Sistema de recuperación de sincronización con estimador de Maximum Likelihood y filtros polifásicos

Juan Tiago Ruiz Rodriguez
Proyecto Final Diseño Digital Avanzado
Fundación Fulgor
juantiagoruiz@gmail.com

***Resumen*— En este informe se describe la arquitectura, flujo de diseño y proceso de verificación para la implementación en FPGA de un circuito de recuperación de sincronización. Se logra la alineación de tiempo de muestras a través de la selección de fase de un filtro acoplado polifásico. La arquitectura presentada permite alinear la salida de datos al momento en que la muestra de la señal reconstruida se encuentra en su máxima amplitud (mayor apertura de ojo), corrigiendo la fase de la señal entrante y adecuándose a offsets y jitter ocasionados por la desincronización de los relojes.**

Se presenta el desarrollo de un sistema completo de recuperación de fase, consistente en un filtro polifásico de 64 fases para generar un desfase temporal por selección de fase, un interpolador polifásico de 16 fases que reconstruye la señal a partir de dos muestras entregadas por el primer filtro, y un lazo de realimentación conformado por un detector de error de temporización (TED), un filtro de lazo (loop filter) tipo PI, y un módulo de control que acumula el error y ajusta dinámicamente la fase seleccionada. El diseño fue implementado y verificado sobre FPGA, con foco en eficiencia de recursos y latencia.

Palabras clave—timing recovery, filtros polifásicos, recuperación de tiempo, recuperación de sincronización, TED d máxima similitud, polyphase interpolation.

I. INTRODUCCIÓN

Los sistemas modernos de comunicaciones digitales síncronas requieren la adquisición y seguimiento tanto de la portadora como del reloj de temporización desde la señal recibida, sin que estos estén explícitamente presentes. En particular, la sincronización en tiempo —también conocida como recuperación de sincronización (timing recovery)— es esencial para lograr una correcta demodulación de la señal digital.

En esta aplicación nos enfocamos en la recuperación de temporización, utilizando una arquitectura que aprovecha el sobremuestreo típico en sistemas QAM modernos (2 a 4 muestras por símbolo) para implementar interpoladores digitales capaces de estimar la fase óptima de muestreo. Este sobremuestreo no solo permite cumplir con el criterio de Nyquist, sino también corregir digitalmente errores de fase debidos a offsets y jitter en la señal recibida.

La solución propuesta se inspira en las arquitecturas de recuperación de temporización desarrolladas por Fred Harris y Chris Dick, las cuales emplean filtros polifásicos para realizar una interpolación eficiente de la señal. En particular, se utiliza un módulo PRBS (Pseudo-Random Binary Sequence) de longitud 5 que a partir de una semilla genera un patrón de unos y ceros que se alimentan a un filtro polifásico transmisor de 64 fases que genera dos muestras por símbolo, desfasadas de acuerdo con la fase seleccionada. Estas dos muestras alimentan un segundo filtro polifásico receptor de 16 fases, que realiza la interpolación para reconstruir la muestra óptima de la señal. A partir de esta muestra y su derivada, un detector de error de temporización basado en el criterio de máxima verosimilitud (maximum likelihood) estima el error de fase, el cual es filtrado y acumulado por un lazo de control que actualiza dinámicamente el índice de fase utilizado en el filtro interpolador receptor.

Finalmente como salida del sistema se obtiene el dato que corresponde a la mayor amplitud de ojo alcanzada en la interpolación que corresponde a la muestra más óptima para ser procesada por el detector de símbolos.

La arquitectura general del sistema se puede observar en la figura 1.

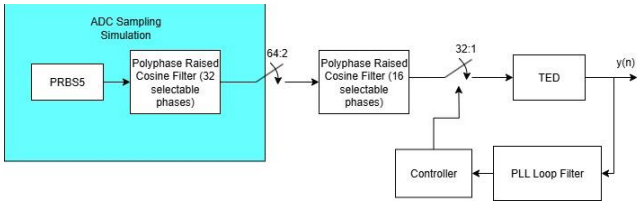

Figura 1. Arquitectura general del sistema.

A efectos de una mejor visualización y entendimiento de los filtros polifásicos, los mismos funcionan en todas sus fases y a través de muestreadores se realizan los muestreos a través de diezmados para conectar sus salidas a los módulos contiguos.

Desde el controlador se utilizan clock enables y contadores para la selección de fase de los filtros polifásicos y la habilitación y procesamiento de las muestras, y se realiza el control para los muestreos tanto en la fase seleccionada del primer filtro polifásico como de la fase definida por el error acumulado para el filtro polifásico correspondiente a lo que sería el "receptor".

La detección del error de tiempo (TED) se logra utilizando una técnica de máxima verosimilitud que es óptima para su incorporación a las arquitecturas de filtros polifásicos.

Esta arquitectura une el proceso de interpolación con el de modelado del pulso, de esta manera los distintos caminos del filtro polifásico representan una colección de filtros acoplados a distintos offset de tiempo entre las posiciones de la muestra de entrada y la posición del valor de correlación del pico de la muestra de salida. El proceso de recuperación de tiempo solo debe determinar qué conjunto de coeficientes del filtro coincide o está mas cercano al offset entre las muestras de entrada y de salida. Utiliza un proceso de realimentación con phase locked loop (PLL) para determinar la fase correcta.

El sistema se implementa en una FPGA AMD Artix 7 35T, acompañado de simulación en software en punto flotante para

prueba de la arquitectura planteada y de una simulación en punto fijo para correlación de vectores con las salidas obtenidas de la FPGA.

II. DESCRIPCIÓN GENERAL DEL SISTEMA

En esta sección se explicará la función de cada bloque de la figura 1 a alto nivel.

A. PRBS

El módulo PRBS es el encargado de generar el patrón de bits que luego se modula en el sistema, en este caso elegimos una PRBS de longitud 5, para tener un patrón que se repita constantemente (cada 31 bits) y poder generar una salida autónoma de muestras al habilitar el sistema.

B. Filtro polifásico transmisor.

Se utiliza un filtro polifásico transmisor con un sobremuestreo grande para tener distintos *timing offsets* de los que seleccionar como estímulo.

Este filtro es el encargado de sobremuestrear las muestras de entrada generadas por el módulo PRBS y realizarle modelado de pulso para su transmisión. El mismo posee 64 fases de 12 coeficientes cada una, estos 64 conjuntos de coeficientes permiten que, al seleccionarlos de a pares espaciados en 32, se formen dos muestras por símbolo que poseen distintos delays fraccionales de muestras. Esto quiere decir que de todas las muestras de salidas generadas por el filtro, que se pueden observar en la figura 2.

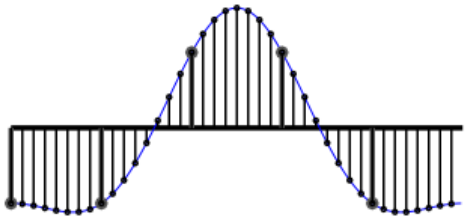

Figura 2. Muestras de salida de filtro polifásico interpolador.

Se seleccionan las deseadas a partir del desfase que deseemos generar, y se descartan el resto. En el ejemplo de la figura 3, para un diezmado con offset de 1/8 se seleccionan las muestras marcadas en rojo

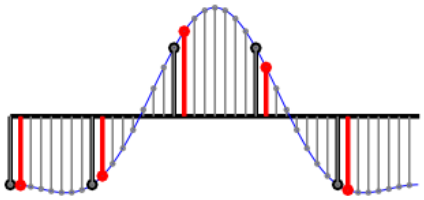

Figura 3. Muestras en rojo correspondientes a salida de una fase.

Para generar un diezmado con offset de 2/8 se seleccionan las muestras de color verde, como indica la figura 4.

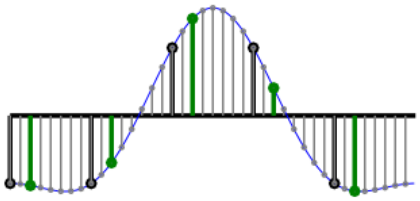

Figura 4. Muestras en verde correspondientes a salida de una fase.

Para generar un diezmado con offset de 3/8 se seleccionan las muestras de color azul, como indica la figura 5.

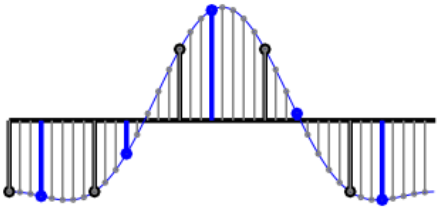

Figura 5. Muestras en azul correspondientes a salida de una fase.

Y así sucesivamente con las 32 fases seleccionables, ya que cada conjunto de muestras se corresponde a un conjunto de coeficientes del filtro interpolador. Esta selección de fase se controla a partir de una entrada al sistema que nos permite retrasar la señal por una cantidad determinada menor a una muestra.

El diezmado de la salida del filtro transmisor se realiza a través de un *sample and hold* que realiza la captura e inserción de ceros en la señal cuando se activa un enable que indica que el conjunto de los coeficientes que producen la salida actual del filtro se corresponde al delay deseado en el sistema.

Este conjunto de la PRBS junto al filtro polifásico transmisor forma una emulación de muestreado a través de un ADC de una señal analógica donde desconocemos el timing del reloj transmisor y se obtienen dos muestras por símbolo.

C. Filtro polifásico receptor.

Una vez obtenidas las dos muestras por símbolo recibidas, el primer paso para la recuperación de tiempo es la reconstrucción de la señal original. Para este proceso utilizamos nuevamente un filtro polifásico donde se realiza el proceso de interpolado y filtrado de la señal para obtener la reconstrucción. Se aplica un filtro interpolador de 192 coeficientes distribuidos en 16 fases con 12 coeficientes cada una que trabaja sobre cada una de las muestras obtenidas como salida del filtro anterior.

Luego de la obtención de la señal reconstruida se trabajará la salida de este filtro para obtener cuál de las fases corresponde al valor de máxima apertura de ojo, el cuál será seleccionado como salida del sistema para su posterior procesado por un detector de símbolo.

En la figura 6 se puede observar cómo los distintos conjuntos de coeficientes generan distintos valores de muestras de los cuales uno será el buscado como valor máximo.

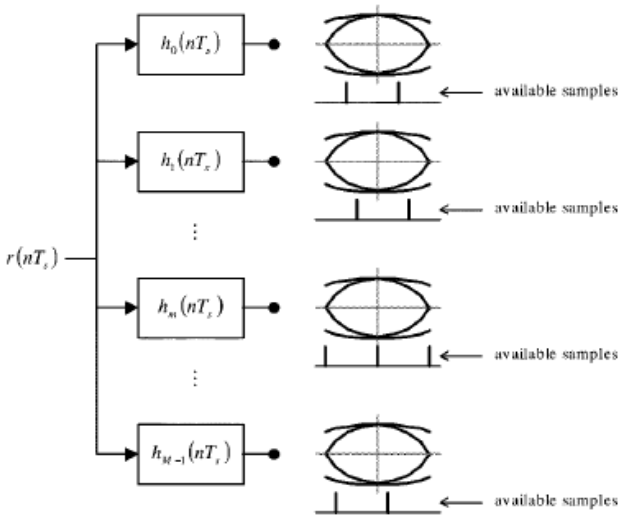

Figura 6. Instantes de muestreo disponibles a la salida de cada fase del filtro interpolador.

D. TED.

Para el detector de error de tiempo se eligió uno que pudiera acoplarse eficientemente a la arquitectura propuesta, de esta manera se propone un TED de máxima similitud (Maximum likelihood, ML) el cual consiste en tomar la salida de tres filtros, el seleccionado de amplitud óptima, y las fases anterior y posterior, por lo que el error está dado por:

$$e(n) = (y_{m+1}(n) - y_{m-1}(n)) * y_m(n)$$

Donde m es la fase seleccionada en nuestro filtro polifásico.

De este modo, la resta entre la fase posterior y anterior genera una aproximada de la derivada, que multiplicada por el signo de la fase seleccionada nos dará un valor de error que tienda a la amplitud máxima del ojo, para los cuatro casos posibles ilustrados en la figura 7.

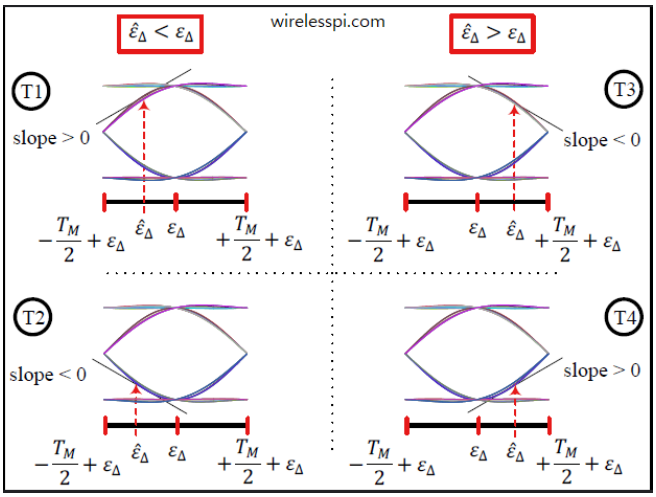

Figura 7. Posibles resultados de salida del TED para estimación y corrección de sincronización.

E. Filtro de lazo.

El filtro de lazo se seleccionó con una arquitectura con coeficiente proporcional e integral para poder filtrar el error proveniente del TED. Produce un acondicionamiento de la señal de error para mejor control de la fase a seleccionar.

F. Controlador de fase.

El sistema se controla en su totalidad desde un módulo top que instancia los distintos módulos y forma lógica de control para los mismos. Este módulo es el encargado de generar los muestreos de las salidas, diezmados, instanciar contadores, conectar señales de habilitación entre las distintas partes y genera una acumulación progresiva del valor de error de la salida del filtro de lazo para poder seleccionar la fase correcta del filtro de recepción, que es el momento en el cual la amplitud de la salida es óptima para su selección.

III. IMPLEMENTACIÓN EN PUNTO FLOTANTE Y PUNTO FIJO

Se realizó una simulación inicial en punto flotante en Python para prueba de la arquitectura y funcionamiento matemático del sistema. La misma se encuentra en el archivo "timing_recovery_simulation_FP_FXP.ipynb".

Inicialmente se realiza el cálculo de los coeficientes de los filtros a utilizar a través de un script de Matlab, cuyo output nos genera ambos filtros separados en fases listos para ser copiados y pegados en la simulación de Python.

El script de Python comienza con la definición de los coeficientes en punto flotante, tanto del filtro de transmisión como del de recepción, graficados en las figuras 8 y 9.

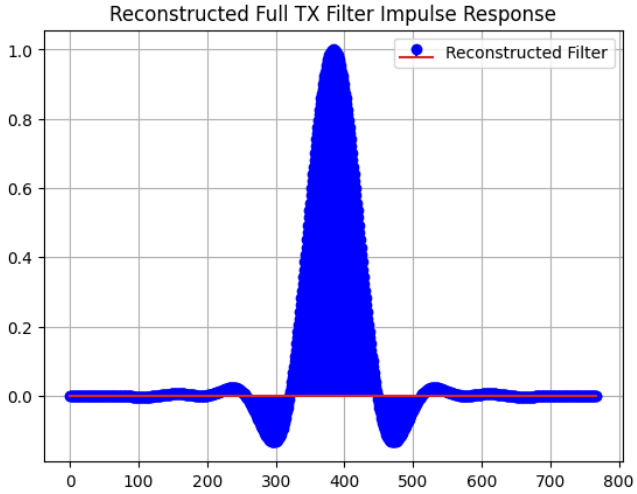

Figura 8. Coeficientes de filtro de transmisión (TX).

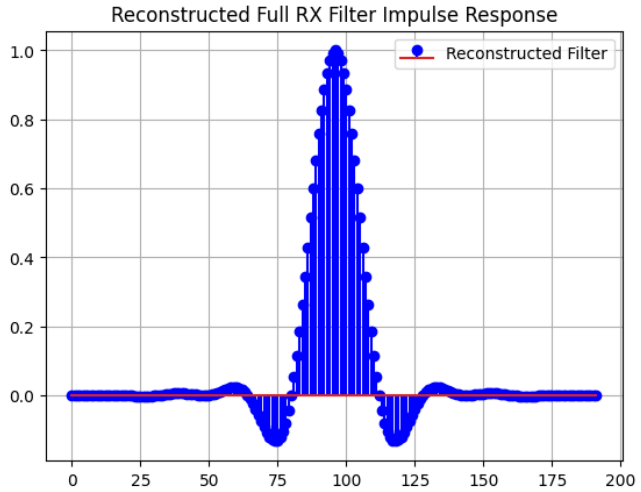

Figura 9. Coeficientes de filtro de recepción (RX).

Posteriormente se genera un patrón de símbolos que corresponden al patrón generado por la PRBS. Se realiza la interpolación con el filtro de transmisión, convolucionando el patrón de la PRBS (con ceros insertados para el sobremuestreo) y los coeficientes del filtro transmisor. El resultado se puede observar en la figura 10.

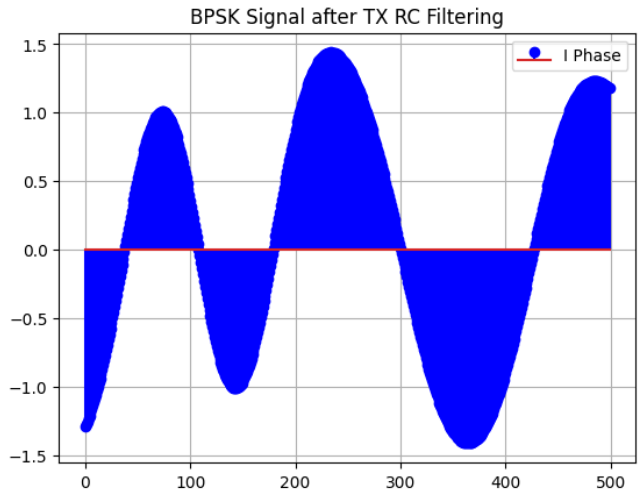

Figura 10. Salida del filtro polifásico de transmisión.

Luego se realiza una selección de fase a través de la selección de muestras espaciadas por un paso (step) determinado por la relación entre las muestras por símbolo del filtro (fases) y las deseadas para el procesamiento. Esto produce un efecto de diezmado de la señal.

```
i_in = 0
phase = 12
SPS = 64
SPS_signal = 2
step = int(SPS/SPS_signal)
I_phaseX = QPSK_I_SRRC[phase::step]
```

El resultado de este muestreo se puede observar en las figuras 11 y 12.

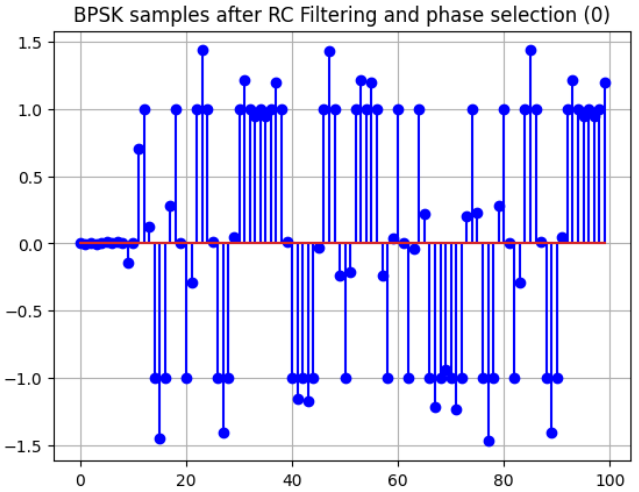

Figura 11. Muestras de salida del filtro polifásico de transmisión para un offset 0.

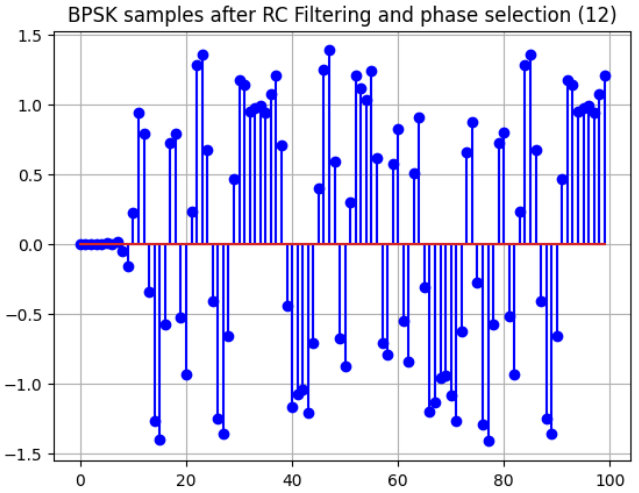

Figura 12. Muestras de salida del filtro polifásico de transmisión para un offset 12.

Acto seguido se realiza la reconstrucción de la señal original a partir de las muestras seleccionadas, esta selección permite generar distintos offset de tiempo que serán los recuperados. Se realiza mediante la convolución de las muestras seleccionadas con inserción de ceros y los coeficientes del filtro receptor.

En las figuras 13, 14 y 15 se pueden observar los resultados de este proceso.

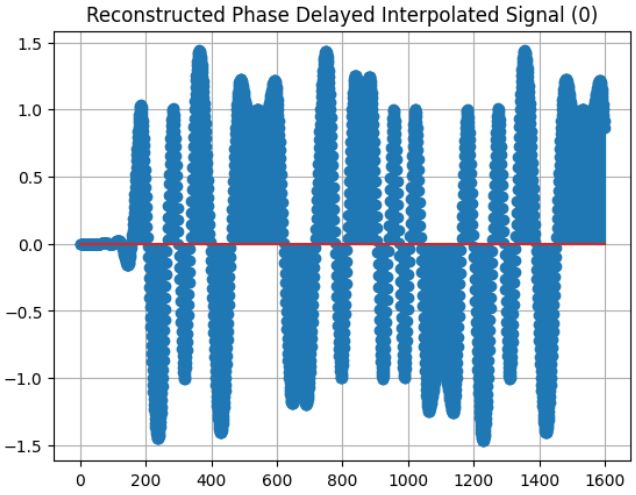

Figura 13. Señal reconstruida (interpolada) en salida del filtro polifásico de recepción para un offset 0.

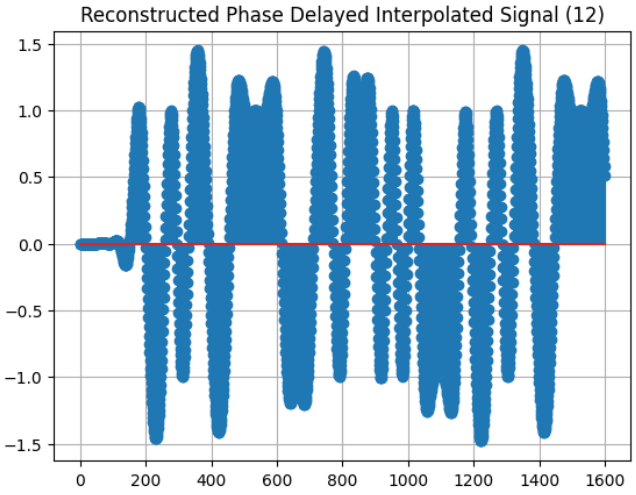

Figura 14. Señal reconstruida (interpolada) en salida del filtro polifásico de recepción para un offset 12.

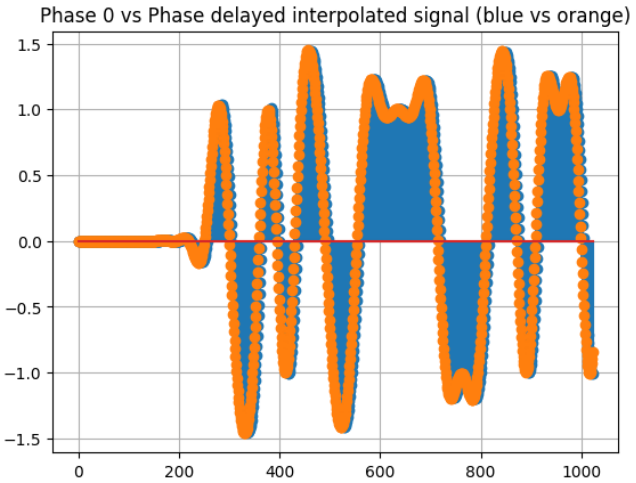

Figura 15. Comparativa para observar el delay generado por la selección de fases distintas.

Luego se recorre toda la señal reconstruida tomando las muestras correspondientes a la fase seleccionada por el acumulador mu:

```
while i_out < len(samples) and i_in+interpol < len(samples):
    out[i_out] = samples_interpolated[i_in*interpol + int(mu*interpol)] # tomamos lo que creemos que es el mejor sample

    mu_index = int((mu*16))
    x = samples_interpolated[i_in*16+int(mu*16)] * (samples_interpolated[i_in*16 +1+int(mu*16)]-samples_interpolated[i_in*16 -1+ int(mu*16)]) #TED

    mm_val = np.real(x
    mm_val_int = mm_val*0.001 + mm_val_int # termino integral
    mu += SPS + 0.3*mm_val+mm_val_int
    i_in += int(np.floor(mu)) # redondeamos al int mas cercano ya que es un index
    mu = mu - np.floor(mu) # sacamos la parte entera de mu
    i_out += 1 # incrementamos en 1 el indice de salida
```

Aquí el valor de la variable mu converge hacia un valor donde el error calculado por el TED se vuelve estable, de manera que los resultados obtenidos por el procesamiento de la señal se pueden visualizar en las siguientes figuras:

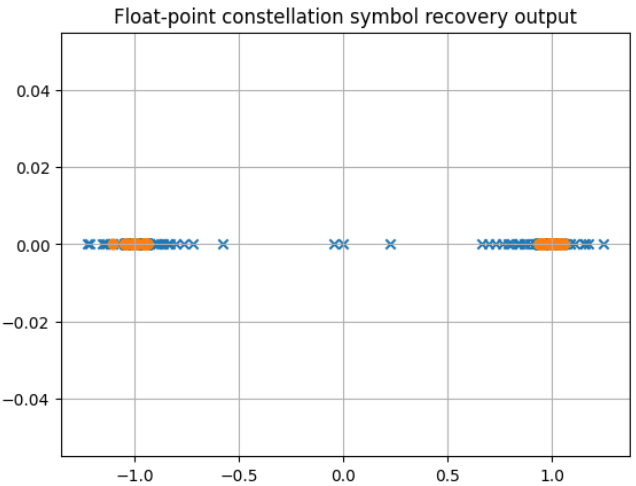

Figura 16. Constelación de salida de la señal recuperada, graficadas en X azules las primeras 1500 muestras (previo a estabilización de los valores) y con O naranja las 2500 muestras restantes (valor estabilizado).

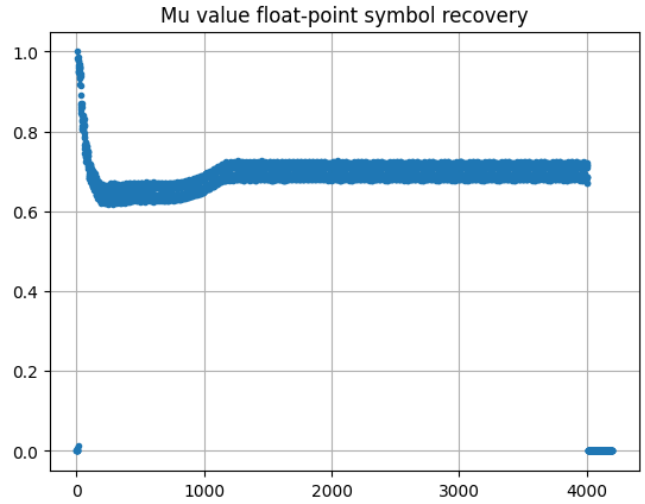

Figura 17. Valor del acumulador de fase (mu) a lo largo del recorrido por las muestras.

Se puede ver como el valor se estabiliza luego de un transitorio, la variabilidad del valor estabilizado está dada por el patrón generado por la PRBS (poca variabilidad de los valores) y los valores de Kp y Ki seleccionados para el filtro de lazo, que son 0.3 y 0.001 respectivamente. En la gráfica de la constelación, se encuentran marcadas con cruz azul las primeras 1500 muestras y con círculo naranja las restantes, donde podemos ver que una vez que se estabiliza el valor de mu, la salida se mantiene constante en torno a 1. Se podría haber buscado un valor de Kp que tuviese menor variabilidad en el valor final pero se buscó priorizar una relación entre rapidez de reacción del sistema y estabilidad final (sistemas con Kp más bajo respondían muy lentamente).

En la figura 18 se puede observar el diagrama de ojo despúes de la interpolación realizada en el filtro receptor, con líneas verdes verticales que indican los instantes de muestreo sin recuperación de sincronización.

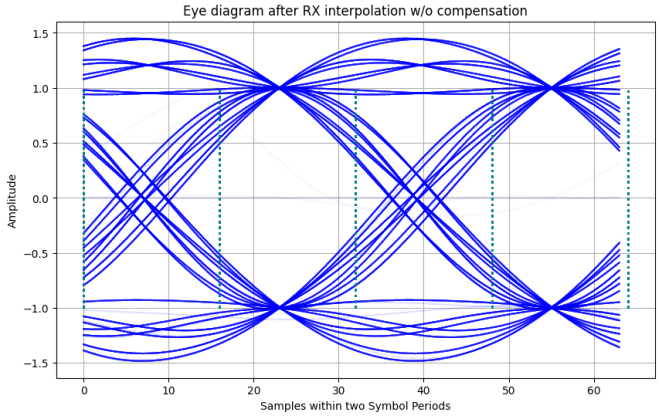

Figura 18. Diagrama de ojo despúes de interpolación RX sin sincronización.

En la figura 19 se puede observar el diagrama de ojo despúes de la interpolación realizada en el filtro receptor, con líneas verdes verticales que indican los instantes de muestreo, una vez realizada la recuperación de sincronización y teniendo una muestra por símbolo.

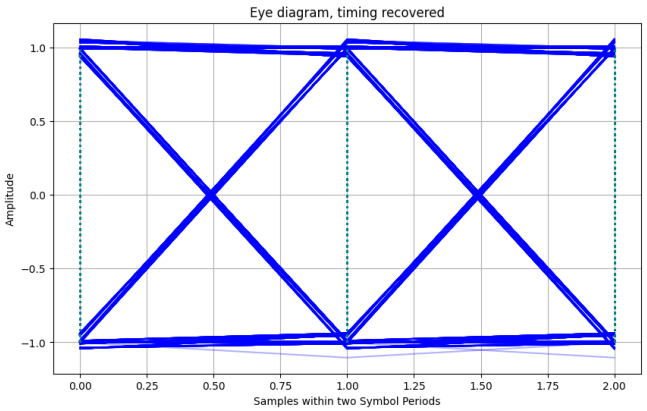

Figura 19. Diagrama de ojo despúes de interpolación RX con sincronización.

Se puede observar que el instante de muestreo se encuentra en el lugar óptimo (mayor apertura de ojo) luego de la sincronización.

Luego de tener los resultados para punto flotante, se realizó la simulación en punto fijo. La conversión de los distintos parámetros a punto fijo se realizó utilizando la biblioteca fxpmath, con el siguiente formato a modo de ejemplo:

```
h0_fxp = Fxp(h00, signed=True, n_word=coeff_NB, n_frac=coeff_NBF)
```

Para la señal de entrada se utilizó un formato de S(8,7),para las señales intermedias de datos se utilizó un formato S(9,7), debido a que esta cantidad de bits era suficiente para representar los valores que se buscaba tener en la señal (amplitud de -2 a 2, sin tanto detalle en cuanto a resolución).

Para el caso de los coeficientes si se optó por utilizar una resolución de S(13,12), ya que eran los que otorgaban una relación de SNR de al menos 40 dB para la representación de los coeficientes del filtro calculados. Los resultados fueron los siguientes:

En las figuras 20 y 21 se encuentran graficados los coeficientes de los filtros polifásicos de transmisión y recepción respectivamente, en representación en punto fijo.

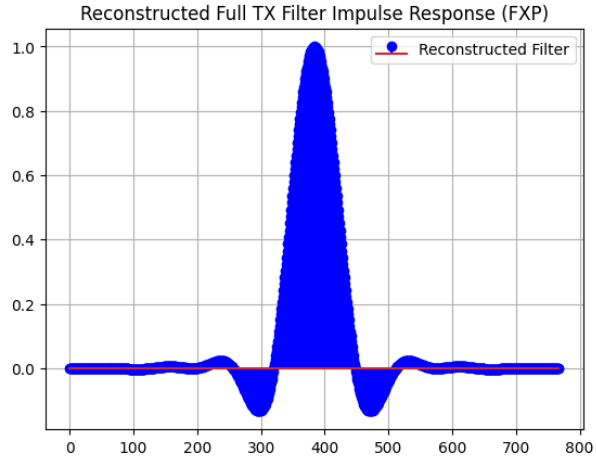

Figura 20. Coeficientes de filtro de transmisión (TX) en punto fijo.

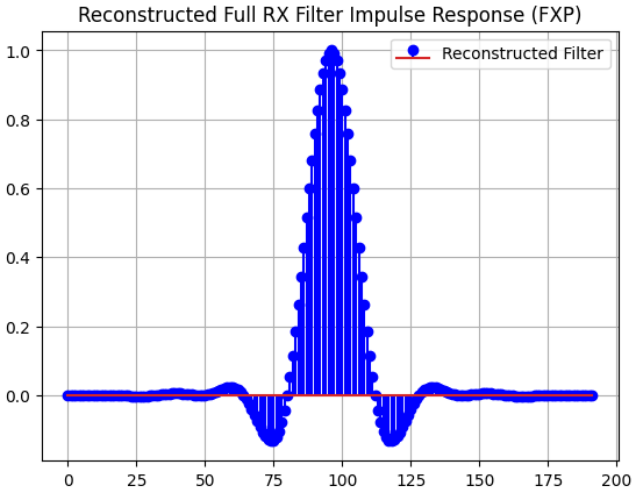

Figura 21. Coeficientes de filtro de transmisión (RX) en punto fijo.

En la figura 22 se puede observar la interpolación del patrón de bits obtenido de la PRBS por el filtro transmisor, en punto fijo.

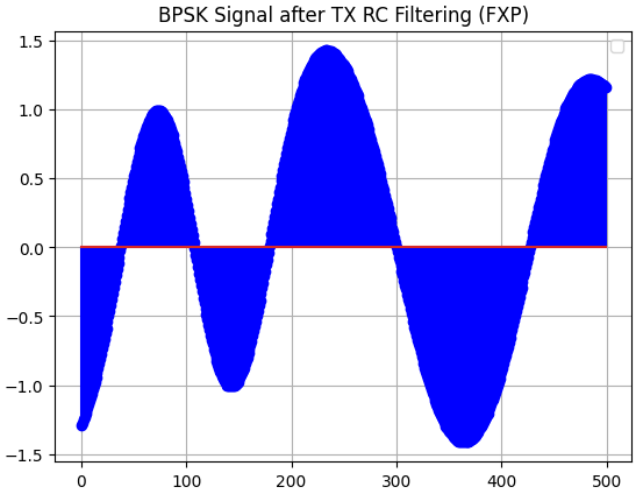

Figura 22. Salida del filtro polifásico de transmisión en punto fijo.

En las figuras 23 y 24 se puede visualizar el diezmado producido por la selección de muestras de la señal interpolada para generar distintos offset de tiempo.

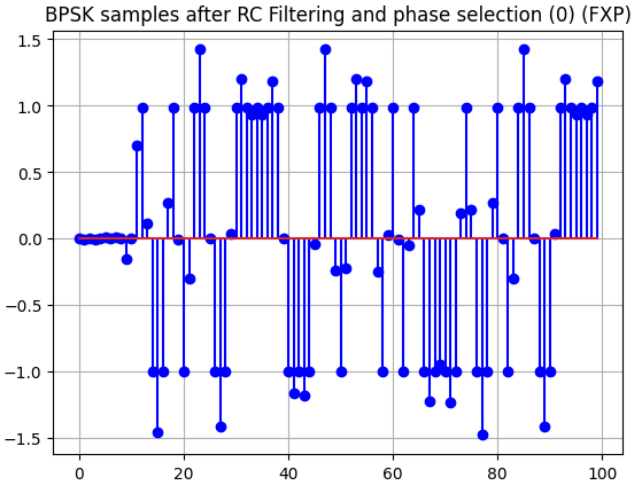

Figura 23. Muestras de salida del filtro polifásico de transmisión para un offset 0, en punto fijo.

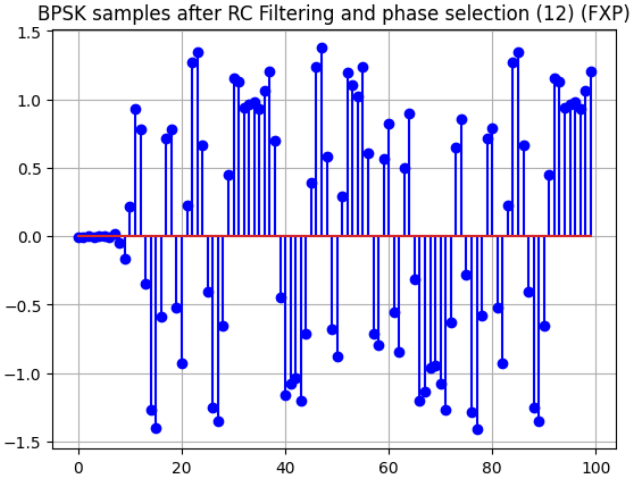

Figura 24. Muestras de salida del filtro polifásico de transmisión para un offset 12, en punto fijo.

Y en la figura 25 se puede observar la reconstrucción en la salida del filtro polifásico receptor de la señal original, comparando entre los dos offset mostrados en las figuras anteriores, para punto fijo.

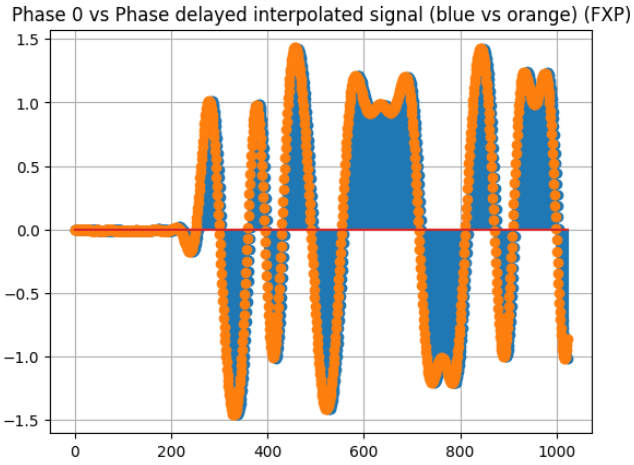

Figura 25. Comparativa entre señales de distinto offset, en punto fijo.

Finalmente, se observan en las figuras 26 y 27, tanto la constelación de símbolos obtenida de la simulación en punto fijo como el valor de la acumulación de fase (mu).

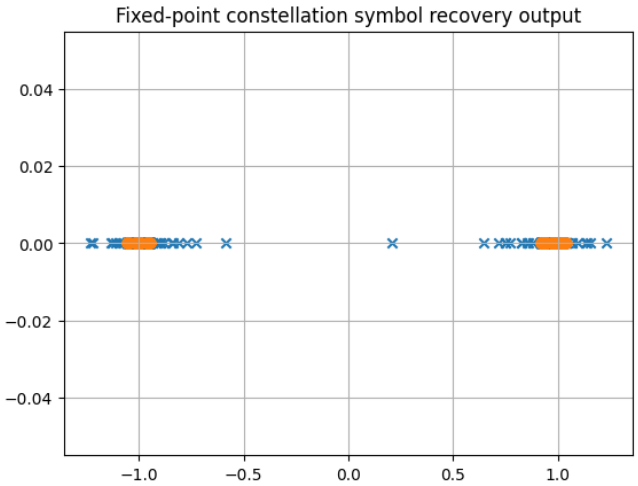

Figura 26. Constelación de salida de la señal recuperada en punto fijo, graficadas en X azules las primeras 1500 muestras (previo a estabilización de los valores) y con O naranja las 2500 muestras restantes (valor estabilizado).

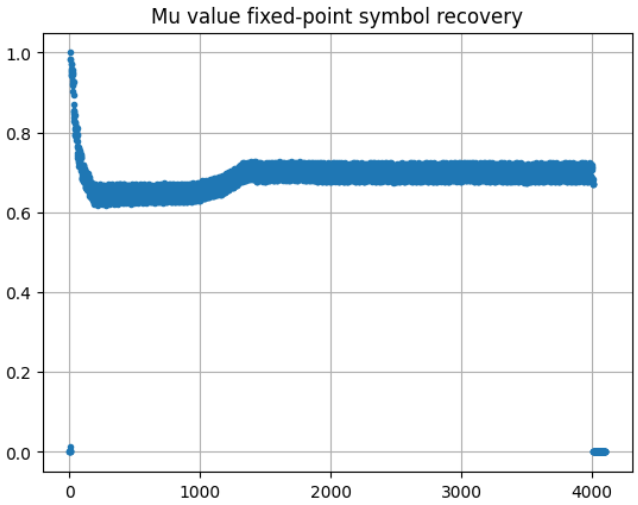

Figura 27. Valor del acumulador de fase (mu) en punto fijo a lo largo del recorrido por las muestras.

En comparación, para un desfase de 14 unidades, los resultados de la recuperación son más evidentes y se obtiene un diagrama de ojo perfecto a la salida. Esto se produce debido a la granularidad del filtro polifásico receptor que solo posee dieciséis fases para elegir. Se puede observar dicho diagrama de ojo y su constelación de símbolos en las figuras 28 y 29.

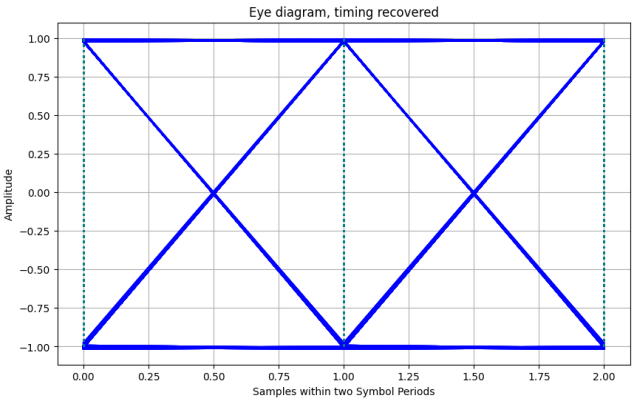

Figura 28. Diagrama de ojo después de interpolación RX con sincronización. Para desfase de 14.

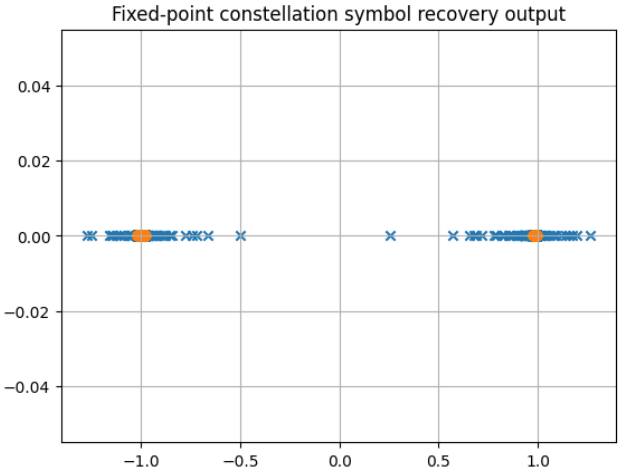

Figura 29. Constelación de salida de la señal recuperada en punto fijo, graficadas en X azules las primeras 1500 muestras (previo a estabilización de los valores) y con O naranja las 2500 muestras restantes (valor estabilizado). Para desfase de 14.

De igual modo, en las siguientes figuras se muestra el resultado de la simulación de distintos valores de coeficientes Kp, tanto para punto flotante como punto fijo. En las mismas se puede observar que el valor de convergencia es el mismo, aunque haya variaciones debido a la limitación de representación de valores dada por la cantidad de bits utilizada.

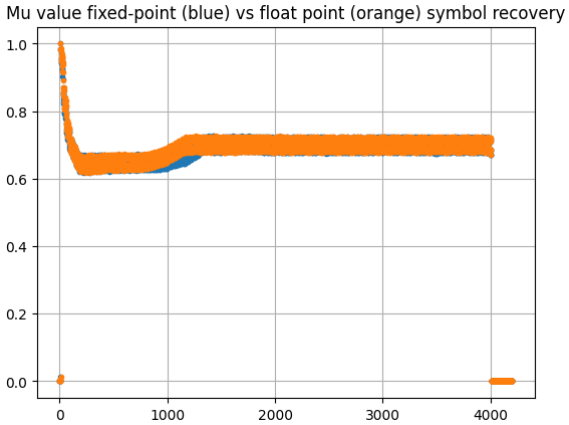

Figura 30. Valor del acumulador de fase (mu) en punto fijo contra punto flotante para Kp = 0.3.

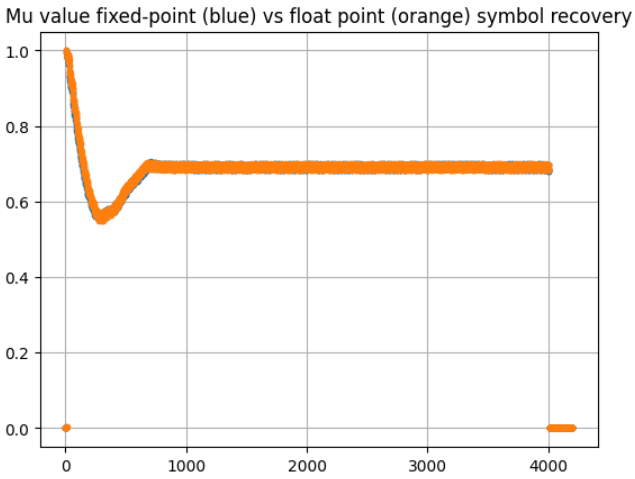

Figura 31. Valor del acumulador de fase (mu) en punto fijo contra punto flotante para Kp = 0.1.

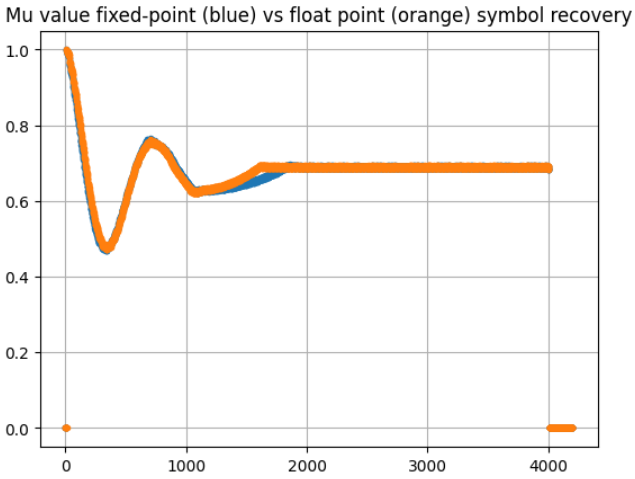

Figura 32. Valor del acumulador de fase (mu) en punto fijo contra punto flotante para Kp = 0.05.

Se buscó que ambas implementaciones converjan a los mismos valores de selección de fase, lo cual fue logrado ya que en ambos casos para el mismo tipo de estímulo convergen a la misma selección de fase, siendo sus diferencias el error de conversión a punto fijo.

IV. Detalle de implementación en FPGA

En esta sección se revisa en detalle la implementación de los distintos módulos que componen el sistema y su implementación en verilog.

5.1. PRBS

Para la generación de símbolos se diseño una PRBS de longitud 5, la cual consta de un registro de desplazamiento de 5 etapas que recorre todos los posibles estados (excepto todos ceros) antes de repetirse. Al ser de longitud 5 la misma posee 5 registros, los cuales permiten obtener una secuencia de $2^5 - 1 = 31$ valores distintos. Para lograr alcanzar esta longitud máxima de secuencia se debe utilizar un polinomio primitivo, que en el caso de nuestra aplicación es el siguiente:

$$x^5 + x^3 + 1$$

Esto indica que el valor "nuevo" que se genera como salida de la PRBS es el resultado de una operación XOR entre los bits 5 y 3, y en cada ciclo se desplaza el registro hacia la izquierda generando un valor nuevo en la posición [0]. Esta generación posee algunas características interesantes como un número de 1s y 0s casi igual (para este caso hay 16 unos y 15 ceros), posee una distribución de frecuencias plana (por lo que son buenos para pruebas de comunicación y su autocorrelación fuera del pico principal es baja. El módulo debe inicializarse con una semilla para poder comenzar a generar valores, la cual tiene que ser distinta de todos ceros, la semilla elegida fue la carga de los registros con "00001". La arquitectura de la PRBS5 se puede observar en la figura 33.

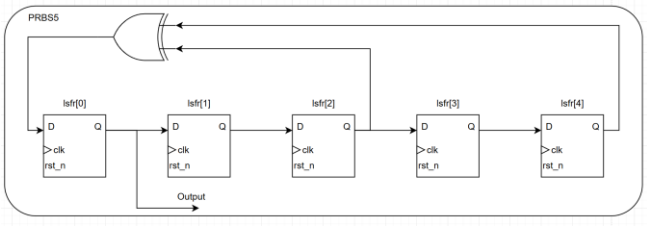

Figura 33. Arquitectura de PRBS5.

Este modulo genera 1s y 0s, los cuales a través de un multiplexor en el top level seleccionan entre dos valores correspondientes a +1 y -1 en formato S(8,7).

5.2. Generación y estructura del filtro polifásico transmisor y receptor

El filtro transmisor diseñado para este sistema realiza un upsampling de 64 sobre una muestra por símbolo generada por la PRBS, por lo que en el diseño requerimos un filtro de coseno realzado de 64 muestras por símbolo, para que al ser diezmado en razón 32-a-1 nos presente un set de muestras con delay de tiempo operando a 2 muestras por símbolo. De igual modo, el filtro receptor trabaja sobre estas 2 muestras por símbolo con un diseño de filtro de coseno realzado de 16 muestras por símbolo, obteniendo finalmente 32 muestras por símbolo que corresponden a las fases seleccionables por el controlador (el filtro posee solo 16 fases pero además se posee un flag de selección de muestra que es lo que nos otorga las 32 opciones, que se explicará luego). Para el diseño de los filtros polifásicos se calcularon los coeficientes en Matlab para obtener un filtro de coseno realzado de valor Alpha = 0.5, group delay de 12 y un oversampling de 64 para el filtro transmisor y de 16 para el filtro receptor.

```
hh = rcosdesign(0.5, 12, 64, 'normal');
gg = rcosdesign(0.5, 12, 16, 'normal');
```

Acto seguido debemos normalizar los valores máximos para que su valor máximo sea 1.

```
gg = gg / max(gg);
hh = hh / max(hh);
```

Y finalmente descartamos las muestras de los bordes para que nos quede de una longitud que sea múltiplo de las fases que poseen (32 y 16) y los reordenamos en filas y columnas para formar los sets de coeficientes de cada fase

```
hh2 = reshape(hh(1:768),64,12);
gg2 = reshape(gg(1:192),16,12);
```

Una implementación de fuerza bruta del filtro polifásico requiere la operación de M (siendo M su cantidad de fases) filtros que actúan en paralelo donde el valor de entrada se va presentando en cada instante de tiempo como entrada a cada

uno de estos filtros. La interacción de las líneas de delay en cada camino de fase con el set de cambios sincrónicos puede ser comparado a un conmutador de entrada que entrega muestras sucesivas a las distintas fases del filtro de M-fases, como ilustra la figura 34.

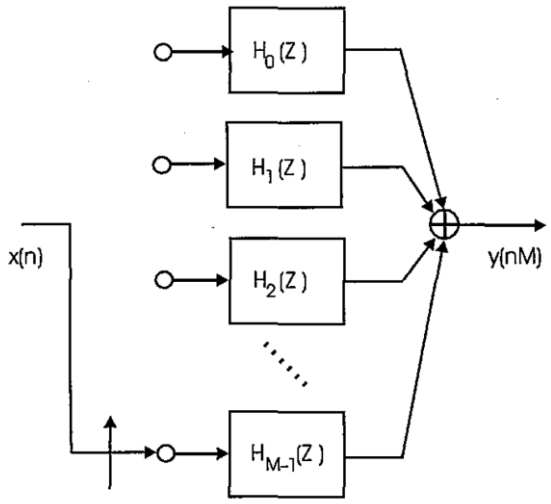

Figura 34. Filtro polifásico visto como conmutador de entrada hacia distintos filtros correspondientes a cada fase.

En la realidad y para hacer más eficiente la utilización de recursos podemos construir un único filtro en el cual tenemos M sets de coeficientes que se seleccionan en cada instante para realizar la interpolación. De esta manera podemos utilizar un contador accionado por un reloj rápido, que en un tiempo de símbolo recorre todas las fases del filtro, generando el sobremuestreo buscado. La estructura para esta arquitectura del filtro polifásico implementado es la mostrada por la figura 35, donde los sets de coeficientes son seleccionados a través de un multiplexor por un contador y se utiliza una sola línea de delay de muestra de entrada de longitud 12 (la cantidad de taps del filtro). Esto corresponde a una aplicación de hardware folding para poder reutilizar la estructura del filtro con los distintos sets de coeficientes.

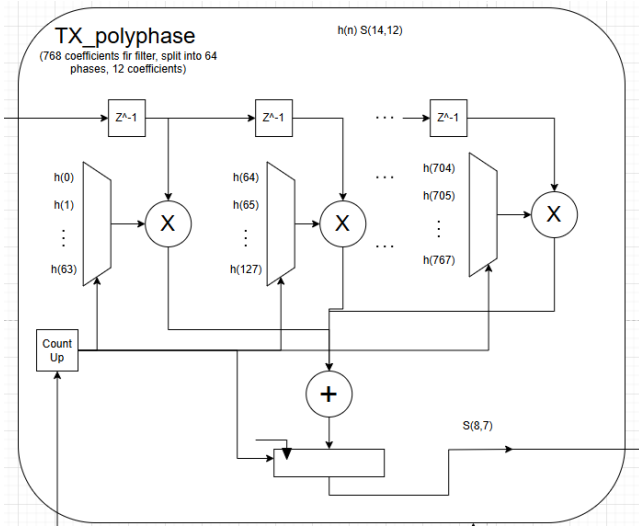

Figura 35. Arquitectura de mínimum storage para filtro polifásico.

Sobre este diseño luego se tuvo que aplicar un registro sobre los coeficientes seleccionados, un registro sobre el contador de entrada para tener un valor estabilizado del mismo y registros de pipeline sobre el árbol de sumas de salida para acortar caminos críticos (suma de 12 valores distintos) en el diseño, quedando finalmente así:

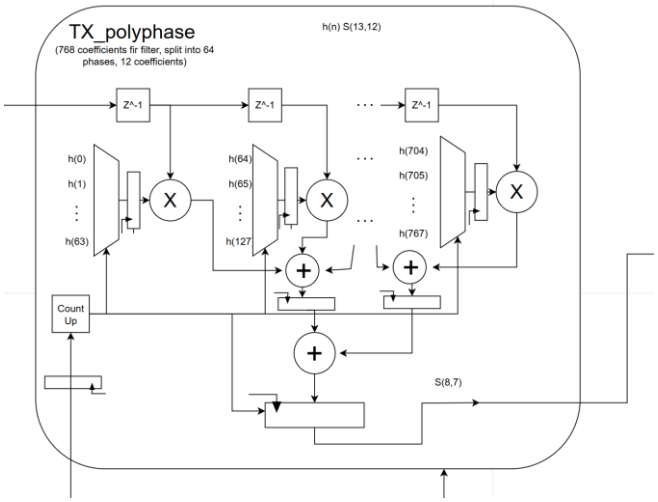

Figura 36. Arquitectura de mínimum storage para filtro polifásico, con registros de pipeline utilizados.

La misma arquitectura se aplicó para el filtro receptor pero utilizando solo 16 sets de coeficientes. Tanto los registros de el valor del contador, como el registro de los coeficientes y de las sumas parciales del árbol de suma introducen latencia en la salida del filtro, la cual debió ser cuidadosamente tenida en cuenta en la parte del controlador para asegurarnos que al querer seleccionar una determinada fase actual dada por el set de coeficientes señalados por el contador, el valor final que se seleccionara correspondiera a dicha fase.

Se analizó como opción o posible implementación alternativa realizar la estructura transpuesta de este filtro. La misma tiene la siguiente forma:

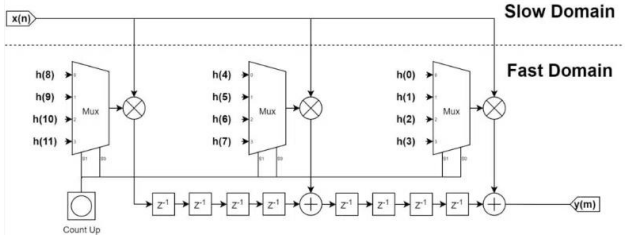

Figura 37. Arquitectura transpuesta del filtro polifásico de mínimum storage.

En este caso los registros de delay pasan a después de la multiplicación y en vez de actualizarse a cada reloj de símbolo se actualizan con la misma frecuencia que el contador. En este caso en vez de 12 registros (uno por cada tap) se tendrían 768 registros (uno por cada coeficiente), por lo que el área de utilización es mayor pero el camino crítico es mucho más corto y se podrían alcanzar velocidades de implementación más altas que no fueron necesarias para este caso.

También se investigó con respecto a las estructuras óptimas de filtros polifásicos propuestos por el fabricante. En el documento Virtex-5 FPGA XtremeDSP Design Considerations UG193 de Xilinx se propone una estructura de filtro optimizada para la arquitectura de Xilinx, lo cual nos otorgaría el mejor rendimiento en timing y área. A pesar que el documento es aplicable a Virtex 5, también lo es para las

demás FPGA ya que su arquitectura DSP es muy similar. Dicha estructura se encuentra presente en la figura 38.

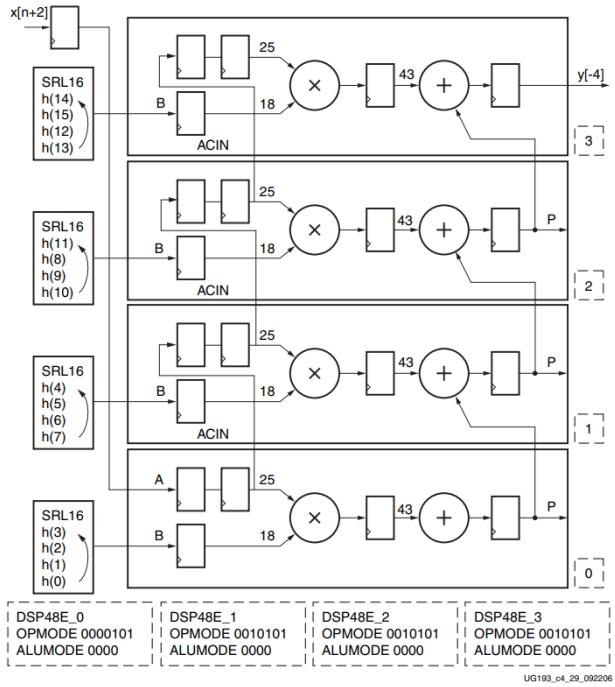

Figura 38. Estructura óptima propuesta en documentación de Xilinx para filtros polifásicos.

En este caso no se exploró la implementación de esta alternativa, aunque si se encontraron propuestas valiosas en la misma para utilizar, como el uso de pipelining en el árbol de sumas o el registro de los valores de los sets de coeficientes previo a su multiplicación.

5.3. TED de máxima verosimilitud

Para el caso del timing error detector se eligió una implementación de máxima verosimilitud a partir de un paper de fred Joel harris que se utiliza mucho para este tipo de implementaciones ya que para su cálculo utiliza el valor de la fase seleccionada, la fase anterior y la posterior. La misma se puede observar en la figura 39.

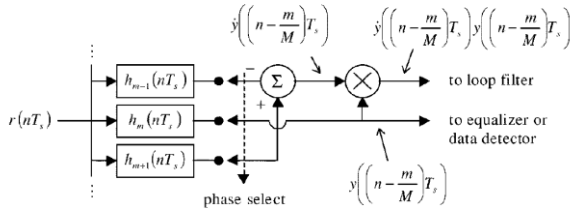

Figura 39. Implementación de TED con máxima similitud.

Esto se podría lograr con la utilización de tres filtros, pero en este caso como para fines visuales estamos utilizando la reconstrucción completa de la señal original a partir de la muestra, se utilizan dos registros encadenados a la salida del filtro polifásico receptor, de manera que en un determinado momento, tenemos en el segundo registro el valor de la fase seleccionada menos uno $(y_{m-1}(n))$, en el primer registro el valor de la fase seleccionada $(y_m(n))$ y en la salida del filtro el valor de la fase seleccionada mas uno $(y_{m+1}(n))$, por lo que tenemos los tres valores necesarios para calcular el error mencionado anteriormente.

$$e(n) = (y_{m+1}(n) - y_{m-1}(n)) * y_m(n)$$

En el momento en que se tienen estos tres valores se acciona un enable a través del controlador y entran los valores al timing error detector, el cual posee la siguiente arquitectura.

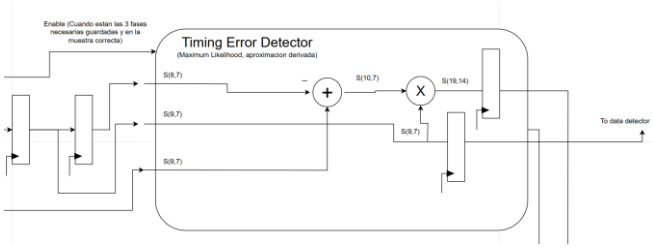

Figura 40. Arquitectura del timing error detector.

Una vez que se recibe el enable, se actualizan los valores de los registros y se envía el enable hace el lazo de filtro para el condicionamiento de la señal de error.

Se evaluó la posibilidad de realizar una implementación con un filtro polifásico que contiene los coeficientes de la derivada del filtro de coseno realzado. A través de esa implementación el sistema utiliza ese valor multiplicando por la salida del filtro y obtiene el término de error. Esta propuesta utiliza como recursos otra implementación de un filtro polifásico pero reduce el trabajo de cómputo del TED, por lo que podría ser válida. La arquitectura para la misma se puede observar en la figura 41.

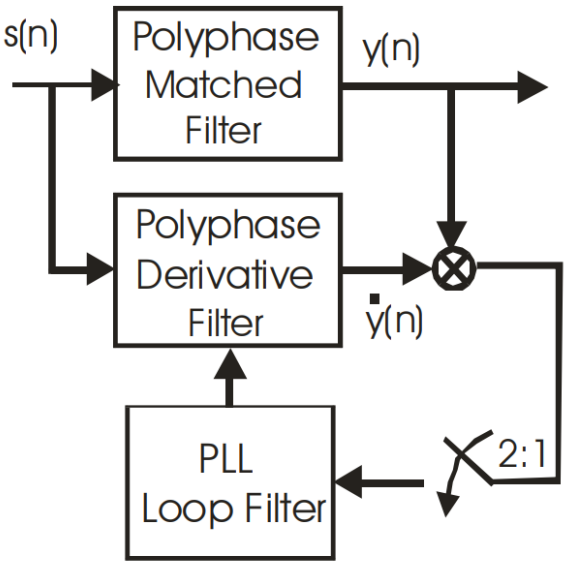

Figura 41. Arquitectura para cálculo de TED utilizando filtro polifásico derivativo.

Para este caso, el error está dado por:

$$e(n) = \dot{y}_m(n) * y_m(n)$$

5.4. Filtro de lazo (Loop Filter)

La implementación el filtro de lazo se realiza para acondicionar la señal de error y poder controlar la respuesta

del sistema. Para este caso se utilizó un control de tipo PI, esto significa que posee un componente proporcional y un componente derivativo, por lo que la salida está dada por

$$error(t) = K_p * e(t) + K_i * \int_0^t e\,(t)$$

La arquitectura de la implementación se puede ver en la siguiente figura.

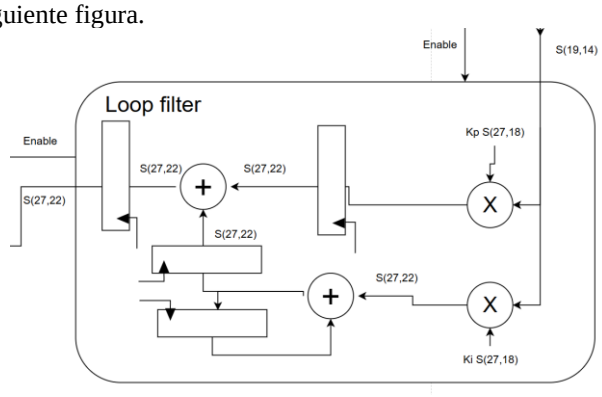

Figura 42. Arquitectura del filtro de lazo.

Los valores seleccionados para los coeficientes fueron $K_p = 0.3$ y $K_i = 0.01$, estos valores fueron testeados experimentalmente y fue una combinación que permitía una respuesta moderadamente rápida del sistema pero no agresiva, y corrección del error en estado estable sin provocar demasiado overshooting, dándole una respuesta no oscilatoria.

5.5. Control de fase

El control de las fases y del sistema se encuentra en el módulo top que instancia al mismo. Desde este módulo se controla todo lo referido al manejo de las muestras y enables del sistema.

El funcionamiento básico se centra en un contador que va de 0 a 63 (sel2) y aumenta en 1 en cada período de clock. Como este contador es de 6 bits, tomando los 4 bits interiores, tenemos un reloj de 0 a 15 (sel) que aumenta en 1 cada dos períodos de clock y que completa su ciclo dos veces dentro del contador a 63.

Cuando el contador está por hacer overflow y empezar en 0, se enciende el flag de prbs_sample_en, el cual muestrea la salida actual de la PRBS por un tiempo de un periodo de reloj y enciende el flag de clk_symbol, el cual le avisa al módulo PRBS que puede generar un nuevo valor (a ser leído en el próximo periodo de símbolo) y le avisa al filtro polifásico transmisor que se está presentando una nueva muestra proveniente de la PRBS.

Una vez presentada la entrada con el valor por un período de reloj al filtro transmisor, la entrada es 0 por los siguientes 63 períodos de reloj, ya que el valor se procesa dentro del filtro. En cada período de reloj el contador va aumentando y va seleccionando un set de coeficientes dentro del filtro correspondientes a una fase distinta para lograr la interpolación de la muestra presentada y generar una señal.

Esta señal es muestreada por un ciclo de reloj dependiendo del valor de la entrada de control de fase (variable de 0 a 31). De esta manera, de la señal con sobremuestreo de 64, se realiza un diezmado para obtener dos muestras por símbolo que se encuentran con un desfase temporal definido por la entrada.

Estas dos muestras son presentadas al filtro polifásico receptor en los momentos donde el contador sel está por llegar a 0, coincidiendo con el clk_sample. Esto se debe en ambos casos a que la muestra entrante debe primero ser multiplicada por la fase 0 de los coeficientes para la correcta reconstrucción de la señal, por eso son tan críticas las cuestiones de alineación de tiempo en el procesamiento de las muestras en el sistema.

Cuando *sel* es equivalente (considerando a las latencias del sistema) al valor indicado por el puntero de fase, se activa un enable que selecciona la salida actual del filtro para ser procesada por el TED y filtro de lazo. Este valor se actualiza una vez por símbolo en el acumulador de fase que es un registro del cual se toman los 4 bits (para seleccionar entre las 16 fases del filtro receptor) más significativos para asignarlos al puntero de fase que nos indica cuál es la fase donde se encuentra el pico máximo de la señal. El detalle de esto último se comentará luego de una breve explicación.

Selección de muestra: Al tener dos muestras de la señal modulada por intervalo de símbolo la incógnita aparece en si el valor pico se encuentra en la primer o segunda muestra. Como a priori desconocemos la respuesta, se analizará que sucede en ambos casos, presentados en la figura 43.

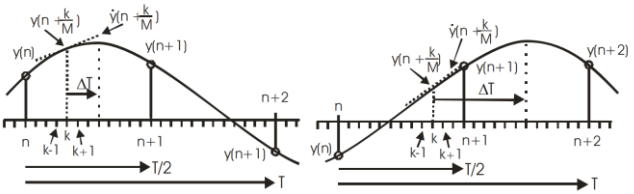

Figura 43. Dos condiciones:

Lado izquierdo, el pico de la señal en el primer intervalo y

Lado derecho, el pico de la señal en el segundo intervalo

En el caso del sistema propuesto, tenemos un filtro polifásico con 16 sets de coeficientes (fases 0 a 15) que nos permiten particionar el intervalo de reloj de T/2 en 16 segmentos. Asumiendo que el caso de la izquierda es correcto y que comenzamos tomando esa muestra. Seteamos el flag de odd/even en “0” indicando que estaremos leyendo las muestras impares y colocamos el puntero de fase en algún valor precargado, como podría ser “8” que es el centro del intervalo mencionado. El TED junto al filtro de lazo incrementan el valor del puntero de fase hacia índices más altos, hacia la derecha, en respuesta a que el producto $e(n) = (y_{m+1}(n) - y_{m-1}(n)) * y_m(n)$ es positivo y aumenta el contenido del registro acumulador de donde se define el puntero de fase. Eventualmente se alcanza un punto donde el producto se vuelve cero (pico, donde $y_{m+1}(n) - y_{m-1}(n)$ son iguales, entonces el producto es cero) y el acumulador (y consecuentemente, puntero) se establecen en un valor estable.

Si ahora consideramos el caso de la derecha de la figura, pero nuevamente suponemos que el pico está en la primera mitad del intervalo con el flag de odd/even en “0” el puntero en “8”, el TED y filtro de lazo comenzarán a aumentar el acumulador y puntero a valores mayores igual que en el caso anterior, hasta llegar al índice “15” donde aún todavía buscará pasar al próximo índice “16”. Aunque no existe un índice “16”, si

existe un próximo índice, el índice "16" corresponde al índice "0" del siguiente intervalo, correspondiente a la siguiente muestra. Cuando el acumulador de fase realiza un overflow en respuesta a un incremento del error proveniente del lazo de error, debemos activar el flag de odd/even a "1" y utilizar las muestras pares en vez de las impares. Y de igual manera se produce ante el underflow por un decrecimiento del valor del acumulador ocasionado por un valor negativo del lazo de error estando en las muestras pares hacia las impares y las veces que sea necesario.

Para realizar el control de que el proceso de actualización del registro acumulador de fase se realice en el momento correcto y con la muestra correcta se lleva un control de lo que corresponde a la muestra par/impar en base a si nos encontramos en la primera o segunda mitad del contador a 63 que controla el filtro transmisor, a través del flag sample_sel que se pone en alto en la primer mitad del ciclo si odd_even está en alto o en la segunda mitad si está en bajo. Debido a la latencia introducida por los registros de la salida del árbol de sumas y de registro de el valor de contador en el filtro polifásico de transmisión, se retarda el flag sample_sel para que el valor de salida del filtro corresponda al valor del contador de selección de coeficientes de fase correcto en la selección correcta de muestra.

Este flag sample_sel_aligned es utilizado para que en el momento en que el valor del contador de 0 a 15 sea igual al índice de fase tomado a partir del acumulador de fase, se active el flag enable_ted que realiza la captura de los valores de los registros de desplazamiento de salida del filtro polifásico receptor hacia el módulo de TED, donde se computa la salida y una vez lista se presenta junto al enable del filtro de lazo a dicho módulo. En este módulo nuevamente se procesa el valor obtenido en el TED y una vez listo, se presenta al acumulador de fase junto a un enable para que se realice la actualización del valor del mismo, lo cual se produce una vez por ciclo.

En la figura 44 se puede visualizar la arquitectura para el controlador del sistema.

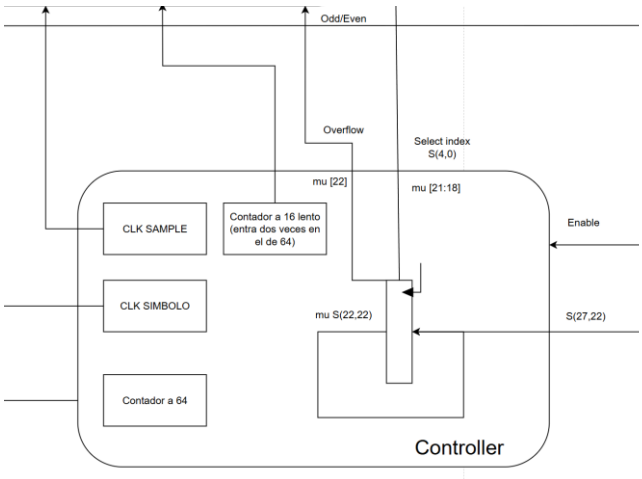

Figura 44. Arquitectura del controlador del sistema.

Finalmente, en la figura 45 se encuentra la arquitectura del sistema completo.

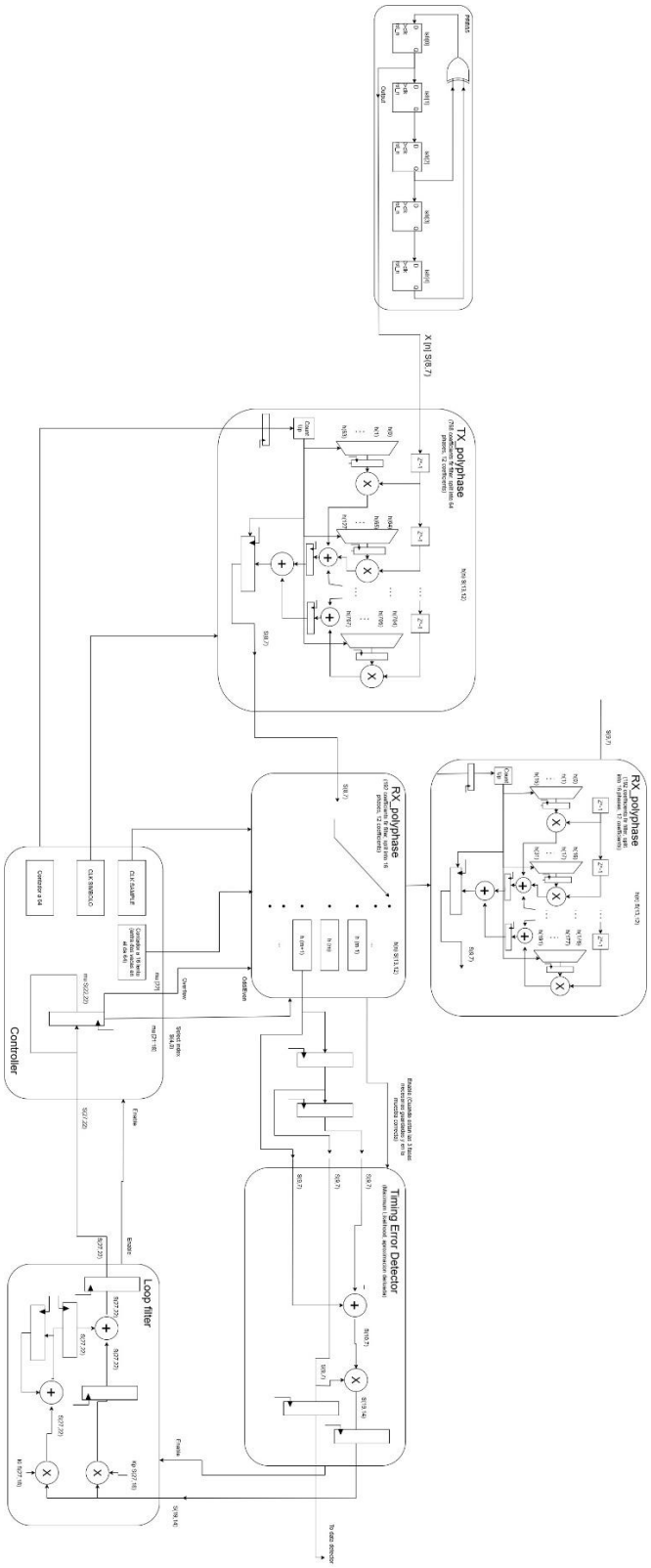

Figura 45. Arquitectura del sistema completo.

V. RESULTADOS Y VERIFICACIÓN

Para verificación de resultados del diseño se utilizó el software Vivado para la implementación del diseño en una FPGA AMD Artix 7 35T. Una vez descripto el diseño en RTL se hicieron pruebas a través de un testbench donde el estímulo consistió en seleccionar una fase como entrada y ver cómo respondía el sistema. La primera simulación del RTL se puede observar en la figura 45.

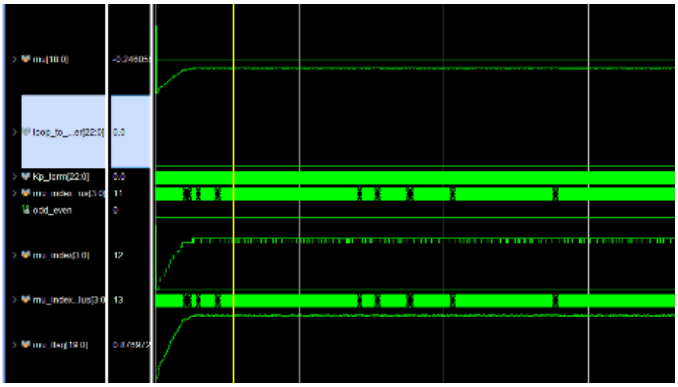

Figura 45. Simulación en RTL hacia un desfase fijo.

Como se puede observar el comportamiento es el esperado, donde el valor del error produce un incremento en el acumulador de fase hasta establecerse en un valor estable donde la salida es óptima.

El paso siguiente fue modificar los estímulos para que consistan en variar la entrada de fase para variar la muestra que se seleccionaba del filtro transmisor y evaluar la recuperación lograda por el sistema. Los resultados se pueden observar en la figura 46 y en la figura 47 un acercamiento hacia el comportamiento en los casos donde debe conmutar el enable de odd/even para selección de la primer fase de la siguiente muestra.

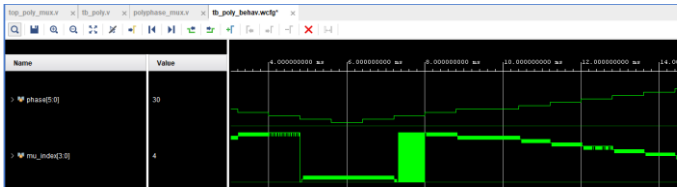

Figura 46. Simulación en RTL con desfase variable.

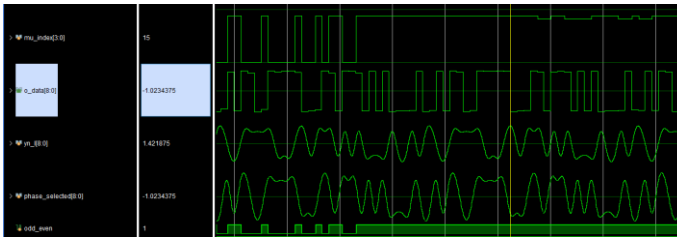

Figura 47. Detalle de comportamiento flag odd/even.

Se puede observar tanto la correcta recuperación a cada paso del desfase introducido, manteniendo siempre la salida en los valores óptimos, e incluso verificando el funcionamiento del flag de odd/even para la selección de muestra.

Para la implementación del diseño en FPGA, se instanciaron dentro del top del módulo los módulos de VIO (Virtual Input Output) e ILA (Integrated Logic Analyzer). El módulo VIO se generó para poder controlar las señales del control de fase, reset, y enable. El módulo ILA se generó para poder visualizar las señales correspondientes a la salida de datos de todo el sistema, la salida del filtro polifásico transmisor, la salida del filtro polifásico receptor y el valor del puntero de fase (mu_index). El resultado de la implementación fue el siguiente:

ILA

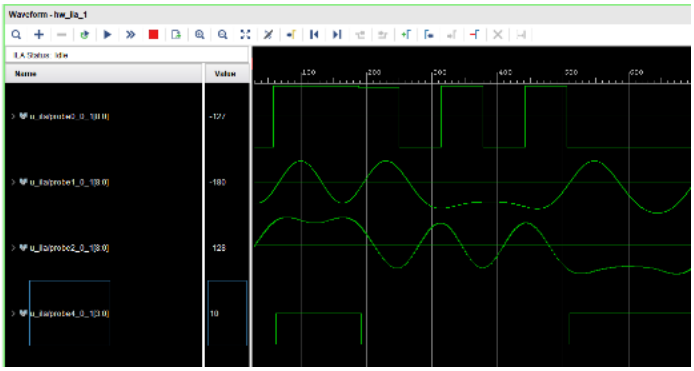

Figura 48. Visualización de señales de FPGA con ILA.

Se exportaron estas señales a un archivo de .csv para poder realizar la correlación de vectores contra la simulación en punto fijo realizada en Python. El archivo se exportó bajo el nombre “iladata_final.csv”

Finalmente en la simulación de Python se leyeron estos datos desde el archivo .csv y se corroboraron con los datos obtenidos en la ejecución de la simulación en punto fijo para las mismas condiciones, obteniendo una igualdad de resultados. En la simulación Python se pueden ver estas igualdades en las variables vector_m, vector_m2, vector_m3 y vector_m4.

La implementación se realizó a una velocidad de 100 MHZ. En cuanto a utilización de recursos, los recursos utilizados en la implementación sin VIO e ILA fueron los siguientes:

Utilization — Post-Synthesis | **Post-Implementation** — Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	367	20800	1.76
FF	353	41600	0.85
DSP	28	90	31.11
IO	17	210	8.10
BUFG	1	32	3.13

Figura 49. Recursos utilizados en implementación sin VIO e ILA.

Para este caso, el worst negative slack (WNS) fue de 0.378ns. El histograma de timing se encuentra en la figura 50.

Figura 50. Histograma de timing para implementación sin VIO e ILA.

Donde el camino crítico está dado por el esquemático de la figura 51.

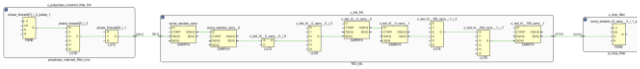

Figura 51. Esquemático de camino crítico.

Para la implementación con VIO e ILA, los datos fueron los siguientes:

Utilization — Post-Synthesis | Post-Implementation

Graph | Table

Resource	Utilization	Available	Utilization %
LUT	1827	20800	8.78
LUTRAM	150	9600	1.56
FF	2850	41600	6.85
BRAM	1.50	50	3.00
DSP	28	90	31.11
IO	1	210	0.48
BUFG	2	32	6.25

Figura 52. Recursos utilizados en implementación con VIO e ILA.

Para este caso, el worst negative slack (WNS) fue de 0.201ns. El histograma de timing se encuentra en la figura 53.

Figura 53. Histograma de timing para implementación con VIO e ILA.

Donde el camino crítico está dado por el esquemático de la figura 54.

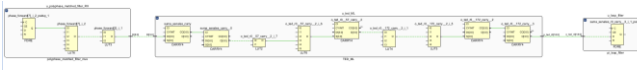

Figura 54. Esquemático de camino crítico.

Para este diseño se hizo una prueba de la velocidad máxima de funcionamiento alcanzable y la misma fue de 107.5 MHZ, como muestra la figura 55.

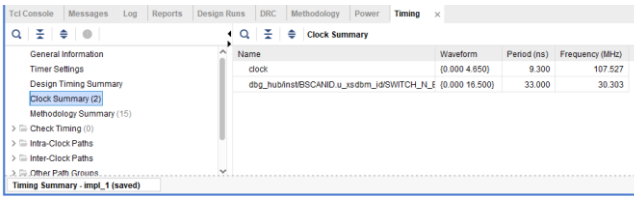

Figura 55. Máxima frecuencia de funcionamiento a la cual se puede implementar el sistema.

VI. Conclusion y trabajo futuro

Se logró la implementación de un sistema de recuperación de reloj junto a sus generadores de estímulos dentro de una FPGA para un sistema de transmisión de datos BPSK (canal simple).

Los filtros polifásicos son los elementos más demandantes aritméticamente en la implementación, y son los principales en cuanto a consumo de recursos y área, por lo que son el foco principal si se quisieran implementar mejoras, teniendo muchas posibilidades disponibles, tanto por el lado de las implementaciones con filtros transpuestos, implementaciones con lectura de los coeficientes desde memorias ROM, las implementaciones óptimas propuestas por el fabricante hasta las posibles implementaciones con elementos MAC (multiply and accumulate). La búsqueda del diseño implementado fue minimizer área ocupada en la FPGA, y como consecuencia, la multiplexación por división de tiempo (hardware folding) es un tema significante en la implementación.

Se denota que la complejidad del circuito de recuperación de reloj está asociada con los elementos de control y el manejo de los datos a través de los distintos módulos que componen el sistema.

Referencias

[1] Multirate digital filters for symbol timing synchronization in software defined radios – fred joel harris, Michael Rice https://www.researchgate.net/publication/3234710_Multirate_digital_filters_for_symbol_timing_synchronization_in_software_defined_radios?enrichId=rgreq-3e03fbe9ff2a9f93d053ef15f291f353-XXX&enrichSource=Y292ZXJQYWdlOzMyMzQ3MTA7QVM6OTkxMjYxODAwNTcwOTZAMTQwMDY0NDkzMTU3Nw%3D%3D&el=1_x_2&_esc=publicationCoverPdf

[2] Architecture and Simulation of Timing Synchronization Circuits for the FPGA Implementation of Narrowband – Chris Dick, Benjamin Egg, fred joel harris https://www.wirelessinnovation.org/assets/Proceedings/2006/sdr06-1.2-2-dick.pdf

[3] Multirate Signal Processing for Communication Sytems – fred joel harris

[4] Virtex-5 FPGA XtremeDSP Design Considerations UG193 https://docs.amd.com/v/u/en-US/ug193

[5] Part 5: Polyphase FIR filters https://vhdlwhiz.com/part-5-polyphase-fir-filters/

[6] Polyphase filter / Farrows interpolation https://dsprelated.com/showarticle/22.php

[7] Strategies for pipelining logic https://zipcpu.com/blog/2017/08/14/strategies-for-pipelining.html

[8] Fxp math Python library https://github.com/francof2a/fxpmath