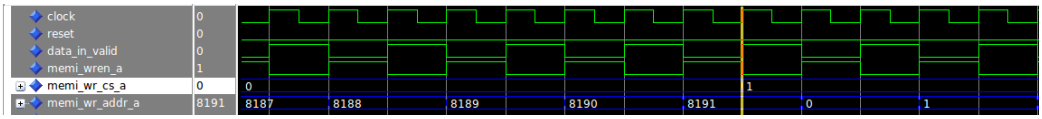


Figura E.9: Simulación del bloque `sym_acq_control` en etapa de escritura en la memoria de entrada.

Lectura memoria de entrada En la Figura E.10 se presenta una simulación de la etapa de lectura de la memoria de entrada. La misma comienza cuando la cantidad de muestras recibidas (`s_useful_samples`) es igual a la cantidad de muestras solicitadas (`n_rcvd_val_in`). En ese instante se da un aviso al bloque de procesamiento (quien solicitó las `n_rcvd_val_in` muestras) mediante la señal `n_rcvd_out`. Este instante está marcado con la línea amarilla.

En consecuencia, el bloque de control comienza a recibir solicitudes de lectura de datos mediante los dos puertos para tal propósito: el 1 y el 2 (`data_1_req_in` y `data_2_req_in`). Los índices solicitados para la primer lectura corresponden a las dos primeras muestras de las 500 solicitadas.

Como el bloque de control en ese instante, y en los próximos períodos de reloj, continúa escribiendo en la primer memoria interna del bloque `mem_ram_2_port_in`

(memi_wr_cs_a = 0), solo se podrá acceder a este *chip* de memoria mediante el puerto B. Finalmente, el bloque de control genera las señales de acceso a lectura correspondientes tan sólo por el puerto B. En este sentido, tres períodos después de la solicitud de lectura, se genera la señal de dato válido tan sólo para el puerto 1 y para el puerto 2 de error (data_1_valid_out y data_2_error_out respectivamente). Como el lector habrá notado el bloque de control prioriza las lecturas por el puerto 1. Los datos entregados por el puerto 1 corresponderán a las direcciones 0, 2, 4 y 6 como se observa en memi_rd_addr_b.

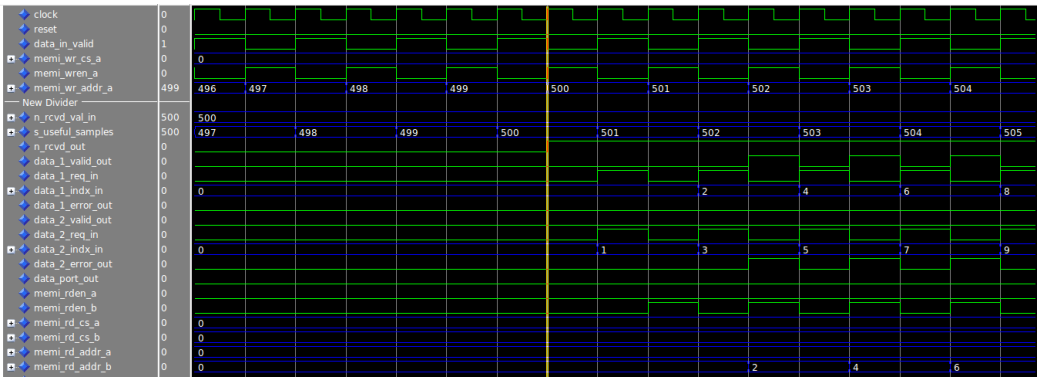


Figura E.10: Simulación del bloque sym_acq_control en etapa de lectura de la memoria de entrada.

Escritura memoria de salida En la Figura E.11 se presenta una simulación para la situación en la que el bloque de control maneja las señales para generar una escritura en la memoria de salida mem_ram_2_port_out. La escritura comienza con la solicitud en la señal req_wr_symb_in como se marca mediante la línea amarilla de la figura. Ya que en ese momento no se están realizando lecturas de la memoria de salida, el bloque de control contesta que es posible realizar la escritura mediante la señal ack_wr_symb_out en 1 (la misma se mantiene con este valor durante todo el período de escritura en memoria).

A continuación, se comienzan a recibir avisos de datos válidos por ambos puertos (data_wr_1_valid_in y data_wr_2_valid_in). Los mismos se mapean a los puertos A y B de la memoria de salida como se observa en las señales memo_wren_a, memo_addr_a y sus análogos del puerto B.

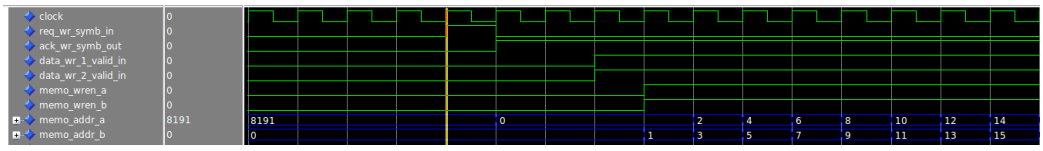


Figura E.11: Simulación del bloque sym_acq_control en etapa de escritura en la memoria de salida.

Pruebas en Hardware Las pruebas en *hardware* de este bloque fueron realizadas en conjunto con el `sym_acq_process` y las memorias. Debido a la naturaleza del mismo la verificación se hizo conjuntamente con estos bloques ya que lo que se verificó fue si los datos estaban correctos. Las pruebas realizadas se presentan en el Capítulo 4.

E.3.5. Diseño del bloque `sym_acq_process`

El bloque `sym_acq_process` se encarga de realizar el procesamiento del `sym_acquisition`. El mismo consiste básicamente en dos funciones: sincronización con los símbolos OFDM de la señal de entrada y corrección fraccional en frecuencia de los mismos. Como recordará el lector, el procesamiento depende de los parámetros: modo, CP y SNR. Para explicar el funcionamiento se considerará que el modo será 3 (modo a utilizar en la aplicación), por lo tanto el largo de símbolos en este caso será $N = 8192$. También se hará el supuesto de que el CP vale 1/4, lo cual corresponde al valor máximo del largo del prefijo: $L = 2048$.

El `sym_acq_process` recibe la señal de entrada mediante una interfaz por la cual puede solicitar determinada cantidad de muestras, las cuales deben estar disponibles en un *buffer*. El mismo estará compuesto por una memoria y un bloque de control que realizará la lectura y escritura de su contenido, de forma transparente para el bloque `sym_acq_process`. Con el objetivo de generalizar la explicación de este bloque se hará referencia a la memoria de entrada ya explicada (`mem_ram_2_port_in`) como *buffer*. Una vez obtenida la habilitación de la cantidad solicitada puede comenzar con el procesamiento o bien continuar con el mismo. De forma análoga, el mismo puede indicar la cantidad de muestras ya procesadas y que por lo tanto se pueden liberar del *buffer*.

La interfaz de salida se utiliza para la escritura de los datos procesados correspondientes a determinado símbolo, la escritura se realiza mediante una habilitación externa.

Tanto la interfaz de entrada como la de salida cuentan con dos puertos de lectura/escritura de datos, lo cual permite realizar operaciones sobre un par de muestras en cada flanco de reloj (con algunas limitaciones en cada caso).

E.3.5.1. Especificación y Requerimientos

A continuación se especifica el funcionamiento del bloque `sym_acq_process` y se definen algunos requisitos del mismo:

Reloj de muestreo y procesamiento

- El reloj de funcionamiento del bloque será el mismo que el de muestreo de la señal (en la aplicación se utilizará un reloj de $2 * f_s$, con $f_s = 512/63$ MHz $\approx 8{,}13$ MHz). El mismo genera una nueva muestra cada dos flancos de reloj.

Parámetros del procesamiento

- Los parámetros: modo, CP y SNR, serán indicados mediante los puertos mode, cp y snr, respectivamente.

Interfaces de entrada/salida

- Las interfaces de entrada y salida del bloque cuentan con dos puertos de lectura y escritura de datos, respectivamente (puerto 1 y 2 en cada caso). En ambos casos, se podrá utilizar ambos puertos en cada flanco de reloj, con las restricciones que se indican más adelante. Ya sea tanto en el caso de lectura como en el de escritura, los puertos 1 y 2 se utilizarán con distinto criterio a explicar en cada caso.

- Cada puerto de los mencionados consiste en un par de entradas (o salidas según corresponda), uno corresponde a la muestra en fase I y otro a la de cuadratura Q. Las muestras serán válidas cuando lo indique la señal de habilitación que dispone cada puerto (en el caso de lectura se indica además si se generó un error en la lectura de la muestra).

- Cada muestra I o Q está compuesta por la cantidad de *bits* indicados por el parámetro DATA_WIDTH el cual se podrá configurar de acuerdo a la señal de entrada (en la implementación se utilizarán 16 *bits*). Las mismas están representadas en complemento a dos.

Lectura de muestras desde el *buffer* de entrada

- El bloque podrá manejar un largo de ventana del *buffer* máximo con cantidad de *bits* indicada por el parámetro ADDR_TOTAL_WIDTH. Para poder realizar la sincronización inicial, el largo debería ser mayor a $2N+L = 18432$ muestras. En la implementación se utilizará una cantidad de ADDR_TOTAL_WIDTH $= 15$ *bits*, lo cual corresponde a un largo de ventana de: $2^{15} = 32768$ muestras, mayor al mínimo necesario.

- En el caso de lectura, el bloque debe solicitar la cantidad de muestras necesarias para su procesamiento. Para esto debe indicar la cantidad solicitada en el puerto de salida n_rcvd_val_out[ADDR_TOTAL_WIDTH-1..0] y mediante el puerto de entrada n_rcvd_in recibe el aviso de confirmación correspondiente a la solicitud.

- La lectura de muestras del *buffer* se realiza mediante un índice y una solicitud de lectura asociada. El índice corresponde al valor del puerto de salida data_1_indx_out y la solicitud se realiza mediante el puerto de salida data_1_req_out (ejemplo para el puerto 1 de entrada). El valor del índice corresponde a la ubicación de la muestra solicitada dentro de la ventana del *buffer*. El mismo se deberá encontrar en el rango perteneciente al intervalo $[0, n-1]$, siendo n la cantidad de muestras solicitadas en el puerto n_rcvd_val_out (cuando la misma haya sido validada).

- Si la muestra solicitada en determinado puerto de lectura está disponible para su lectura en el *buffer*, la misma se recibirá por los puertos `data_rd_1_i` y `data_rd_1_q` cuando la señal de entrada `data_1_valid_in` lo indique. En caso contrario se recibirá un error mediante el puerto `data_1_error_in` (ejemplo para el puerto 1 de entrada).
- El *buffer* de lectura garantiza entregar muestras solicitadas en simultáneo por ambos puertos, si los índices difieren al menos el valor N. En caso contrario, se priorizará la solicitud del puerto 1 y se entregará la muestra asociada a dicho puerto.
- La cantidad de muestras ya procesadas que el *buffer* puede liberar se indica mediante la señal `data_done_out` y la habilitación de ello con el puerto `data_done_valid_out`. De esta forma, se actualizan los punteros `data_1_indx_out` y `data_2_indx_out` hacia las muestras del *buffer*.

Sincronización y entrega de símbolos en el *buffer* de salida

- El bloque se deberá sincronizar con todos los símbolos de la señal de entrada, los mismos deberán ser entregados a la salida. Por lo tanto, el procesamiento deberá ser lo suficientemente rápido para poder entregar todos los símbolos recibidos.
- Los símbolos entregados deberán ser rotados para realizar la corrección fraccional en frecuencia.
- Cada vez que se encuentre un símbolo y se proceda a escribir las muestras a la salida, se deberá dar el aviso mediante el puerto `req_wr_symb_out`. Mediante la señal `ack_wr_symb_in` se indica si el *buffer* de salida está disponible para recibir las muestras del símbolo. Ambas señales se deben mantener activas hasta que se complete la transferencia.
- El *buffer* de salida admite la escritura de datos por los dos puertos disponibles. Las mismos deben estar ordenados de forma creciente, considerando el puerto 2 como superior al 1.
- En caso de estar des-sincronizado con los símbolos de la señal, deberá dar el aviso mediante el puerto de salida `out_of_sync_out`.

E.3.5.2. Interfaz

En la Figura E.12 se observa la interfaz del bloque `sym_acq_process`.

A continuación se describe cada una de las entradas y salidas del bloque, agrupadas según su función:

Control

- `clock`:
  Reloj de entrada del bloque, corresponde al reloj de muestreo de la señal.

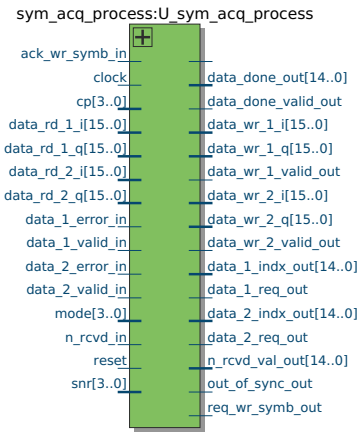


Figura E.12: Interfaz del bloque sym_acq_process.

- reset:
  Reset del bloque, activo por nivel alto.

Parámetros del procesamiento

- mode[3..0]:
  Señal de entrada que indica el modo de transmisión de los símbolos en notación entera sin signo.

- cp[3..0]:
  Señal de entrada que indica el largo del CP. El valor satisface la siguiente ecuación: $CP = \frac{1}{2^{cp}}$, con cp representado en entero sin signo.

- snr[3..0]:
  Entrada que indica el valor de SNR de la señal de entrada (señal OFDM). El valor está indicado en dB representado en entero sin signo.

Solicitud y descarte de muestras en el *buffer* de entrada

- n_rcvd_val_out[ADDR_TOTAL_WIDTH-1..0]:
  Señal de salida que indica la cantidad de muestras necesarias para el procesamiento del bloque, las mismas deben estar disponibles en el *buffer* de entrada para su lectura. El valor está representado en entero sin signo.

- n_rcvd_in:
  Entrada que indica si la cantidad de muestras solicitadas por el puerto n_rcvd_val_out están disponibles.

- data_done_out[ADDR_TOTAL_WIDTH-1..0]:
  Salida que indica la cantidad de muestras que el *buffer* de entrada puede descartar.

- data_done_valid_out:
  Señal de salida que indica si el valor del puerto de data_done_out es válido y por lo tanto se deben descartar las muestras correspondientes.

Puertos de entrada A continuación se explican las señales correspondientes al puerto 1 de lectura (el puerto 2 es análogo).

- data_1_req_out:
  Salida para indicar solicitud de lectura por el puerto.
- data_1_indx_out[ADDR_TOTAL_WIDTH-1..0]:
  Salida con el valor correspondiente al índice de la muestra solicitada por el puerto, la misma está representada en entero sin signo.
- data_rd_1_i[DATA_WIDTH-1..0]:
  Puerto de entrada con el valor I de la muestra solicitada, representado en complemento a dos.
- data_rd_1_q[DATA_WIDTH-1..0]:
  Ídem, con el valor Q.
- data_1_error_in:
  Entrada que indica si se produjo un error en la lectura de la muestra solicitada.
- data_1_valid_in:
  Señal de entrada que indica si las muestras en los puertos data_rd_1_i y data_rd_1_q son válidas.

Puertos para indicar resultado del procesamiento y solicitud de escritura de símbolos

- req_wr_symb_out:
  Salida para indicar solicitud de escritura de muestras del símbolo procesado.
- ack_wr_symb_in:
  Entrada de habilitación de escritura de muestras en el *buffer* de salida.
- out_of_sync_out:
  Señal que indica si el procesamiento está des-sincronizado.

Puertos de salida A continuación se explican las señales correspondientes al puerto 1 de salida (el puerto 2 es análogo).

- data_wr_1_valid_out:
  Salida para indicar escritura por el puerto.
- data_wr_1_i[DATA_WIDTH-1..0]:
  Puerto de salida con el valor I de la muestra a escribir, representada en complemento a dos.

- data_wr_1_q[DATA_WIDTH-1..0]:
  Ídem, con el valor Q.

E.3.5.3. Funcionamiento

A continuación se explicará en detalle el funcionamiento del bloque. Para esto, se dividirá en secciones que abordan características específicas del mismo.

Etapas de procesamiento El procesamiento se realiza en distintas etapas de forma cíclica. Las mismas consisten en las siguientes: **Inicio**, **Cálculo** y **Salida**.

Etapa de procesamiento: Inicio En el **Inicio**, el bloque deberá solicitar la cantidad mínima de muestras necesarias previo a comenzar el cálculo de la correlación. Para comenzar el mismo, el mínimo necesario de muestras es de N, debido a que la correlación se calcula entre muestras separadas por ese valor precisamente (correspondiente a la separación entre el CP y su copia). Luego deberá actualizar la solicitud de acuerdo al requerimiento.

Etapa de procesamiento: Cálculo Durante la etapa de **Cálculo** pueden existir dos posibilidades: fuera de sincronismo o sincronizado. En el caso fuera de sincronismo no se conoce la ubicación del símbolo, por lo tanto, la búsqueda del CP se realizará en toda la ventana de observación ($2N + L$ muestras). De esta forma el rango de valores de correlación a calcular será $N = 8192$. En el caso sincronizado, se estima la ubicación del CP en función del símbolo hallado en el paso anterior, por lo tanto, el rango de búsqueda del CP será reducido. En este caso se calcula la correlación en un rango de 16 valores en torno a la ubicación estimada del CP.

Primero se deberá hallar el primer valor de la función Λ, para lo cual es necesario realizar $L = 2048$ lecturas desde el *buffer*. Por lo tanto será necesario actualizar la solicitud de muestras, ésta se incrementa la misma cantidad de muestras que ingresan al *buffer*, es decir una muestra cada dos flancos de reloj. Luego se deberá iterar tantas veces como la cantidad de valores a hallar de Λ (según el rango de búsqueda definido anteriormente).

Una vez que se obtengan los valores de la función correlación en todo el rango de búsqueda (ya sea el caso sincronizado o no), se deberá decidir si el máximo de los mismos indica la presencia del CP del símbolo. En caso de que el máximo no supere cierto umbral, indicando que no se encontró un símbolo, se realizará una nueva búsqueda pero comenzando más adelante en la señal. Por lo tanto, las muestras que queden excluidas del nuevo rango serán descartadas. En este caso se pasará a la etapa de **Inicio**. En caso de que el máximo indique la presencia del CP del símbolo se pasará a la etapa de **Salida**. Se deberán descartar las muestras previas al símbolo, incluso el CP hallado, de forma de liberar el espacio en el *buffer* y actualizar los punteros correspondientes.

Etapa de procesamiento: Salida En esta etapa se deberán extraer las muestras del símbolo encontrado y realizar la corrección en frecuencia, ambas tareas se

realizarán en simultáneo.

La corrección será realizada por el bloque `sym_acq_derote`. El mismo aguarda por el valor de $\gamma(\theta)$ y calcula su fase para poder realizar la corrección, dando el aviso cuando se encuentre pronto para ello. El `sym_acq_process` dará el aviso de solicitud de escritura, aguardando por la habilitación de la misma.

Una vez obtenida la habilitación de escritura, se comenzará con la lectura del símbolo y la rotación de sus muestras previa a la salida. Esta tarea se realizará mediante el uso de los dos puertos de entrada y salida, de forma de minimizar el menor tiempo de procesamiento. En caso de haber algún error con la lectura en los puertos de entrada, se deberá realizar nuevamente la lectura de la muestra asociada al error para garantizar la integridad del símbolo completo.

Una vez finalizada la escritura, el bloque indica el descarte de las muestras correspondientes al símbolo en el *buffer* de la entrada y aguarda por la llegada de la cantidad de muestras mínima para realizar un nuevo cálculo, pasando la etapa de **Inicio**.

Luego, el funcionamiento del bloque continúa de la misma forma descripta anteriormente a través de las distintas etapas.

Definición de una máquina de estados para el control de las etapas

Para poder manejar las distintas etapas del funcionamiento de forma ordenada se definió una máquina de estados.

En la Figura E.13 se presenta el diagrama de estados y las condiciones de transición.

A continuación se explican algunos detalles sobre los estados:

- `st_idle`:
  Estado luego de un *reset*. Permite inicializar las señales de control y distintos contadores auxiliares.

- `st_wait_N_2`:
  Corresponde a la etapa de **Inicio**. Durante el mismo se aguarda por la cantidad mínima de muestras en el *buffer* de entrada para comenzar el procesamiento ($N = 8192$).

- `st_lambda_0`:
  Corresponde a la primer parte de la etapa de **Cálculo**. En este caso se realiza la primer parte del algoritmo de la correlación, correspondiente a la sumatoria de L sumandos. Se acumula una suma parcial previa a llegar al primer valor de Λ. La lectura se realiza una vez por cada nueva muestra, es decir una vez cada dos flancos de reloj.

- `st_lambda_L`:
  Es un estado de transición para hallar el primer valor de Λ y preparar el próximo estado.

- `st_lambda_N_L`:
  Corresponde a la parte iterativa de la etapa de **Cálculo**. Aquí se realiza el

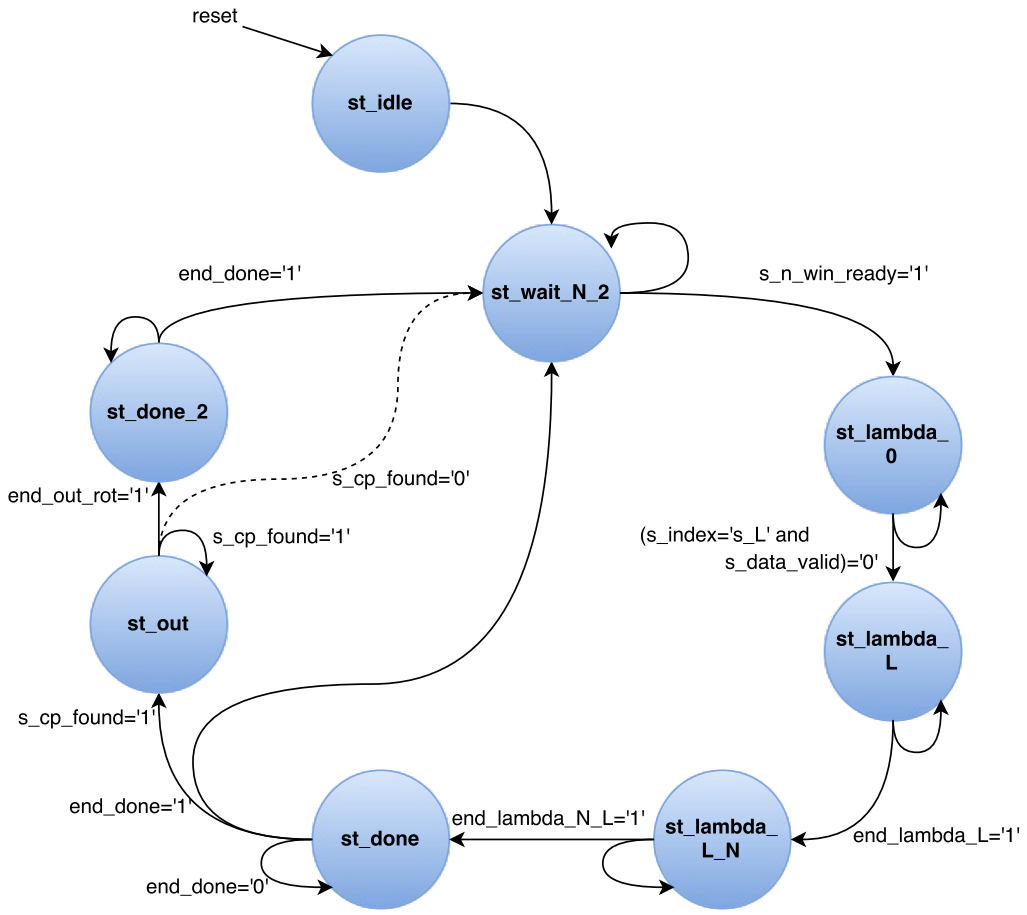


Figura E.13: Diagrama de estados para las etapas de procesamiento del sym_acq_process.

cálculo de Λ en todo el rango de búsqueda (según sea sincronizado o no, $N = 8192$ o 16 valores, respectivamente). Durante el mismo, se realiza el cálculo recursivo, lo cual implica la lectura de muestras nuevas y anteriores.

- st_done:
  En el mismo se determina si el CP fue encontrado, en función del resultado anterior. En caso de que se haya encontrado, se define que el procesamiento está sincronizado, en caso contrario se define que el mismo está fuera de sincronismo. Por lo tanto, se determina si se pasa a la etapa de **Salida** o la de **Inicio**. Se descartan las muestras ya procesadas, en caso de sincronismo se descartan las muestras previas al símbolo (incluso el CP hallado) en caso contrario se descartan las muestras correspondientes a un símbolo y su CP $(N + L)$.

- st_out:
  Estado para etapa de **Salida**. Se realiza la lectura y corrección en frecuencia del símbolo, el cual se entrega a la salida.

- st_done_2:
  Se descartan todas las muestras del último símbolo, salvo las últimas 8 a incluir en el próximo cálculo de la correlación. Se preparan algunas señales para la nueva búsqueda en sincronismo. Se pasa a la etapa de **Inicio**.

La máquina es un referencia determinante de la etapa de procesamiento del bloque. Según la misma, algunos procesos del bloque se activarán y tendrán determinado comportamiento. Existen algunas señales auxiliares que cumplen la misma función, las más importantes serán presentadas en la explicación de los procesos del bloque (ver más adelante).

Señales que determinan las transiciones entre estados:

s_n_win_ready:
Señal que indica si la cantidad de muestras mínima necesaria para comenzar el procesamiento fue habilitada.

s_index:
Contador para la cantidad de valores calculados de la función Λ.

s_L:
Valor del parámetro L.

s_data_valid:
Señal auxiliar para indicar si las muestras solicitadas en la entrada fueron válidas.

end_lambda_L:
Señal para la salida del estado st_lambda_L. Estará activa una vez que se calcule el primer valor de Λ.

end_lambda_N_L:
Indica la salida del estado st_lambda_N_L. Se activa cuando termina el cálculo de Λ en el rango de búsqueda.

end_done: Cuando esté activa se realizará el cambio de estado, esto sucede una vez que se pasen 8 períodos de reloj. La misma señal se utiliza en los estados st_done y st_done_2.

s_cp_found: La misma indica si se encontró la ubicación del símbolo en la última búsqueda, por lo tanto indica si el procesamiento se encuentra sincronizado o no.

Implementación de las funciones mediante procesos en paralelo

A continuación se explica el funcionamiento de los procesos del bloque. Los mismos funcionan en paralelo, algunos de ellos dependen de los estados tal como se indica en cada caso.

Proceso n_rcvd_proc: Proceso para generar el valor de los parámetros N, CP, L y SNR. Los mismos serán representados mediante las señales s_N, s_cp, s_L y s_snr respectivamente.

Proceso sta_mach_proc: Máquina de estados, el estado actual se representa con la señal STATE a la cual se le asigna el valor de NXSTATE en cada flanco de reloj.

Proceso nx_sta_mach_proc: En el proceso se asigna el valor del próximo estado según el actual y las condiciones correspondientes (ver Figura E.13).

Proceso proc: En el mismo se asignan los valores de las señales s_n_win, s_req_en, s_index y odd (las cuales se explican a continuación) según el estado y algunas condiciones.

La señal s_n_win se asigna con el valor de muestras solicitado en cada caso. Durante st_wait_N_2 toma el valor de $N + 2$. En los estados st_lambda_0 y st_lambda_N_L se incrementa una unidad cada dos flancos de reloj. De esta forma al terminar st_lambda_N_L vale $2N + L + 2 = 18434$. En el estado st_out vale $N = 8192$.

Mediante s_req_en se habilita la solicitud de muestras por los puertos de entrada. El contador s_index indica la cantidad de valores calculados de la función Λ. La bandera odd permite alternar entre dos casos de forma cíclica, para realizar operaciones en flancos alternos.

Proceso n_rcvd_proc: Proceso para la solicitud de muestras en el *buffer* de entrada. En cada flanco copia el valor de la señal s_n_win al puerto de salida n_rcvd_val_out, indicando la cantidad de muestras solicitada. También copia el valor de n_rcvd_in en la señal interna s_n_win_ready.

Proceso req_data_proc: Este proceso se encarga de la solicitud de lectura desde la entrada. El mismo funciona durante los estados st_lambda_0, st_lambda_N y st_out. La lectura se podrá realizar según el valor de s_req_en.

En el caso st_lambda_0 solicita muestras separadas N índices por cada puerto, por L veces (sin lecturas repetidas). Durante st_lambda_L_N realiza s_range+L lecturas dobles. Una para las muestras nuevas y otra para eliminar el término recursivo (mediante L muestras hacia atrás), cada una igual al caso anterior. El valor de s_range indica la cantidad de valores a calcular de la correlación, este valor se define de acuerdo a la búsqueda en sincronismo o no. El mismo puede valer 16 o N, correspondiente al sincronismo o no respectivamente.

Durante el estado st_out realiza la lectura de las muestras del símbolo encontrado (N lecturas, al menos). La misma se realiza en orden debido que la salida debe estar ordenada. En este caso se deberán realizar re-lecturas en caso de errores. Se definió que la lectura se realice de dos formas posibles, una continua y otra repetitiva para recuperación ante errores (mediante la señal req_cont activa e inactiva respectivamente). En el caso continuo se leen muestras consecutivas de forma ordenada, mientras no se produzcan errores en la lectura de las mismas. En el caso repetitivo se solicita la misma muestra por ambos puertos hasta que se pueda leer por ambos puertos sin errores. Estos casos están determinados por el proceso out_data_proc que recibe las muestras leídas. En el caso repetitivo, el proceso debe llevar la cuenta de la cantidad de muestras solicitadas de esta forma. Además debe indicar mediante la señal rd_cont si la lectura se realizó de forma continua o repetitiva, la cual es análoga a req_cont para el caso de lectura.

La solicitud de muestras termina si se completó la lectura de todo el símbolo.

Proceso corr_proc: En este proceso se realiza el cálculo de la correlación correspondiente a la función Λ. La cual se compone de γ y Φ, que se calculan

mediante sumatorias, de forma recursiva. Para esto se asignan variables con el valor de los elementos que componen las sumatorias en función de las muestras leídas. Los mismos se asignan a registros auxiliares que almacenan los valores de cada elemento y los acumulan en registros que corresponden a los valores de las sumatorias.

Durante el cálculo recursivo se debe tener especial cuidado en la acumulación mediante las distintas señales auxiliares, en particular en los casos de borde. Por este motivo, fue creado el estado de transición `st_lambda_L`. De esta forma se hace posible separar los casos de borde de los casos iterativos permitiendo realizar distintas formas de procesamiento. En general, las asignaciones se realizan según casos alternos, los cuales permiten calcular nuevos y viejos sumandos. Estos sumandos permiten calcular nuevos valores de las funciones recursivas en función del valor anterior.

En el proceso se determina el valor de θ mediante el valor del máximo de Λ, además registra el valor de $\gamma(\theta)$. Al final de la iteración determina si el valor máximo de Λ supera un umbral, indicando mediante la señal `s_cp_found` si el CP fue hallado.

A continuación se explican algunas señales y variables auxiliares y en particular las correspondientes a la función de correlación.

La muestra leída en el puerto 1 será $r[k] = r_I[k] + j * r_Q[k]$, de índice k relativo a la ventana de observación. La del puerto 2 será $r[k+N]$.

Variables:

`v_abs_r0`:

Variable que contiene el sumando:

$$|r[k]|^2 = |r_I[k]|^2 + |r_Q[k]|^2$$

`v_abs_rN`:

Ídem anterior para la muestra desfasada N índices:

$$|r[k+N]|^2 = |r_I[k+N]|^2 + |r_Q[k+N]|^2$$

`v_r0_rN_i`:

Variable con el sumando:

$$real\,(r[k]r^*[k+N]) = (r_I[k]r_I[k+N] + r_Q[k]r_Q[k+N])$$

`v_r0_rN_q`:

Ídem anterior para la parte imaginaria:

$$imag\,(r[k]r^*[k+N]) = (r_Q[k]r_I[k+N] - r_I[k]r_Q[k+N])$$

Las mismas se representan con el doble de *bits* de cada muestra I o Q.

`v_ab2_g`:

Variable auxiliar para el cálculo de γ^2. La misma utiliza el doble de *bits* que la función γ.

`s_ab2_g`:

Señal con el valor de γ^2. También utiliza el doble de *bits* que la función γ.

Señales correspondientes a funciones:

Las señales correspondientes a funciones serán representadas con el doble de *bits* de cada muestra I o Q. Por lo tanto, utilizarán $2 * \texttt{DATA\_WIDTH} = 32$ *bits*.

s_lambda:
Función Λ evaluada en determinado índice. La misma tiene asociada la señal lambda_valid que indica si su contenido es válido.

s_gamma_i:
Parte real de la función γ evaluada en determinado índice.

s_gamma_q:
Ídem parte imaginaria.

s_f_phi_i:
Función proporcional a la función Φ en determinado índice. La misma corresponde al valor $2 * \Phi$.

Señales correspondientes a sumandos auxiliares:

f_g_pp_i:
Registro auxiliar con el valor de v_r0_rN_i para agregar en la suma que genera el valor de γ.

f_g_pp_q:
Ídem para la parte imaginaria.

f_f_pp_i:
Señal con el valor de v_abs_r0+ v_abs_rN para acumular en el cálculo de Φ.

El uso de las señales anteriores será descrito en detalle en más adelante en la sección de Algoritmos.

Proceso done_data_proc: El proceso permite realizar acciones posteriores al procesamiento de la señal. Una acción corresponde al descarte de muestras y la otra a la asignación de señales de control según el resultado del procesamiento.

El descarte sucede durante los estados st_done y st_done_2. En ambos casos se asigna la cantidad de muestras a descartar según el resultado del procesamiento. Esto se realiza asignando a la señal s_n_don el valor correspondiente y activando la salida data_done_valid_out en un flanco de reloj. Durante el estado st_done se le asigna $\theta + L$ en caso de sincronismo (todas las muestras previas al símbolo) y $N + L$ en caso contrario (el tamaño de un símbolo y su prefijo). En el caso de st_done_2, se asigna el valor $N - 8$, correspondiente a un símbolo completo salvo algunas muestras (8) a incluir en la búsqueda siguiente.

En el estado st_done se asigna la señal de salida out_of_sync_out indicando si el bloque se encuentra fuera de sincronismo. En caso de sincronismo se activa la salida req_wr_symb_out durante los estados st_done y st_out solicitando la escritura de las muestras del símbolo hallado.

Proceso out_data_proc: En este proceso se verifica la lectura correcta de las muestras del símbolo desde el *buffer* de entrada. El proceso funciona únicamente durante el estado st_out. En caso de errores de lectura, el proceso debe realizar una pausa e indicar cual muestra se debe solicitar nuevamente en la entrada. Cada

muestra del símbolo que se haya leído correctamente se copia al bloque interno sym_acq_derote para que realice la corrección en frecuencia.

En condiciones normales (lectura sin errores por ambos puertos) el proceso copia cada muestra de los puertos de entrada hacia las señales data_rot_1_i, data_rot_1_q, cuando lo indique la señal data_rot_1_valid, estas señales ingresan al bloque sym_acq_derote para que realice la corrección de las mismas (ejemplo para el puerto 1, el puerto 2 es análogo).

La detección de errores se realiza teniendo en cuenta el criterio de lectura que realiza el proceso req_data_proc durante el estado st_out (ver más arriba). Dicho proceso puede realizar una lectura continua o repetitiva. Mediante la señal req_cont se indica si la lectura por parte del proceso req_data_proc se debe realizar de forma continua o repetitiva.

En caso de existir error en alguna de las muestras leídas, el proceso pausa la copia durante 10 períodos de reloj e indica que la lectura se debe realizar de forma repetitiva hasta que no se produzcan errores de lectura. De esta forma, se permite un tiempo de espera hasta que la lectura se vuelva a sincronizar (desde la muestra asociada al error). Esto es necesario debido al retardo existente entre la solicitud de lectura en el *buffer* de entrada y la respuesta correspondiente, además del tiempo necesario para reconfigurar la solicitud.

Bloque sym_acq_derote

La función de corrección en frecuencia es realizada mediante el bloque sym_acq_derote. La corrección se realiza en forma serial, a diferencia de los procesos del sym_acq_process, los cuales trabajan en paralelo. El bloque podría haberse ubicado fuera del sym_acq_process, se realizó dentro del mismo para unificar las funciones de procesamiento dentro de un solo bloque. Otra causa corresponde a que se utilizan dos puertos para realizar la corrección y por lo tanto la misma se puede realizar la mitad del tiempo que se realizaría con un solo puerto. En consecuencia, resultó sencilla la comunicación entre el sym_acq_derote y el resto de los procesos.

Para realizar la corrección se debe hallar el valor de $\angle\gamma(\theta)$, por lo tanto se debe tener a disposición el valor de $\gamma(\theta)$. El valor de $\gamma(\theta)$ es provisto al sym_acq_derote para que calcule su fase. Esto se realiza mediante las entradas gam_in_i y gam_in_q que indican la parte real e imaginaria de $\gamma(\theta)$, respectivamente, el valor correspondiente será válido cuando lo indique la entrada gam_in_valid.

Una vez que se tenga el valor de $\angle\gamma(\theta)$, el bloque configura su funcionamiento para realizar la rotación de las muestras. Cuando esté pronto, da el aviso mediante la salida derote_ready_out.

Luego realiza la rotación de las muestras que ingresen por los puertos de entrada. El bloque asume que todas la muestras de la entrada del puerto 1 son válidas y que las del puerto 2 son válidas solo si están en conjunto con las del puerto 1. También asume que la muestra del puerto 2 es la siguiente a la del puerto 1. Las muestras válidas, según el criterio anterior, serán rotadas y copiadas a la salida.

Interfaz del bloque sym_acq_derote

En la Figura E.14 se observa la interfaz del bloque sym_acq_derote.

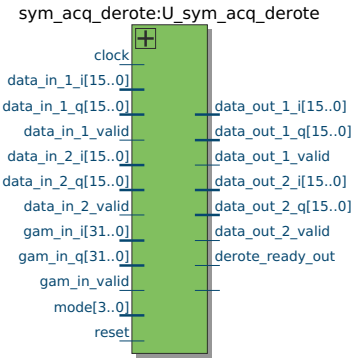


Figura E.14: Interfaz del bloque sym_acq_derote.

A continuación se describe cada una de las entradas y salidas del bloque, agrupadas según su función:

Control

- clock:
  Reloj de entrada del bloque, corresponde al reloj de muestreo de la señal.

- reset:
  Reset del bloque, activo por nivel alto.

Parámetros

- mode[3..0]:
  Señal que indica el modo de transmisión al igual que en el sym_acq_process, por lo que determina el valor del largo del símbolo N.

- gam_in_i[2*DATA_WIDTH-1..0]:
  Señal con el valor de la parte real $\gamma(\theta)$.

- gam_in_q[2*DATA_WIDTH-1..0]:
  Ídem para la parte imaginaria.

- gam_in_valid:
  Señal que indica si el contenido de las anteriores es válido.

- derote_ready_out:
  Señal de salida para indicar si el bloque está pronto para realizar la corrección en frecuencia.

Puertos de entrada Los puertos de entrada son similares a los del `sym_acq_process`, la diferencia es que las muestras son entregadas sin ser solicitadas. A continuación se muestran las señales para el puerto 1 (el puerto 2 es análogo).

- `data_in_1_i[DATA_WIDTH-1..0]`:
  Puerto de entrada con el valor I de la muestra solicitada, representado en complemento a dos.
- `data_in_1_q[DATA_WIDTH-1..0]`:
  Ídem, con el valor Q.
- `data_in_1_valid`:
  Señal de entrada que indica si las muestras en los puertos `data_in_1_i` y `data_in_1_q` son válidas.

Puertos de salida Los puertos de salida son iguales a los del `sym_acq_process`. A continuación se muestran las señales para el puerto 1 (el puerto 2 es análogo).

- `data_in_1_i[DATA_WIDTH-1..0]`:
  Puerto de entrada con el valor I de la muestra solicitada, representado en complemento a dos.
- `data_in_1_q[DATA_WIDTH-1..0]`:
  Ídem, con el valor Q.
- `data_in_1_valid`:
  Señal de entrada que indica si las muestras en los puertos `data_in_1_i` y `data_in_1_q` son válidas.

Funcionamiento del bloque `sym_acq_derote` El `sym_acq_derote` realiza la corrección en frecuencia de cada símbolo. La corrección se realiza mediante la rotación de fase creciente de las muestras del símbolo. El único parámetro necesario para la corrección es el valor $\gamma(\theta)$ (además del largo del símbolo). Luego el bloque halla el valor de su fase para determinar la rotación a aplicar sobre cada muestra. El bloque instancia dos bloques `cordic`, los cuales se encargan de realizar la rotación de las muestras de cada puerto.

El `sym_acq_derote` calcula el valor de $\angle\gamma(\theta)$ mediante uno de los bloques `cordic`, debido a que los mismos pueden realizar también el cálculo de fase de determinada muestra.

Una vez que obtiene el valor de $\angle\gamma(\theta)$, prepara los bloques `cordic` para que realicen la rotación y da el aviso por la salida `derote_ready_out` que está pronto para recibir las muestras del símbolo.

Para realizar las funciones anteriores se creó una máquina de estados dentro un único proceso. En la Figura E.15 se presenta el diagrama de estados y las condiciones de transición.

A continuación se explican los detalles sobre el procesamiento durante los estados:

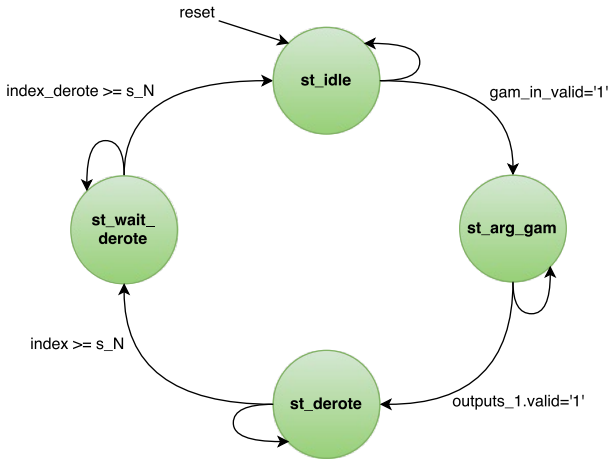


Figura E.15: Diagrama de estados del sym_acq_derote.

- st_idle:
  Estado luego de un *reset*. Aguarda en el mismo hasta que la señal de entrada gam_in_valid esté activa indicando que está disponible el valor de $\gamma(\theta)$. En dicho momento, configura el bloque cordic_1 para que realice el cálculo de fase, asignando a la señal cordic_mode el valor CORDIC_VECTORING, pasando al estado st_arg_gam. También copia en las entradas del cordic_1 los siguientes valores:

  - inputs_1.x: Parte imaginaria de $\gamma(\theta)$.
  - inputs_1.y: Parte real de $\gamma(\theta)$.
  - inputs_1.z: Valor 0, el mismo es necesario para obtener la fase de la entrada.

- st_arg_gam:
  Estado para el cálculo de $\angle\gamma(\theta)$. Aguarda en el mismo hasta que se obtiene la salida del bloque cordic_1 (debido a la latencia del calculo), indicada por la señal outputs_1.valid. Cuando esto sucede, copia el valor de $\angle\gamma(\theta)$ indicado por la señal outputs_1.z en la señal arg_gam. Otras consecuencias son el cambio al estado st_derote y el aviso de que se encuentra pronto para la rotación de las muestras, indicado mediante la señal de salida derote_ready_out. Finalmente configura el modo de los bloques cordic para que realicen la rotación, para esto se asigna a la señal cordic_mode el valor CORDIC_ROTATION.

- st_derote:
  Durante este estado se realiza la rotación de las muestras de la entrada. Aguarda en el mismo hasta que el total de muestras que ingresan por los puertos de entrada sea igual a N y por lo tanto se complete la lectura del símbolo. Mientras permanece en el estado realiza dos funciones, una es la

de lectura de muestras de la entrada y otra de escritura de muestras en la salida.

Para la lectura copia las muestras de cada puerto en las entradas de los bloques `cordic`. Mediante la variable `index` lleva la cuenta de las muestras leídas. Para cada muestra asigna al `cordic` el valor correspondiente a la rotación:

$$-i/N * \angle\gamma(\theta)$$

Siendo i el valor de la variable `index`.

La escritura a la salida del `sym_acq_derote` se realiza copiando las salidas de los bloques `cordic`, mediante la variable `index_derote` lleva la cuenta de la cantidad de muestras.

Una vez terminada la lectura cambia al estado `st_wait_derote` para terminar la rotación de las muestras.

- `st_wait_derote`:
  Durante este estado solo realiza la función de escritura, de la misma forma que en el estado `st_derote`, hasta completar el símbolo (en ese momento: `index_derote`$= N$). Una vez finalizada la escritura cambia al estado `st_idle` para aguardar por un nuevo símbolo.

Algoritmos

Búsqueda del prefijo cíclico mediante correlación La búsqueda del CP se realiza mediante la función correlación, Λ:

$$\Lambda(\theta) = |\gamma(\theta)| - \rho\Phi(\theta)$$

con:

$$\gamma(m) = \sum_{k=m}^{m+L-1} r[k]r^*[k+N],$$

$$\Phi(m) = \sum_{k=m}^{m+L-1} \frac{|r[k]|^2 + |r[k+N]|^2}{2},$$

y $\rho = \sigma_s^2/(\sigma_s^2 + \sigma_n^2) = SNR/(1+SNR)$.

Por lo tanto, la ubicación del CP corresponde al valor de $\theta = \widehat{\theta}_{EMV} = arg_\theta max\{\Lambda\}$.

Para el cálculo de las funciones γ y Φ se considerarán las siguientes identidades:

$$\gamma(m) = \sum_{k=m}^{m+L-1} r[k]r^*[k+N] = \sum_{k=m}^{m+L-1} g_k$$

$$\Rightarrow \gamma(m) = g_m + g_{m+1} + ... + g_{m+L-1}$$

$$\Rightarrow \gamma(m) = -g_{m-1} + g_{m-1} + g_m + g_{m+1} + ... + g_{m+L-1}$$

$$\Rightarrow \gamma(m) = -g_{m-1} + \sum_{k=m-1}^{m-1+L-1} g_k + g_{m+L-1}$$

$$\Rightarrow \gamma(m) = -g_{m-1} + \gamma(m-1) + g_{m+L-1}$$

De forma análoga para Φ:

$$\Phi(m) = \sum_{k=m}^{m+L-1} \frac{|r[k]|^2 + |r[k+N]|^2}{2}$$

$$\Rightarrow 2\Phi(m) = \sum_{k=m}^{m+L-1} |r[k]|^2 + |r[k+N]|^2 = \sum_{k=m}^{m+L-1} f_k$$

$$\Rightarrow 2\Phi(m) = f_m + f_{m+1} + ... + f_{m+L-1}$$

$$\Rightarrow 2\Phi(m) = -f_{m-1} + f_{m-1} + f_m + f_{m+1} + ... + f_{m+L-1}$$

$$\Rightarrow 2\Phi(m) = -f_{m-1} + \sum_{k=m-1}^{m-1+L-1} f_k + f_{m+L-1}$$

$$\Rightarrow 2\Phi(m) = -f_{m-1} + 2\Phi(m-1) + f_{m+L-1}$$

Por lo tanto, es posible calcular las funciones $\gamma(m)$ y $\Phi(m)$ en función del resultado anterior: $\gamma(m-1)$ y $\Phi(m-1)$, agregando un sumando (g_{m+L-1} y f_{m+L-1}) y restando otro (g_{m-1} y f_{m-1}), en cada caso. Para hallar el primer valor de γ y Φ, $\gamma(0)$ y $\Phi(0)$, se puede usar la misma lógica:

$$\gamma(0) = \sum_{k=0}^{L-1} r[k]r^*[k+N] = \sum_{k=0}^{L-1} g_k$$

$$\Rightarrow \gamma(0) = g_0 + g_1 + ... + g_{L-1}$$

Si se define $\gamma(-L) = g_0$

$$\Rightarrow \gamma(n-L) = \gamma(n-L-1) + g_{n-1}$$

con $n \in [1, L]$

$$\Rightarrow \gamma(0) = \gamma(-1) + g_{L-1}$$

De forma análoga para $2\Phi(0)$:

$$\Phi(0) = \sum_{k=0}^{L-1} \frac{|r[k]|^2 + |r[k+N]|^2}{2}$$

$$\Rightarrow 2\Phi(0) = f_0 + f_1 + ... + f_{L-1}$$

Si se define $2\Phi(-L) = f_0$

$$\Rightarrow 2\Phi(n-L) = 2\Phi(n-L-1) + f_{n-1}$$

con $n \in [1, L]$

$$\Rightarrow 2\Phi(0) = 2\Phi(-1) + f_{L-1}$$

Por lo tanto el primer valor de γ y 2Φ se puede hallar de igual manera inicializando los registros correspondientes y acumulando los sumandos g_{n-L} y f_{n-L} (con $n \in [1, L]$) sobre los mismos. Por lo tanto el cálculo de $\gamma(0)$ y $2\Phi(0)$ implica L iteraciones. Es decir es necesario realizar L lecturas de las muestras. Luego, una vez obtenido el primer valor $\gamma(0)$ y $2\Phi(0)$, cada nuevo valor necesita realizar dos lecturas desde la entrada. Por ejemplo, para el siguiente valor $\gamma(1)$ y $2\Phi(1)$, es necesario sumar g_L y f_L y restar g_0 y f_0, respectivamente. Por lo tanto, cada nuevo valor implica realizar dos lecturas desde la entrada. Debido a que cada muestra nueva llega cada dos flancos de reloj y la lectura de muestras se puede realizar en cada flanco de reloj, el cálculo de las funciones γ y Φ se puede realizar a la misma cadencia con la que llegan las muestras.

Estas funciones se calculan en simultáneo debido a que en todos los casos utilizan las mismas entradas. Las mismas utilizan el doble de *bits* de las muestras (16) debido a que en el cálculo de los elementos g y f se duplica la cantidad de *bits* (32).

Los distintos valores g y f se registran en las siguientes señales:

`f_g_mm_i`: g_{m-1_I} `f_g_mm_q`: g_{m-1_Q}
`f_g_pp_i`: g_{m+L-1_I} `f_g_pp_q`: g_{m+L-1_Q}
`f_f_mm_i`: f_{m-1_I} `f_f_mm_q`: f_{m-1_Q}
`f_f_pp_i`: f_{m+L-1_I} `f_f_pp_q`: f_{m+L-1_Q}

Se considerará que el valor de ρ será constante e igual a 1, lo cual corresponde a un valor de SNR de 0 dB. Esta limitante en el algoritmo se debió a que si ρ es un valor arbitrario puede generar errores en el cálculo de Λ.

La función $\gamma(m)$ será representada por un par de registros: `s_gamma_i`, `s_gamma_q` correspondientes a la parte real e imaginaria. Para $2\Phi(m)$ se utilizará el registro `s_f_phi_i` ya que la función es real pura. Debido a que la función Λ, necesita los valores de $|\gamma|$ y Φ en simultáneo, es necesario registrar el valor de `s_f_phi_i` hasta que se calcule $|\gamma|$.

Para hallar el valor de $|\gamma|$ se calcula primero el valor de $|\gamma|^2$ mediante la siguiente relación:

$$|\gamma|^2 = {\gamma_I}^2 + {\gamma_Q}^2$$

El valor de $|\gamma|^2$ se registra en la señal `s_ab2_g`, la cual utiliza el doble de *bits* de γ. Luego, para hallar $|\gamma|$ se utiliza la siguiente identidad:

$$|\gamma| = \sqrt{|\gamma|^2}$$

Para esto se utilizará el bloque `altsqrt` de Altera que realiza la raíz cuadrada, el resultado se asigna a la señal `s_abs_g` representado nuevamente con la cantidad de *bits* de γ, es decir, el doble de las muestras de la entrada (32).

Finalmente el valor de Λ se calculará de la siguiente forma:

$$\Lambda(\theta) \cong |\gamma(\theta)| - \frac{2\Phi(\theta)}{2}$$

El cual se registra en la señal s_lambda, la misma se representa con el doble de *bits* de las muestras de la entrada (32). Los valores de Λ se calculan a la cadencia de la llegada de muestras (cada dos flancos de reloj), debido a que las funciones γ y Φ se calculan a dicha frecuencia (como se explicaba más arriba). Cada nuevo valor de Λ hallado, se comparará con el máximo Λ_{Max} que se haya determinado anteriormente, en caso de ser mayor se actualizará el valor de Λ_{Max} en la señal s_lambda_max.

Rotación de fases mediante el algoritmo CORDIC Como se adelantaba anteriormente, la rotación de fase de las muestras del símbolo se realiza mediante el algoritmo CORDIC [28].

El mismo permite realizar las rotaciones de forma iterativa en pocos pasos de cálculo y mediante un consumo reducido de recursos. Esto se debe a que el mismo utiliza únicamente sumas, restas, *shifts* y búsqueda en una tabla reducida. La tabla está compuesta de valores de ángulos con tangentes notables, los cuales corresponden a potencias de 1/2. El algoritmo alcanza el resultado de la rotación mediante aproximaciones sucesivas. Particularmente se utiliza para el caso de rotaciones de puntos en coordenadas cartesianas. Si la rotación a realizar es de ángulo α, la misma se puede descomponer en los siguientes ángulos:

$$\alpha \approx \sum_{m=0}^{M-1} \alpha_m$$

Por lo tanto la rotación se puede descomponer en M rotaciones elementales de ángulos α_m. Los ángulos α_m que se eligen para realizar las rotaciones elementales corresponden a ángulos con tangentes notables, así al momento de realizar las rotaciones las operaciones resultan triviales. Los valores que se eligen de α_m tienen los siguientes valores de tangentes:

$$tan(\alpha_m) = \pm\frac{1}{2^m}$$

Por lo tanto, si la muestra a rotar es $A_0 = (x_0, y_0)$ la rotación se pude escribir de la siguiente forma:

$$R_\alpha(x_0, y_0) = A_0 e^{\alpha} \approx A_0 e^{\sum_{m=0}^{M-1} \alpha_m} = A_0 \prod_{m=0}^{M-1} e^{\alpha_m}$$

$$R_\alpha(x_0, y_0) \approx R_{\alpha_m}(R_{\alpha_{m-1}}(...R_{\alpha_0}(x_0, y_0)...))$$

Lo cual se puede escribir de forma recursiva de la siguiente forma:

$$A_{m+1} = (x_{m+1}, y_{m+1}) = R_{\alpha_m}(x_m, y_m) = A_m e^{\alpha_m}$$

Utilizando notación matricial y la matriz de rotación asociada:

$$\Rightarrow \begin{bmatrix} x_{m+1} \\ y_{m+1} \end{bmatrix} = \begin{bmatrix} cos(\alpha_m) & -sin(\alpha_m) \\ sin(\alpha_m) & cos(\alpha_m) \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} x_{m+1} \\ y_{m+1} \end{bmatrix} = cos(\alpha_m) \begin{bmatrix} 1 & -tan(\alpha_m) \\ tan(\alpha_m) & 1 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix}$$

Por lo tanto si los valores de $tan(\alpha_m)$ satisfacen $tan(\alpha_m) = \pm\frac{1}{2^m}$, cada rotación elemental resulta trivial, ya que alcanza con aplicar un *shift* en los *bits* correspondientes a la división por un factor potencia de 2. El factor $cos(\alpha_m)$ es una constante que se puede considerar fuera de los pasos de la iteración. Si $K = \prod_{m=0}^{M-1} cos(\alpha_m)$, por lo tanto la rotación se puede escribir como:

$$\begin{bmatrix} x_M \\ y_M \end{bmatrix} = K \begin{bmatrix} x'_M \\ y'_M \end{bmatrix}$$

con K una constante que solo depende de M:

$$K = \prod_{m=0}^{M-1} cos(\alpha_m) = \prod_{m=0}^{M-1} \frac{1}{\sqrt{1 + \frac{1}{4^m}}}$$

Los valores de (α_m) de determinan en función del resultado de la aproximación al ángulo total α. Si δ_m es el signo de cada (α_m), por lo tanto el error en el paso M de la iteración se puede escribir de la siguiente forma:

$$e_M = \alpha - \sum_{m=0}^{M_1} \delta_m \alpha_m$$

Y el mismo se puede escribir en forma recursiva:

$$e_{m+1} = e_m - \delta_m \alpha_m$$

Por lo tanto realizando el siguiente algoritmo iterativo se puede aproximar al resultado de la rotación:

$$\begin{cases} \delta_m = sgn(e_m) \\ x'_{m+1} = x'_m - \delta_m \frac{y'_m}{2^m} \\ y'_{m+1} = y'_m - \delta_m \frac{y'_m}{2^m} \\ e_{m+1} = e_m - \delta_m \alpha_m \end{cases} \tag{E.1}$$

De forma dual a la rotación, el algoritmo se puede utilizar para realizar rotaciones hasta llegar desde el punto (x_0, y_0) al punto de coordenadas $(|A_0|, 0)$ sobre el eje real. De esta forma se puede hallar el valor de $|A_0|$ y el ángulo del punto inicial. Para esto alcanza con cambiar algunas condiciones en las ecuaciones E.1. Este forma se conoce con el nombre de modo vectorial. El algoritmo se puede generalizar para otro tipo de funciones utilizando el mismo criterio de aproximaciones sucesivas.

Bloque `cordic` El bloque `cordic` se encuentra disponible en la librería de Nuand. El mismo implementa el algoritmo CORDIC para el modo de rotación y vectorial, para lo cual realiza M iteraciones hasta alcanzar el resultado, dicho valor se puede cambiar de forma paramétrica. El resultado de cada rotación se encuentra disponible luego de M flancos de reloj, ya que el mismo realiza la recursión mediante una cadena serial (*pipeline*). De esta forma permite realizar rotaciones independientes en cada flanco de reloj. La cantidad de *bits* con la que se representan las muestras se puede cambiar también de forma paramétrica. En particular, el valor de fases de las muestras está representado con ese tamaño y por lo tanto determina la precisión de las rotaciones a realizar. Por otra parte, el mismo también determina la precisión del resultado debido a que representa las componentes del mismo (parte real e imaginaria). En la aplicación se determinó que se utilizarían $M = 30$ pasos del algoritmo y una representación de 32 *bits*. Esto se debió a que para representar los valores de los ángulos con la precisión requerida era necesario incrementar la cantidad de *bits*. Luego de haber aumentado la cantidad de *bits*, debido a que la salida del `sym_acq_process` utiliza 16 *bits* fue necesario truncar la salida del bloque `cordic`. Esto último genera ruido sobre la señal.

En la Figura E.16 se observa la interfaz del bloque `cordic`.

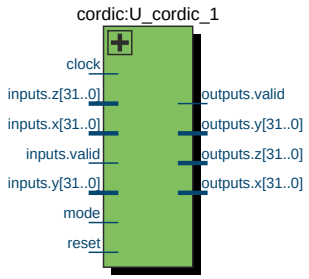


Figura E.16: Interfaz del bloque `cordic`.

A continuación se describe cada una de las entradas y salidas del bloque, agrupadas según su función:

Control

- `clock`:
  Reloj de entrada del bloque, corresponde al reloj de muestreo de la señal.

- `reset`:
  Reset del bloque, activo por nivel alto.

Parámetros

- `NUM_STAGES`:
  Constante que indica el número de pasos del algoritmo, corresponde al valor M (ver más arriba).

- `mode`:
  Señal de entrada para indicar el modo de funcionamiento. Cuando toma el valor `CORDIC_ROTATION` funciona en el modo de rotación de las muestras de la entrada, si toma el valor `CORDIC_VECTORING` realiza el modo vectorial de las mismas.

Entradas

- `inputs`:
  Entrada del tipo `record` que consta de tres señales:
  - `inputs.x[31..0]`: Parte real de la muestra de la entrada representada en complemento a dos.
  - `inputs.y[31..0]`: Ídem parte imaginaria.
  - `inputs.z[31..0]`: Valor de fase a rotar, representado en complemento a dos. El valor corresponde a un ángulo de $\alpha = \frac{z}{2^M}\pi$.
  - `inputs.valid`: Señal que indica si las demás (`x`,`y` y `z`) son válidas.

Salidas

- `outputs`:
  Salida del mismo tipo que `inputs`. En el modo de rotación se obtiene en la señal `outputs.x` el valor real del vector rotado, en la señal `outputs.y` el valor imaginario y en la señal `outputs.z` el error angular de la aproximación. En el modo vectorial se obtiene en la señal `outputs.x` el módulo del vector de la entrada, en la señal `outputs.y` el error de la aproximación y en la señal `outputs.z` el ángulo del vector de la entrada.

Análisis de Tiempos de procesamiento Como se mencionaba en la sección de Algoritmos, el cálculo de la correlación Λ se realiza en dos etapas. En la primer etapa se calcula el primer valor $\Lambda(0)$, la cual implica L iteraciones. En la segunda etapa se calculan los demás valores de forma recursiva realizando dos operaciones por cada nuevo valor. La primera se puede realizar en $2L$ períodos de reloj. La segunda implica 2 períodos de reloj por cada nuevo valor. En el caso de que el bloque esté fuera de sincronismo, se debe realizar la búsqueda del CP en todo el rango de observación $2N + L$, por lo tanto se deben hallar N valores de la correlación.

En el siguiente análisis se supondrá que el tiempo de espera debido las configuración de las distintas etapas será nulo, de forma de facilitar el mismo. Los tiempos implicados son los siguientes:

- Solicitud y respuesta de muestras en el *buffer*.
- Configuración de lectura de muestras y latencia de respuesta del *buffer* de entrada.

- Transiciones de estado.
- Recuperación de errores.
- Configuración de la rotación.
- Latencia del cálculo de la rotación de las muestras.
- Solicitud de escritura en el *buffer* de salida.

Aún considerando el total de los tiempos anteriores, se puede suponer que entre todos no a superan $N/2$ flancos de reloj correspondientes a una cuarta parte de la duración de un símbolo.

Si n representa la cantidad de muestras del *buffer* de entrada. Supongamos que el *buffer* de entrada se encuentra vacío al comienzo de la búsqueda ($n = 0$, por ejemplo luego de un *reset*). Por lo tanto, el tiempo necesario para el cálculo se puede resumir de la siguiente forma:

1. Etapa **Inicio**: Se deben esperar por $n = N + 2 = 8194$ muestras en el *buffer* de entrada. Por lo tanto, implica $2(N + 2) = 16388$ períodos de reloj.
2. Etapa **Cálculo**, $\Lambda(0)$: Se realizan L lecturas para hallar el valor de $\Lambda(0)$, las mismas se realizan en $2L = 4096$ períodos de reloj. Esto se debe a que son necesarias $n = N + L$ muestras en el *buffer*, es decir L muestras adicionales. Por lo tanto, el tiempo acumulado es $(2(N + 2)) + (2L) = 20484$ flancos de reloj.
3. Etapa **Cálculo**, $\Lambda(m) : m \in [1, N - 1]$: Se realizan $N - 1$ cálculos de la función Λ, lo cual implica $2(N-1)$ períodos de reloj. Esto se debe a que son necesarias $n = N+L+N$ muestras nuevas en el *buffer*, es decir N muestras adicionales. Por lo tanto, el tiempo acumulado es $(2(N+2))+(2L)+(2N) = 2(2N+2+L) = 36868$ flancos de reloj, es decir levemente mayor al tiempo de arribo de toda la ventana de observación (ya que la misma vale $2(2N + L)$).

Por esta causa el mínimo necesario para el largo del *buffer* es precisamente $2N+L$.

Luego, suponiendo que se encontró el símbolo, el mismo debe ser leído desde el *buffer*. Considerando el peor caso de ubicación del símbolo, correspondiente a: $\theta = 0$. El *buffer* podrá liberar únicamente L muestras correspondientes al CP. Por lo tanto, la lectura del símbolo se tiene que realizar antes de que el largo total del *buffer* se llene. Como se verá en los siguientes párrafos este límite no será alcanzado.

La lectura del *buffer* de la entrada se realiza utilizando los dos puertos disponibles. Por lo tanto, dependiendo que existan errores de lectura por alguno de los puertos el tiempo total de la lectura podría cambiar. Si T_r representa el tiempo de la lectura del símbolo, el mismo se encontrará en el rango $T_r \in [T_S/4, T_S/2]$, siendo T_S el tiempo de arribo de un símbolo ($2N$ períodos de reloj). Por lo tanto en el peor de los casos el tiempo de lectura de un símbolo es de $T_S/2$. En caso de la búsqueda fuera de sincronismo, el tiempo total de búsqueda y salida del símbolo es de $= 2(2N+2+L)+N = 45060$ períodos de reloj, correspondiente al tiempo de

arribo de la ventana de observación más el de medio símbolo. Por lo tanto el mínimo necesario de memoria para esta restricción es $= (2N + 2 + L) + N/2 = 22530$ ya que cada dos períodos de reloj se recibe una muestra.

En caso de búsqueda en sincronismo el análisis es similar, a menos que el cálculo de Λ se realiza en 16 valores. Por lo tanto, el tiempo total para la búsqueda y salida del símbolo en sincronismo es $2L + 2 * 15 + N$ períodos de reloj, correspondiente a $2L$ para el cálculo de $\Lambda(0)$, $2 * 15$ correspondientes al cálculo de los restantes valores de Λ y N para la lectura del símbolo. Aún considerando los tiempos de espera de configuración y latencia de las distintas etapas, el tiempo total es menor al arribo de cada nuevo símbolo. Por lo tanto, el tiempo total de procesamiento en sincronismo es menor a la llegada de símbolos y entonces se garantiza la salida de todos los símbolos recibidos. Se recuerda que el tamaño del *buffer* de entrada es de $4N = 32768$ muestras mayor al mínimo calculado (22530).

E.3.5.4. Validación

Simulaciones Para la validación del bloque `sym_acq_process` se realizaron simulaciones mediante *test-benchs*, tanto para el bloque completo como para los sub-bloques internos `sym_acq_derote` y `cordic`. A continuación se presentan la simulación del bloque completo y en particular la corrección en frecuencia realizada por el bloque `cordic`.

Simulación del bloque `sym_acq_process` En la Figura E.17 se observa la simulación del `sym_acq_process`. La simulación contiene la búsqueda de dos símbolos luego de un *reset*. El primero se realiza la búsqueda en el intervalo de largo $2N+L$, en la cual se encuentra el CP y luego una segunda búsqueda en sincronismo donde también se encuentra el símbolo.

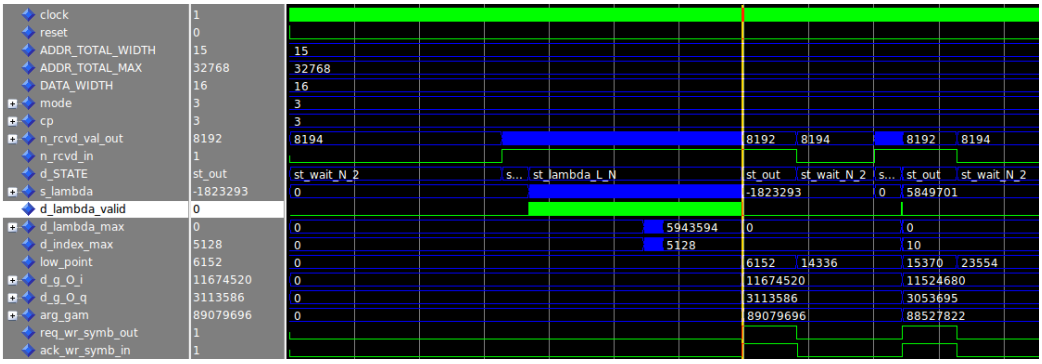


Figura E.17: Simulación del bloque `sym_acq_process`.

En la misma se observan algunas señales de interés, a continuación se explica el comportamiento de ellas:

- `clock`: Reloj de muestreo, debido a la cantidad de flancos se observa como una franja continua.

- reset: *Reset* al comienzo de la simulación.
- ADDR_TOTAL_WIDTH,ADDR_TOTAL_MAX: Indican el largo máximo del *buffer* de la entrada (cantidad de *bits* (15) y cantidad de muestras (32768) respectivamente).
- DATA_WIDTH: Cantidad de *bits* de las muestras (16).
- mode,cp: Parámetros de la señal, modo de transmisión 3 y $CP = 1/8$, respectivamente. Por lo tanto, $N = 8192$ y $L = 1024$.
- n_rcvd_val_out, n_rcvd_in: La primera solicita las muestras de acuerdo a las distintas etapas, las cuales se pueden identificar mediante la señal d_STATE que indica los estados, mediante n_rcvd_in se observa la habilitación correspondiente.
- d_STATE: Señal que indica el estado actual, la misma realiza la siguiente secuencia para ambos símbolos:

  Inicio:st_wait_N_2.

  Cálculo:st_lambda_0→st_lambda_L→st_lambda_L_N.

  Salida:st_done→st_out→st_done_2.
- s_lambda, d_lambda_valid: Señal con el valor de Λ cuando d_lambda_valid lo indica. Se observa que la mayoría de los valores se calculan durante el estado st_lambda_L_N.
- d_lambda_max, d_index_max: Señal con el valor de $\Lambda_{Max} = \Lambda(\theta)$, el índice correspondiente se encuentra en la señal d_index_max. En el ejemplo $\Lambda(\theta) = 5943594$ y $\theta = 5128$.
- low_point: Señal auxiliar que indica donde apunta la primer muestra en el *buffer*. La misma se actualiza cada vez que se descartan muestras. En el ejemplo, durante el primer estado st_out, vale $\theta + L = 6152$, luego al finalizar el estado vale $(\theta + L + N - 8) = 14336$.
- d_g_0_i, d_g_0_q: Señales con la parte real e imaginaria de $\gamma(\theta)$.
- arg_gam: Señal con el valor de $\angle\gamma(\theta)$ normalizado a la representación que utiliza el bloque cordic. En el ejemplo: $\angle\gamma(\theta) \approx 1{,}21rad$.

  Por lo tanto:

  $$\text{arg\_gam} \approx \frac{\angle\gamma(\theta)}{\pi} 2^M$$

  Con $M = 30$ correspondiente al número de pasos del algoritmo CORDIC.
- req_wr_symb_out, ack_wr_symb_in: Señales para la solicitud y habilitación de escritura. Las mismas permanecen activas durante el estado st_out.

De la simulación se observa que entre el primer símbolo y el segundo, el tiempo de espera en el estado `st_out` es del orden de $N + L$ y de esta forma se muestra que el procesamiento del bloque en sincronismo es suficientemente rápido. Por lo tanto, se concluye que el procesamiento del bloque a lo largo de símbolos verifica los requerimientos de tiempo.

La función correlación que realiza el bloque `sym_acq_process` fue verificada mediante la comparación con *scripts* de Octave que emulan el mismo comportamiento numérico. En la Figura E.18 se observa la función Λ para la señal de la simulación presentada en la Figura E.17. Se puede observar los valores de $\Lambda(\theta) = 5943594$ y $\theta = 5128$ hallados por el bloque `sym_acq_process`, correspondientes al máximo de la función correlación y el índice asociado. Los valores corresponden a las señales `d_lambda_max` y `d_index_max` respectivamente de la Figura E.17.

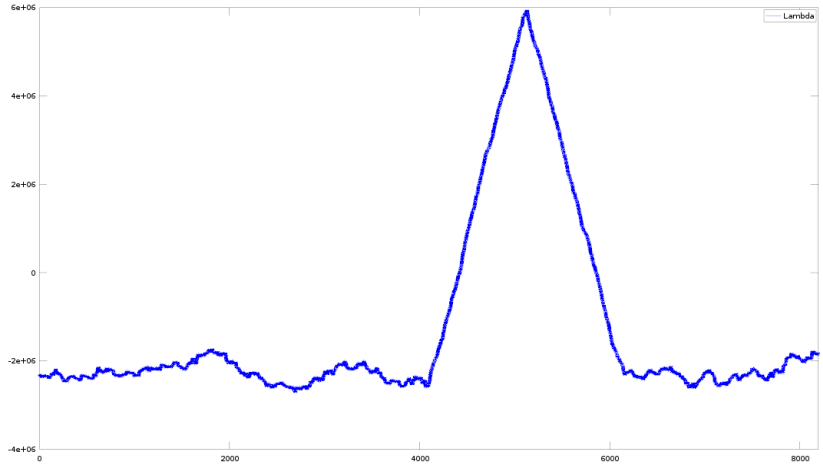


Figura E.18: Función Λ correspondiente a una ventana de observación de $2N + L$.

Corrección en frecuencia mediante el bloque `cordic` Para la simulación del bloque `cordic`, se utilizó una señal de OFDM capturada del aire conteniendo 10000 símbolos. En particular se revisó la rotación que realiza el bloque para un símbolo, a modo de ejemplo. En la Figura E.19 se observa la diferencia de fase entre la rotación mediante el bloque `cordic` y la rotación que realiza Octave (la cual es tomada como referencia) para un símbolo.

En la misma se observa que la diferencia de fase tiene una tendencia lineal, correspondiente a la rotación para la corrección fraccional en frecuencia. Como es evidente la rotación mediante el bloque, tiene una gran cantidad de ruido, debido al truncamiento de la salida como se explicaba más arriba. Pese a esto, la salida del bloque se utilizó como entrada de un *flowgraph* de GNU Radio con parte del receptor `gr-isdbt`, obteniendo finalmente la señal de video de un par de segundos.

Pruebas en Hardware Las pruebas en hardware del bloque fueron realizadas junto al bloque `sym_acquisition`. Las mismas se presentan en el Capítulo 4.

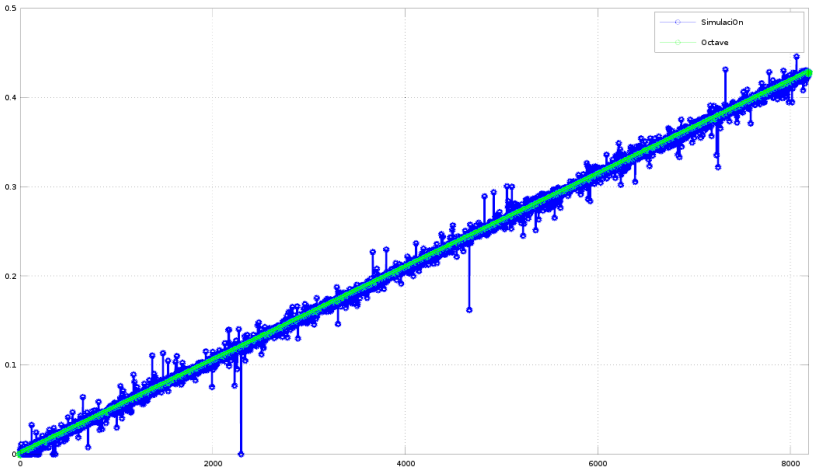


Figura E.19: Simulación del bloque `cordic`. En la Figura se observa la diferencia de fase entre un símbolo rotado con el bloque `cordic` comparado contra el símbolo sin la corrección.

E.3.6. Diseño del bloque `mem_ram_2_port_out`

El bloque `mem_ram_2_port_out` es el bloque de memoria de salida del `sym_acquisition`. El principal requerimiento del mismo es que tenga el tamaño (cantidad de muestras que puede almacenar) del símbolo procesado. Como este está determinado por el modo, y en Uruguay se utiliza el modo 3 que equivale a $N = 8192$ se definió de este tamaño. Otro requerimiento de este bloque es que pueda escribirse lo más rápido posible, en este sentido se optó por una memoria ram de dos puertos igual a la utilizada para la `mem_ram_2_port_in`. Por esta razón, en esta sección se presenta únicamente la interfaz en la Figura E.20.

E.4. Conclusiones

Definición de la Arquitectura Para cada bloque se presentó el diseño, la especificación, el detalle de su funcionamiento y la forma en que fue validado.

Filtro Pasa Bajos: `sdr_en_fpga_filter` El bloque `sdr_en_fpga_filter` implementa un filtro FIR pasa bajos. Este fue necesario debido al ancho de banda utilizado (8,13 MHz, ya que el mismo se encuentra ligado a la frecuencia de muestreo), mayor al mínimo necesario (6 MHz correspondientes a un canal de DTV) , de forma de atenuar la señal en los canales adyacentes. Para el diseño del mismo se definieron los requerimientos de ganancia y *ripple* de la banda pasante y atenuante. De esta forma se obtuvieron los coeficientes y se utilizó un bloque de Nuand que implementa el filtro FIR de los coeficientes determinados. El bloque se probó en el entorno de GNU Radio Companion verificando que el mismo permite filtrar la señal de entrada.

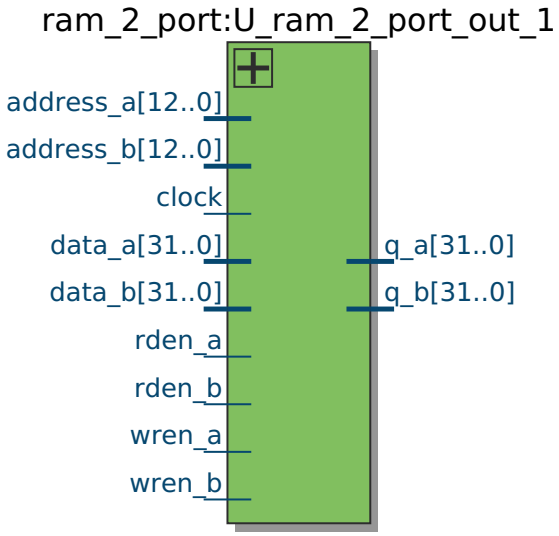


Figura E.20: Interfaz del bloque `mem_ram_2_port_out`.

Bloque `interface` El bloque `interface` permite la lectura de los símbolos de salida del bloque `sym_acquisition` y la escritura en el bloque `fifo_writer`. El mismo se validó mediante simulaciones y pruebas en el FPGA.

En el marco de las pruebas del mismo se encontraron problemas respecto al envío de muestras hacia GNU Radio. Por un lado, se encontró un límite de pausas en la generación de muestras en el FPGA determinado por el `Osmocom Source`. Por otra parte, se encontraron problemas de pérdidas de muestras en el camino hacia la PC. Se verificó con la herramienta *Signal Tap II* mediante la conexión del USB Blaster con el FPGA, que las muestras se generaban en el FGPA sin pérdidas, haciendo el chequeo de los *timestamps*. Al momento de recibir las mismas en la PC se registraron muestras perdidas y saltos en los mismos. En base a esta prueba se determinó que el problema se encontraba en el *buffer* de GNU Radio, correspondiente al bloque `Osmocom Source`. Luego se realizaron capturas incrementando el tamaño de los *buffers* y se verificó que las muestras generadas en el FGPA se recibían íntegramente.

Bloque de procesamiento `sym_acquisition` El bloque de procesamiento `sym_acquisition` corresponde a la versión en el FPGA del bloque de GNU Radio `OFDM Sym Acquisition` del módulo de `gr-isdbt`. El diseño del mismo implicó la especificación de una arquitectura interna al bloque basada en funcionalidades. La misma consiste en bloques de memoria de entrada y salida, en un bloque de control y en uno de procesamiento.

El `sym_acquisition` se encarga de llevar a cabo la sincronización temporal con los símbolos en la entrada y de la corrección fraccional en frecuencia de los mismos. Para la primer tarea, utiliza un algoritmo recursivo correspondiente a la función de correlación. Este algoritmo implicó la definición del mínimo necesario de memoria para la recepción de muestras en la entrada y de los requerimientos de

lecturas sobre la misma. El mismo permite la sincronización con todos los símbolos recibidos. Para llevar a cabo la segunda función se implementó un algoritmo de rotación de las muestras de cada símbolo. La corrección se realizó aplicando una rotación a las muestras de cada símbolo mediante una fase creciente, la cual se realiza mediante el algoritmo iterativo CORDIC.

Bloques de Memoria Teniendo en cuenta las especificaciones y los requerimientos, se definieron los tipos de memoria a utilizar. Estos implicaron un consumo de una gran cantidad de los recursos de memoria disponibles del FPGA, en particular, de bloques M9K.

Se implementó una memoria de entrada `mem_ram_2_port_in` de largo 32768 para almacenar los datos de entrada al bloque `sym_acquisition`. La misma se estructuró en base a un arreglo de cuatro memorias ram de dos puertos, de Altera. El acceso a la misma por dos puertos, facilitó ampliamente su manejo. Para ello fue necesario implementar una serie de bloques auxiliares de tipo decodificadores y multiplexores. El consumo de esta memoria fue de 128 bloques de M9K.

La memoria de salida `mem_ram_2_port_out` se definió del tamaño de un símbolo debido a que el procesamiento entrega esto a su salida. La memoria de salida es del mismo tipo que las componentes de la memoria de entrada. Para la implementación de este bloque se consumieron 32 bloques de M9K.

En resumen, se utilizaron 160 bloques de memoria M9K de los 432 disponibles en el FPGA (37 %).

Bloque de Control Se implementó un bloque de control `sym_acq_control` para el manejo de las memorias para el procesamiento y de las comunicaciones con los bloques aguas abajo y aguas arriba del `sym_acquisition`. Se definieron protocolos de comunicación entre los bloques internos y para las interfaces de entrada y salida para llevar a cabo esas funciones.

Se diseñó una arquitectura para el `sym_acq_control` de procesos en paralelo lo que permitió la ejecución de varias funciones en su mayoría independientes, a la vez. Las mismas son: escritura en memoria de entrada de muestras que recibe el `sym_acquisition`, lectura de las mismas para su procesamiento. Cálculo de muestras útiles. Escritura del símbolo procesado en la memoria de salida y lectura de la misma por parte del bloque `interface`.

Se simuló el bloque para diversas situaciones y se verificó su correcto funcionamiento. La validación concluyó con una prueba integral en *hardware* explicada en el capítulo siguiente.

Bloque de Procesamiento `sym_acq_process` Finalmente, se implementó el `sym_acq_process` que lleva a cabo el procesamiento del bloque `sym_acquisition`. El mismo realiza las funciones correspondientes a la sincronización temporal con los símbolo de la señal de entrada y a la corrección fraccional en frecuencia.

Se definió un algoritmo basado en la implementación del `OFDM Sym Acquisition` del módulo `gr-isdbt` de GNU Radio. Para la implementación del mismo se tuvo en cuenta las limitantes intrínsecas de la plataforma de desarrollo: recursos y

representación numérica limitados. Esto tuvo como consecuencia que se definiera un algoritmo lo más eficiente posible para utilizar la menor cantidad de recursos (de memoria sobre todo) y para que el resultado numérico fuera lo más exacto posible.

Para el caso de la sincronización temporal se implementó el algoritmo de máxima verosimilitud mediante el cálculo de la función de correlación Λ. El máximo valor de dicha función indica la presencia del CP de cada símbolo. Es válido si supera determinado umbral. El cálculo de la misma fue realizado mediante un algoritmo recursivo de cada componente γ y Φ. De esta forma, se minimiza el consumo de recursos y también permite garantizar que el error acumulado de la recursión sea nulo. Por lo tanto, el error de cálculo de Λ será mínimo. El mismo implica el menor tiempo de cálculo posible y permite hallar cada valor de Λ a igual cadencia de la frecuencia de muestreo.

En una primera búsqueda, el algoritmo, requiere de una ventana de $2N + L = 18432$, muestras lo cual condiciona el tamaño mínimo de la memoria de entrada al bloque `sym_acquisition`. Luego, la lectura de muestras correspondientes al símbolo se realiza utilizando el menor tiempo posible. Para esto se realiza una lectura doble de las muestras de la memoria de entrada a través de los dos puertos para tal propósito, en caso de haber errores en la lectura, se realizan re-lecturas para garantizar la integridad de cada símbolo. La búsqueda en sincronismo se realiza sobre un conjunto reducido de valores (16) permitiendo que el cálculo en este caso sea aún más rápido y por lo tanto garantizando la sincronización en régimen dado el tamaño de la memoria de entrada y la cadencia con la que llegan las muestras a la misma.

La corrección fraccional en frecuencia se realiza aplicando una rotación de fase de crecimiento lineal con incremento asociado a la fase del valor de $\gamma(\theta)$.

Mediante el algoritmo iterativo CORDIC se lleva a cabo la misma previo a la salida de las muestras del símbolo. Para esto se utilizan dos bloques `cordic` disponibles en la librería de Nuand, uno para cada puerto de datos. Uno de esos bloques se utiliza para hallar la fase del resultado intermedio $\gamma(\theta)$. Luego, se utiliza dicho valor para realizar la rotación de cada muestra del símbolo. Esto se lleva a cabo para cada componente (I y Q) de la misma mediante cada bloque `cordic`.

El algoritmo permite hallar el resultado de la rotación con buena precisión si se utiliza una gran cantidad de *bits* de representación y de pasos de cálculo. En la implementación se utilizaron $M = 30$ pasos y 32 *bits* de representación. Debido a que la salida del mismo se debe pasar nuevamente a la cantidad de 16 *bits* de representación, que utiliza el resto de la cadena de procesamiento hacia GNU Radio, el bloque genera un gran ruido de fase debido al truncamiento de los datos. Pese a esto último, se verificó que los símbolos corregidos con este algoritmo podían ser decodificados por el resto de la cadena de procesamiento exitosamente.

Apéndice F

Plan de Administración del Proyecto

F.1. Resumen

A continuación se indican los puntos que resumen el contenido del Proyecto:

1. **Cliente:** Pablo Belzarena, Federico Larroca (Grupo ARTES)
2. **Fecha prevista de finalización:** 30 de Abril de 2016
3. **Total de hs. a realizar previstas por el grupo de proyecto:** Se prevé una dedicación semanal de 10 horas por cada integrante del proyecto.
4. **Fecha y descripción de los entregables intermedios:** Hito 1: Fecha estimada: 15 de Setiembre de 2015. Entregables:
   - Arquitectura del sistema: Especificación bloques de la arquitectura del FPGA y cómo se incorporan a la arquitectura que tiene la placa Blade RF.
   - Especificación del Bloque de Correlación.

   Hito 2: Fecha estimada: 15 de Febrero de 2016. Entregables:
   - Especificación Bloques FFT y Ajustes
   - Especificación de la prueba del sistema completo.

F.2. Descripción del Proyecto

Introducción: Actualmente Pablo Belzarena y Federico Larroca integrantes del grupo de Análisis de Redes, Tráfico y Estadísticas de Servicios (ARTES) del Instituto de Ingeniería Eléctrica (IIE) está trabajando con diversos sistemas Software Defined Radio (SDR).

Los mismos han sido diseñados utilizando principalmente dos plataformas de hardware (HW) diferentes: Ethus 100 y Blade RF. Estos sistemas consisten básicamente en recibir ciertas señales RF utilizando el HW para tal propósito en dichas

placas y procesarlas según la aplicación correspondiente. La mayoría de los bloques definidos para sus sistemas los desarrollan en C++ y corren en una PC. En este sentido, se busca llevar a HW determinados bloques mediante su implementación en el lenguaje de descripción de HW VHDL. La finalidad será experimentar diversas posibilidades de mejoramiento de dichos sistemas. Se estudiará si esta solución permitirá abaratar costo de procesamiento y mejorar eficiencia. Para esto se ha realizado una primera experimentación con la placa bladeRF debido a su mayor capacidad.

Antecedentes: **GNU Radio:**

La plataforma de software GNU Radio provee de diversas y extensas herramientas para el desarrollo de sistemas SDR. GNU Radio provee una gran variedad de bibliotecas de código abierto de distintos bloques que se pueden incorporar al diseño. También permite el desarrollo y prueba de nuevos bloques y del sistema completo. Los bloques se definen y desarrollan en lenguaje C++ o Python. El lenguaje C++ se utiliza para la implementación de funciones de cierto requerimiento crítico para el procesamiento de las señales, Python se utiliza principalmente para la definición e interconexión de bloques. Una ventaja de utilizar esta herramienta es que pude ser implementada en software en un entorno de simulación o a nivel de HW.

HW:

La placa bladeRF es una plataforma SDR diseñada para experimentar con comunicaciones RF. Cuenta con una interfaz USB 3.0, TX y RX de RF. Además está compuesta por un microcontrolador 200MHz AMR9 con 512kB de SRAM y un chip Altera Cyclone IV EFPGA el cual puede ser 40KLE o 115KLE. El FPGA cuenta con elementos de lógica dedicada a DSP y permite la interfaz entre el Firmware FX3 y la interfaz de radio. Entre sus principales características encontramos su alta eficiencia, su arquitectura de bajo ruido de alimentación y portabilidad. El rango de frecuencias de la interfaz de radio es 300MHz-3.8GHz. Por lo tanto esta placa cuenta con características deseables para implementar un sistema SDR inclusive pudiendo llevar a HW algunos bloques o el sistema completo de radio.

DTV ISDB-T:

Es una implementación del estándar ISDB-T de Televisión Digital conocido mundialmente como ISDB-Tb desarrollado en GNU Radio. La modulación que se utiliza en el estándar es OFDM.

El demodulador del sistema DTV ISDB-T está compuesto por los bloques de la Figura F.1.

F.3. Objetivo General

El objetivo del Proyecto es la familiarización y estudio de las ventajas y desventajas de la implementación de los tres primeros bloques del sistema SDR denominado DTV ISDB-T en el FPGA de la Placa bladeRF. En particular, se estudiará cuánto procesamiento se le quita al CPU mediante este proceso, así como también cómo resulta el profiling del sistema.

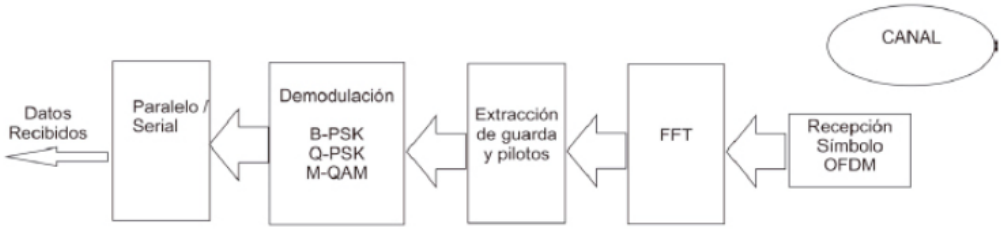


Figura F.1: Bloques demodulador OFMD (http://www.scielo.org.mx/).

F.4. Alcance

El alcance del proyecto comprende la implementación y prueba de integración en HW de los tres primeros bloques del sistema DTV ISDB-T en el FPGA de la placa bladeRF con el chip x115 (Ver Figura 1). Los bloques a pasar a HW son los siguientes:

1. **Bloque Auto-Correlación** Dada la llegada de flujo de símbolos proveniente del sistema RF Front End debe encontrar la ventana de largo 8000, es decir inicio y por ende enumerar todos los siguientes. Para encontrar la ventana se utiliza la Auto-Correlación como estimador de máxima verosimilitud. Por lo tanto alcanza con considerar el valor de máximo de la Auto-Correlación correspondiente a el alineamiento entre el prefijo y su copia asociados a inicio y fin respectivamente. También se realiza un ajuste 'fino' de frecuencia mediante cuentas a definir. Al final del bloque se obtiene identificada la ventana correspondiente a cada trama OFDM.

2. **Bloque FFT** El bloque realiza la FFT de la Trama OFDM devuelta por el bloque anterior obteniendo de esta manera los datos enviados.

3. **Bloque Ajuste** Aquí se realizan los siguientes ajustes: Ajuste fino de Frecuencia. Ajuste del canal. Para esto se utiliza información sobre la codificación utilizada en OFDM. Existen Pilotos predefinidos en los mapas de OFDM algunos son conocidos y otros espúreos. Por lo tanto se deben interpretar los Pilotos conocidos y los espúreos donde éstos últimos es necesario encontrarlos. La atenuación del canal se ecualiza aquí.

F.5. Criterios de éxito

Los criterios de éxito son los siguientes:

- Implementar y testear en el FPGA de la placa bladeRF X115 los bloques Auto-Correlación, FFT y Ajuste. Se ensayarán en HW los bloques por separado y en su conjunto, haciendo transparente al usuario si están implementados en el FPGA o en SW como es actualmente.

- Analizar y comparar con el sistema realizado 100 % en SW. Se busca evaluar ventajas y desventajas de diseñar determinados bloques en HW, así como también comparar dicho sistema en términos de rendimiento y costos de procesamiento con el implementado completamente en SW.
- Se deberá entregar documentación que incluya experiencias del desarrollo del Proyecto y manuales para las distintas herramientas así como también un manual que contenga los pasos necesarios tanto para testear como para utilizar el sistema implementado en HW.

F.6. Actores

Los actores del proyecto son los siguientes: Integrantes del proyecto: Daniel Contrera, Florencia Ferrer Tutores del proyecto: Julio Pérez, Leonardo Etcheverry, Pablo Belzarena Cliente: Grupo ARTES Es importante destacar que Pablo Belzarena integra este grupo y jugará tanto el rol de cliente como de tutor.

F.7. Supuestos

Se asume que el Proyecto se desarrolla con las siguientes condiciones:

1. Entrega del HW (1 placa bladeRF X115 - por parte del grupo ARTES) y SW (Simulink para simulaciones del sistema en el FPGA - por parte de tutores) necesarios para la realización del Proyecto. En este punto se aclara que GNU Radio y Quartus son dos programas que se utilizarán pero son de uso libre y gratuito.
2. Entrega del HW necesario y señales de entrada al sistema para realizaciones de pruebas intermedias (bloques por separado) y del sistema completo. En este punto del proyecto aún no está definido cómo se probarán los bloques por lo tanto no es posible enumerar los elementos necesarios para llevarlo acabo (se piensa que serán necesarias dos placas Blade RF una para generar y otra para recibir).
3. Se asume que se dispondrá de una señal modulada en OFDM según la norma ISDB-T para el ambiente de prueba.
4. Entrega de la documentación necesaria para la comprensión del sistema SDR DTV ISDB-T ya implementado en SW.
5. El sistema DTV ISDB-T está bien definido (documentación de bloques, entradas, salidas, tipos de datos, funcionalidad).
6. Es posible implementar los tres primeros bloques en el FPGA de la placa bladeRF.
7. Se definirán entornos de prueba para cada bloque por separado y para el conjunto.

8. El grupo dispone de al menos 10 horas semanales individualmente promedio de dedicación respecto al proyecto a lo largo de su extensión.

9. Los tutores del proyecto estarán disponibles para la realización de reuniones de avance del proyecto al menos una vez cada dos semanas y así mismo tendrán disponibilidad por mail para consultas puntuales.

10. Se cuenta con la financiación necesaria para adquisición de recursos materiales o software necesario para el desarrollo del proyecto. En este sentido, en principio el único elemento que se debería adquirir son dos placas Blade RF. Esto será provisto por el grupo ARTES por lo que en este punto del proyecto todos los elementos estarían cubiertos. De surgir nuevos se deberán buscar fuentes de financiación.

11. El cliente dispondrá del tiempo necesario para la coordinación de la realización de pruebas.

12. Los tutores estarán disponibles para evaluar y aceptar los entregables así como también la documentación final del proyecto.

F.8. Restricciones

A continuación se enumeran algunas condiciones que restringen el desarrollo del Proyecto:

1. Las placas a utilizar se están importando y aún no han llegado, se estima lo harán para fines de Abril. De llegar en otro momento o en el caso de que fallen será una gran limitante para el desarrollo del proyecto. En caso de darse este escenario se prevé la utilización de una placa de Ethus B100 ya disponible en el IIE.

2. Julio Pérez estará ausente los primeros tres meses del proyecto.

F.9. Especificación funcional del Proyecto

El sistema SDR desarrollado en GNU Radio permite la decodificación de señales de TV partiendo de una señal OFDM en bandabase correspondiente a la norma ISDB-T y realizando el procesamiento en SW sobre una PC. La señal puede ser una grabación guardada en un archivo o una que demodule a bandabase la placa bladeRF.

En la Figura F.2 se observa un esquema de la implementación en SW haciendo uso de la placa BladeRF. Particularmente se indican los bloques a pasar a HW.

El sistema a desarrollar será capaz de realizar algunas etapas del procesamiento de la demodulación OFDM de la señal en bandabase obtenida del HW. El sistema estará implementado parcialmente sobre la placa bladeRF. Se utilizará el FPGA de la misma para alojar los bloques Auto-Correlación, FFT y Ajuste. Por lo tanto

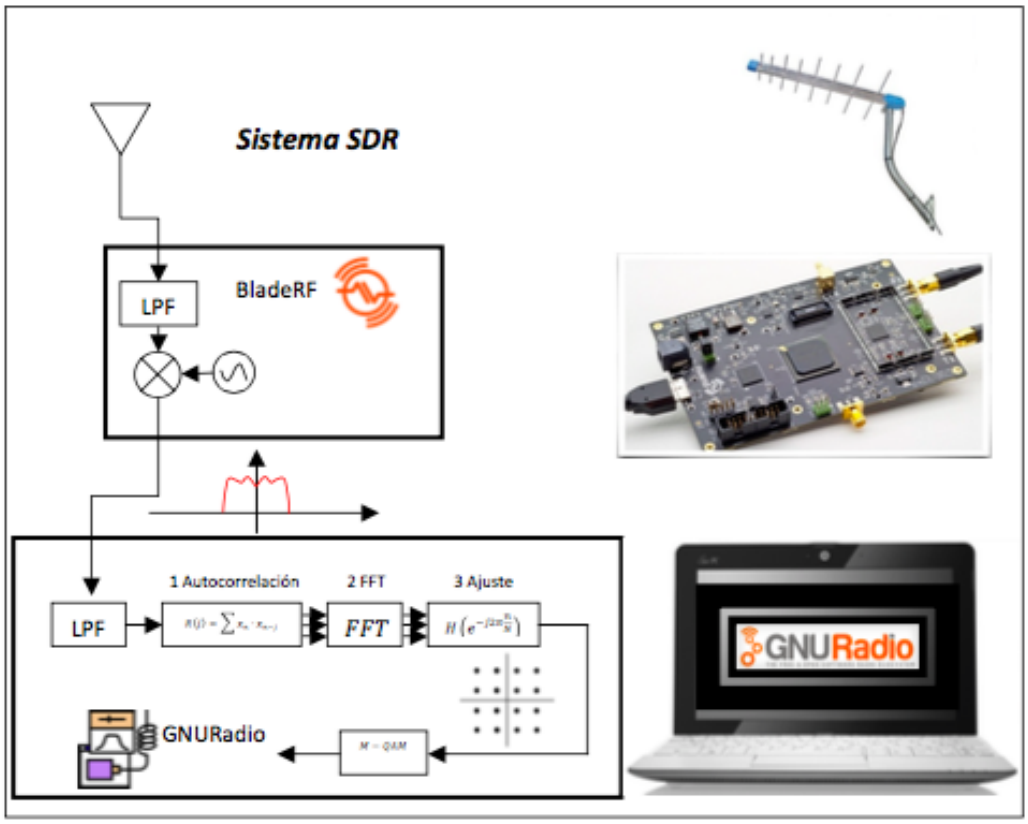


Figura F.2: Sistema SDR implementado en SW.

el sistema implementado permitirá realizar menos pasos siguientes en el SW para finalmente obtener la señal original.

En la Figura F.3 se observa un esquema de la implementación de los bloques Auto-Correlación, FFT y Ajuste desarrollados en el FPGA de la placa BladeRF.

F.10. Objetivos específicos

Los Objetivos específicos del Proyecto son los siguientes:

1. Completar la Capacitación en el uso de las herramientas de HW y SW para desarrollar el Proyecto: SDR, GNU Radio, OFDM, bladeRF y VHDL.

2. Definir la arquitectura de HW. Implementar y probar individualmente en la placa bladeRF los bloques: Bloque Auto-Correlación, Bloque FFT y Bloque Ajuste.

3. Prueba de integración del sistema SDR completo: definición de las pruebas y análisis de los resultados de las mismas.

4. Realizar la documentación y presentar los entregables para la aceptación del

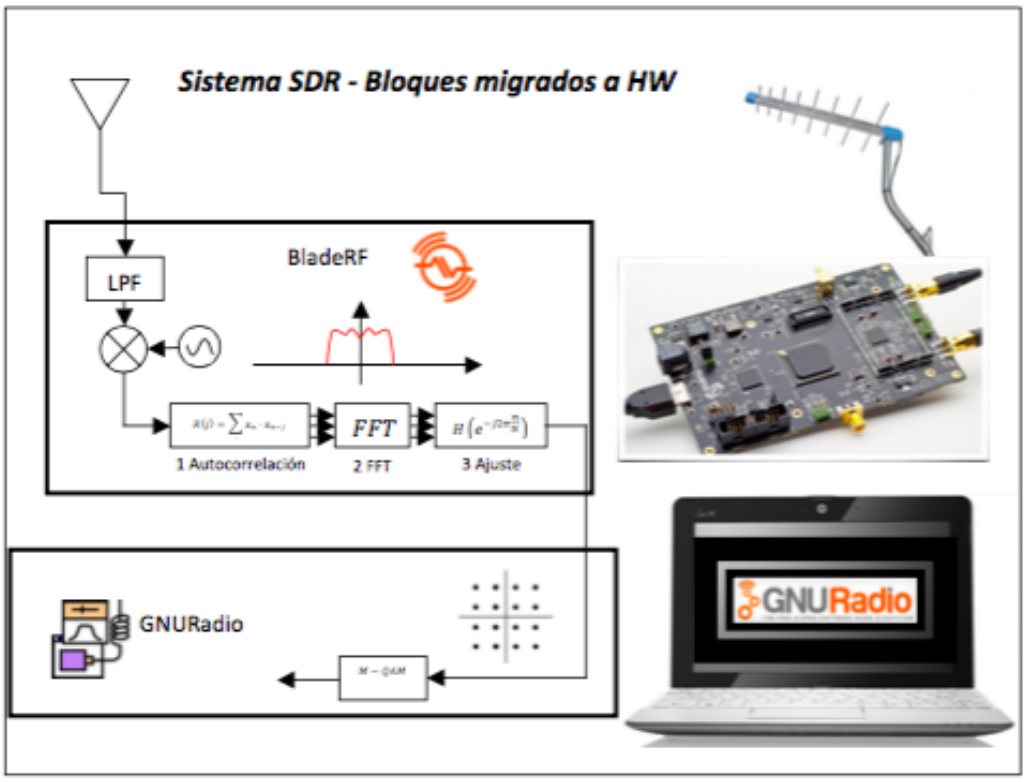


Figura F.3: Sistema SDR implementado en HW y SW.

Proyecto en las siguientes instancias: Hito 1, Hito 2 y Entrega de documentación final.

Se realizó el Plan de Administración del Proyecto (PAP) que permite definir de antemano varios elementos del Proyecto. Algunos de ellos son: objetivos, actores, supuestos, restricciones y tareas. Esto permite determinar el contexto en el que se realizará y su organización. El PAP será utilizado como herramienta para realizar el seguimiento y eventualmente se deberá redefinir algunos elementos de forma que permita alcanzar el objetivo general del Proyecto. Se realizarán entregables que permitirán reflejar el avance en temas trascendentales a presentar en las instancias de los hitos. Se realizará la documentación de las diferentes tareas. Particularmente se documentará donde sea necesario desarrollar nuevas experiencias como por ejemplo los criterios elegidos para la implementación en HW. La documentación permitirá la aprobación posterior del Proyecto. Se realizará una primer etapa en la que se capacitará en el entendimiento del tipo de sistema a trabajar y uso de las herramientas de HW y SW. Esto permitirá la comprensión del sistema original, la especificación de la arquitectura de HW a utilizar y la especificación de los bloques de SW a pasar a HW. Una vez completada la capacitación será posible comenzar en la etapa de Implementación. Allí se definirá en primera instancia la arquitectura del sistema mediante la especificación de la arquitectura del FPGA y de la placa. Luego se realizará la especificación, implementación y prueba de cada bloque (unit

testing). Esto permitirá facilitar la implementación conjunta de los mismos. Será fundamental tener claro el flujo de datos entre bloques del HW y con el SW antes de implementar cada bloque por separado. Se realizará la prueba del sistema completo. Para esto será necesario definir el entorno en el que se realizará de forma que permita garantizar la viabilidad, definir el alcance y describir los resultados a registrar de la misma. Se analizará el profiling del nuevo sistema permitiendo evaluar la implementación realizada y en particular, compararla con el sistema original. A continuación se describen los entregables de las tareas asociadas a los objetivos específicos. El entregable de la Tarea Capacitación será la documentación de la misma (incluyendo los puntos SDR, GNU Radio, OFDM, VHDL y bladeRF). Se caracterizará por brindar los elementos necesarios para lograr conocer y aprender a usar las herramientas descritas. Podrá ser aprobado si contiene los elementos fundamentales que describen cada uno de los puntos y particularmente los detalles no documentados en otros medios, que influirán en la realización del Proyecto. La Tarea Implementación tendrá como entregables la especificación y documentación de la arquitectura del sistema y de los bloques que la conforman. Tendrá como característica la descripción de la arquitectura del FPGA de la placa y de los bloques desde el punto de vista general de su funcionamiento. Además debe contener la especificación de cada bloque por separado de forma que quede determinada la característica de cada uno de forma independiente. La Tarea de Prueba tendrá como entregables la documentación conteniendo la especificación, análisis y resultados de la prueba del sistema completo. La especificación de la misma será tal que sea posible reproducirla con la información aportada en la documentación. El análisis contendrá las características de la prueba, resultados y comparación con el sistema original. Podrá ser aprobado si contiene las características generales en las que se desarrolló la prueba y los detalles que permitieron realizarla con todos los bloques en simultáneo. Además deberá contener una comparación objetiva con el sistema original de la cual se deriven conclusiones sobre las ventajas y desventajas de la implementación. Los Hitos tendrán los siguientes entregables:

Hito 1:

- Arquitectura del sistema: Especificación bloques de la arquitectura del FPGA y cómo se incorporan a la arquitectura que tiene la placa bladeRF.
- Especificación del Bloque de Correlación.

Hito 2:

- Especificación Bloques FFT y Ajustes.
- Especificación de la prueba del sistema completo.

F.11. Tareas

En la Figura F.4 se observa el primer nivel del WBS correspondiente al Proyecto.

Figura F.4: WBS: Estructura de Tareas, Primer nivel.

F.12. Análisis de Costos

Realizamos una estimación de ingresos asociados a la dedicación horaria en el Proyecto. Fijamos de forma arbitraria un monto de $100 (cien pesos uruguayos) por hora de dedicación de cada recurso humano (hora hombre). Por lo tanto estimamos un ingreso total de $40000 para cada recurso.

F.13. Análisis de Riesgos

A continuación se presentan posibles riesgos del Proyecto. Se estima el nivel de impacto y probabilidad de ocurrencia de cada uno y se asocia un plan de respuesta si corresponde. Los riesgos identificados a priori son los siguientes:

1. **R1** El HW (Placa Blade RF) estará disponible más tarde de lo esperado (30/04/2015).
2. **R2** El HW contendrá averías que harán imposible su utilización.
3. **R3** El HW no podrá contener los bloques definidos en SW.
4. **R4** La descripción de los bloques de SW requiere más tiempo de lo supuesto debido al tiempo necesario para interpretar el código C.
5. **R5** La implementación en VHDL no será posible para todos los bloques.
6. **R6** Un recurso estará menos tiempo del disponible.
7. **R7** Las tareas pueden retrasarse dentro de los intervalos comprendidos por los buffers.
8. **R8** Las tareas pueden retrasarse rebasando los buffers.

En la Tabla F.1 se cuantifican los riesgos y se asocia un plan de respuesta a cada uno.

La Tabla F.2 contiene la distribución de riesgos según Probabilidad de ocurrencia y Nivel de impacto.

En la Tabla F.3 se cuantifican los riesgos luego del plan de respuesta.

La Tabla F.4 contiene la distribución de riesgos luego del plan de respuesta.

Riesgo	Descripción	Probabilidad de ocurrencia	Nivel de Impacto	Plan de Respuesta
R1	HW demorado	Moderado	Medio	Realizar otras tareas previas
R2	HW averiado	Poco probable	Extremo	Utilizar otro ejemplar de HW
R3	HW incompatible	Poco probable	Alto	Utilizar otro tipo de HW
R4	SW difícil de interpretar	Moderado	Medio	Recurrir a ayuda
R5	VHDL no adecuado	Moderado	Extremo	Elegir bloques a implementar
R6	Recurso menor al 100 %	Poco probable	Bajo	Se debería redefinir duración de tareas
R7	Buffer disponible	Muy probable	Ninguno	–
R8	Buffer lleno	Poco probable	Alto	Se debería redefinir fechas y entregables

Tabla F.1: Evaluación de riesgos y plan de respuesta.

		Probabilidad de ocurrencia		
		Poco probable	**Moderado**	**Muy probable**
Nivel de Impacto	**Ninguno**			R7
	Bajo	R6		
	Medio		R2 - R4	
	Alto	R3 - R8		
	Extremo	R2	R5	

Tabla F.2: Distribución de riesgos.

Riesgo	**Descripción**	**Plan de Respuesta**	**Cambiar WBS**	**Probabilidad de ocurrencia**	**Nivel de Impacto**
R1	HW demorado	Realizar otras tareas previas que no dependan del HW	SI	Moderado	Bajo
R2	HW averiado	Utilizar otro ejemplar de HW de la misma marca (implica tiempo de espera en la compra)	SI	Poco probable	Medio
		Utilizar otro ejemplar de HW de otra marca (implica re-capacitación)	SI	Poco probable	Medio
R3	HW incompatible	Utilizar otro ejemplar de HW de otra marca: implica re-capacitación	SI	Poco probable	Medio
R4	SW dificil de interpretar	Recurrir a ayuda	SI	Poco probable	Medio
R5	VHDL no adecuado	Elegir bloques a implementar	SI	Moderado	Medio
R6	Recurso menor al 100 %	Se debería redefinir duración de tareas	SI	Poco probable	Bajo
R7	Buffer disponible	–	No	Muy probable	Ninguno
R8	Buffer lleno	Se debería re-definir fechas y entregables	SI	Poco probable	Medio

Tabla F.3: Evaluación de riesgos luego del plan de respuesta.

		Probabilidad de ocurrencia		
		Poco probable	**Moderado**	**Muy probable**
Nivel de Impacto	**Ninguno**			R7
	Bajo	R6	R1	
	Medio	R2 - R3 - R4 - R8	R5	
	Alto			
	Extremo			

Tabla F.4: Distribución de riesgos luego del plan de respuesta.

Apéndice G

Datos de horas y evaluación de gestión

Al comienzo del proyecto se evaluó la dedicación mensual que este llevaría en función de las tareas definidas en ese punto y del tiempo disponible para el proyecto estipulado por los créditos otorgados por la materia. Este estimaba una dedicación total de 1120 horas grupales ocupadas hasta el mes de Abril. Esta primer estimación estuvo equivocada por los siguientes puntos:

- No se tuvo en cuenta el tiempo final que se le debería dedicar al cierre del proyecto el cual está compuesto por las siguientes tareas: preparación de la demo, preparación de la defensa, preparación (póster y demo para el stand) y participación de la Ingeniería de Muestra así como también realización de la documentación final para el cierre de la materia (correcciones finales de la documentación, artículo y CD). El grupo le dedicó aproximadamente 300 horas a las mismas.

- Surgieron una serie de imprevistos a lo largo del proyecto que implicaron mayor dedicación a ciertas tareas. Por ejemplo: etapa de familiarización de la placa y del algoritmo a implementar. Estas tareas duraron el doble de lo previsto, consumiendo 100 horas más de las planificadas.

- No se tuvo en cuenta que debería implementarse un protocolo de comunicaciones entre la placa y la PC. Tarea que se estima en una dedicación grupal de 400 horas.

- En cuanto a la implementación y etapa de prueba en el FPGA se había estimado un total de 600 horas para el pasaje de tres bloques. Se utilizaron 800 horas para el pasaje de tan sólo un bloque.

En función de como se dio el proyecto fue necesario solicitar prórroga de 1 mes.

Se concluye que la falta de experiencia llevó a que se desestimara la duración de ciertas tareas así como también que se prolongaran algunas. Por un lado en la etapa de implementación pero también en la parte administrativa y cierre.

Mes	**Dedicación Mensual (hs/mes)**
Marzo	170
Abril	73
Mayo	50
Junio	50
Julio	79
Agosto	64
Septiembre	155
Octubre	124
Noviembre	118
Diciembre	48
Enero	51
Febrero	117
Marzo	169
Abril	280
Mayo	280
Junio	200
Julio	10
Agosto	80
Septiembre	80
Octubre	44
Total	**2124**

Tabla G.1: Dedicación (hs/mes) (Horas de ambos integrantes del grupo)

Bibliografía

[1] Altera. *Cyclone IV FPGA Device Family Overview*. URL: https://www.altera.com/en_US/pdfs/literature/hb/cyclone-iv/cyiv-51001.pdf.

[2] Altera. *Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide*. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_ram_rom.pdf.

[3] Altera. *Logic Analyzer*. URL: ftp://ftp.altera.com/up/pub/Tutorials/DE2/Digital_Logic/tut_signaltapII_vhdlDE2.pdf.

[4] Altera. *ModelSim Altera*. URL: https://www.altera.com/products/design-software/model---simulation/modelsim-altera-software.html.

[5] Altera. *Quartus II Web Edition*. URL: http://dl.altera.com/?edition=lite.

[6] Altera. *SignalTap II*. URL: ftp://ftp.altera.com/up/pub/Tutorials/DE2/Digital_Logic/tut_signaltapII_vhdlDE2.pdf.

[7] Altera. *USB Blaster*. URL: https://www.altera.com/products/boards_and_kits/download-cables.html.

[8] FING ARTES - IIE. *Análisis de Redes, Tráficos y Estadísticas de Servicios*. URL: http://iie.fing.edu.uy/investigacion/grupos/artes/.

[9] FING ARTES - IIE. *ISDB-T receiver in GNU Radio*. URL: https://github.com/git-artes/gr-isdbt.

[10] Scott Chacon. *Pro Git*. URL: https://git-scm.com/book/es/v1.

[11] Cypress. *Designing a GPIFTM II Master Interface*. URL: http://www.cypress.com/file/124206/download.

[12] Cypress. *Hoja de datos del CYUSB301X*. URL: http://www.cypress.com/documentation/datasheets/cyusb301x-cyusb201x-ez-usb-fx3-superspeed-usb-controller.

[13] *"EN 300 744 V1.6.1. " 2009. Framing structure, channel coding and modulation for digital terrestrial television.* URL: www.etsi.org/deliver/etsi_en/300700_300799/300744/01.06.01_60/en_300744v010601p.pdf.

[14] Wikipedia the free encyclopedia. *Software-defined radio.* URL: https://en.wikipedia.org/wiki/File:SDR_et_WF.svg.

[15] ETTUS. *USRP X310.* URL: https://www.ettus.com/product/details/X310-KIT.

[16] *GNU Radio. GNU Radio Manual and C++ API Reference.* URL: http://gnuradio.org/doc/doxygen/.

[17] P. Borjesson J.-J. van de Beek M. Sandell. "ML Estimation of Time and Frequency Offset in OFDM Systems". En: *Signal Processing, IEEE Transactions* 45.7 (1997), págs. 1800-1805.

[18] Federico Larroca, Pablo Flores-Guridi, Gabriel Gómez, Víctor González Barbone y Pablo Belzarena. "An open and free ISDB-T full seg receiver implemented in GNU radio". En: *Wireless Innovation Forum Conference on Wireless Communications Technologies and Software Defined Radio (WInnComm 16), Reston, Virginia, USA, 15-17 mar.* 2016, págs. 1-10. URL: http://iie.fing.edu.uy/publicaciones/2016/LFGGB16.

[19] Nuand LLC. *bladeRF Schematics.* URL: https://nuand.com/bladerf.pdf.

[20] Nuand LLC. *bladeRF Software Defined Radio.* URL: https://nuand.com.

[21] Nuand LLC. *bladeRF Software Defined Radio - Presentación de la placa.* URL: http://nuand.com/bladeRF-brief.pdf.

[22] Nuand LLC. *bladeRF USB 3.0 Superspeed Software Defined Radio Source Code.* URL: https://github.com/Nuand/bladeRF.git.

[23] Lime Micro. *Calibration Guide del LMS6002D.* URL: http://www.limemicro.com/download/LMS6002Dr2-Programming_and_Calibration_Guide-1.1r1.pdf.

[24] Lime Micro. *Hoja de Datos del LMS6002D.* URL: http://www.limemicro.com/download/LMS6002Dr2-DataSheet-1.2r0.pdf.

[25] JICA Expert Nobuyuki Sato. *Basic technology of ISDB-T.* URL: https://eva.fing.edu.uy/course/view.php?id=662.

[26] GNU Octave. *Octave.* URL: https://www.gnu.org/software/octave/.

[27] Association of Radio Industries y Businesses. *Transmission System For Digital Terrestrial Television Broadcasting*. URL: www.arib.or.jp/english/html/overview/doc/6-STD-B31v2_2-E1.pdf.

[28] M. Rice. *Digital Communications: A Discrete-time Approach*. Pearson/Prentice Hall, 2009. ISBN: 9780130304971. URL: https://books.google.com.au/books?id=EB3r7JtXlWwC.

[29] RTL-SDR.com. *About RTL-SDR*. URL: http://www.rtl-sdr.com/about-rtl-sdr/.

[30] GNU Radio. *Software GNU Radio*. URL: http://gnuradio.org.

Esta página ha sido intencionalmente dejada en blanco.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el miércoles 19 octubre, 2016.
http://iie.fing.edu.uy/