Universidad Nacional del Sur

Proyecto Final de Carrera

Implementación de un conversor tiempo a digital en FPGA.

Filsinger Miqueas

Contents

Introducción

Un TDC (Time to Digital Converter) es un circuito electrónico capaz de medir un intervalo de tiempo con alta precisión y convertirlo en un valor digital. Fueron impulsados principalmente en el campo de la física atómica y de altas energías [1], radares tipo LiDAR [2], y aplicaciones biomédicas como tomografías por emisión de positrones (PET). Estas distintas aplicaciones normalmente utilizan ASICs off-the-shelf para implementarlo; sin embargo los recientes avances tecnológicos en FPGA han permitido realizar diseños que alcanzan una precisión temporal del orden de los pocos picosegundos, contando con la ventaja de ser flexibles y la posibilidad de integrar la aplicación en un único dispositivo.

Particularmente, en [3] se diseñó y fabricó un chip que implementa cadenas de retardo programables, donde el elemento de retardo se basa en un inversor CMOS que mediante un registro de configuración puede añadir capacidades a su salida (SCI) o modificar su resistencia equivalente (CSI), como muestra la Figura 1. Esta configuración permite la posterior ecualización de la cadena de retardos, celda a celda, siendo el retardo promedio configurable de $\tau' = 65$ps. En este trabajo se explora la posibilidad de replicar y mejorar dicha precisión dentro de una FPGA, con el fin de poder obtener idealmente varias muestras en el orden de las decenas de picosegundos, de modo que se logre una ecualización aproximada de la cadena de retardos.

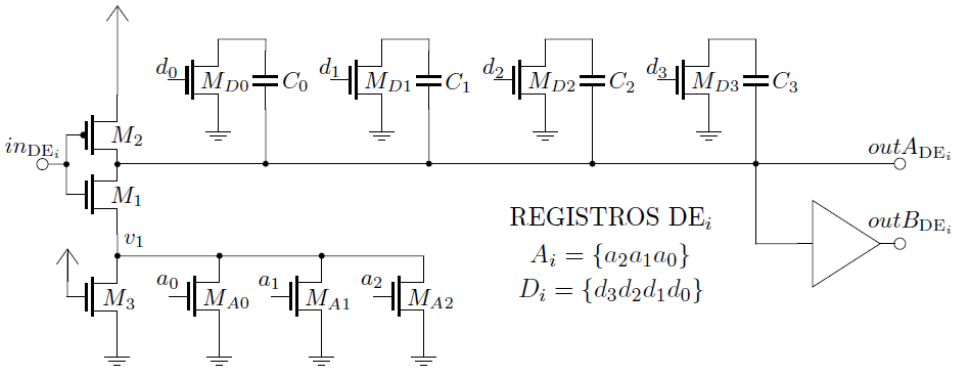


Figure 1: Esquemático circuital del elemento de retardo.

El enfoque se centra en maximizar la precisión y la linealidad, dos de los principales criterios de rendimiento de un TDC. La precisión se refiere a la mínima unidad que el instrumento es capaz de medir, mientras que el concepto de linealidad refiere a la capacidad de mantener la respuesta entrada-salida proporcional en todo el rango de trabajo. Son interesantes además algunos otros aspectos como el rango de medición, el tiempo muerto, su consumo de energía, y la frecuencia de muestreo, sin embargo se omitieron en este trabajo.

Estado del arte

En esta sección se explicarán algunos de los últimos avances propuestos en la bibliografía. Hasta ahora se ha hablado de aquellos aspectos esperables en un TDC, pero es importante tener en cuenta que existen distintos paradigmas en su diseño dependiendo de su destino: ASIC o FPGA. La libertad en la colocación de componentes (placement & routing) con en el diseño de ASICs permite minimizar retardos indeseados, algo que resulta desafiante en el diseño de FPGAs. Por esta razón, se basa en gran medida en [4] ya que propone un estudio, clasificación y análisis de las distintas arquitecturas específicamente orientadas a FPGA, algo que no se había hecho hasta el momento.

Machado propone clasificar las distintas arquitecturas según los componentes que utiliza, de esta forma propone la taxonomía que se muestra en la Figura 2.

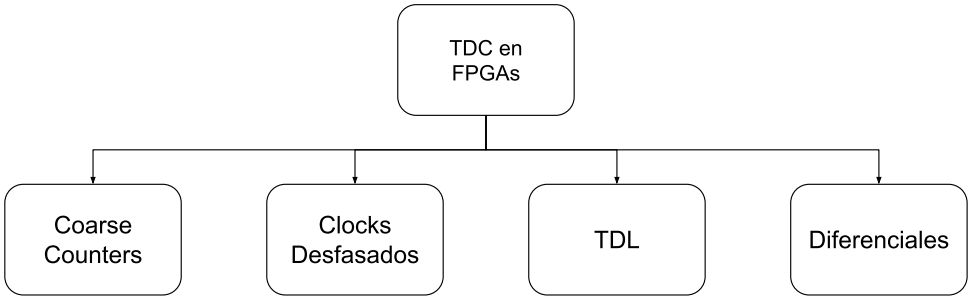


Figure 2: Clasificación de las distintas arquitecturas.

Las arquitecturas de tipo *Coarse Counter* (contador grueso), son aquellas que utilizan un simple contador para estimar el ancho de pulso de una señal, según la cantidad de clocks del sistema que entran en ella; su precisión es entonces el periodo de reloj. Las arquitecturas basadas en *Clocks Desfasados* son todas aquellas que utilizan más de un clock para estimar la medición: algunas de ellas deciden la medición a partir de saber cuál clock puede observar primero la llegada del pulso, mientras que otros utilizan elementos adicionales para la interpolación. Por otro lado está tal vez una de las arquitecturas más utilizadas, aplicada en este trabajo, y es aquella que utiliza *Tapped Delay Lines* (TDL: lineas de retardo registradas) para realizar la interpolación y generar la medición fina. Esto es, se ingresa el pulso de entrada en una cadena de elementos que introducen un retardo uniforme; al muestrear el estado de la cadena entonces se infiere cuántos retardos la señal cruzó, con lo que se estima una medición. Dentro de esta clasificación existen distintas formas de utilizar esta forma de interpolación, algunas arquitecturas extienden el rango de medición al combinar también un Coarse Counter (la topología Nutt) otras utilizan varias cadenas de retardo con el fin de mejorar la precisión realizando un promedio de ellas, y otras combinan más de una cadena y más de un clock. En cualquiera de sus formas, se clasifican según su componente principal de interpolación: la cadena de retardos. Por último, las arquitecturas *Diferenciales* se basan en medir la diferencia de retardo entre dos elementos. En estas existen principalmente dos tipos: cadenas de retardos compuestas de dos elementos de retardo distintos, y osciladores en anillo. Por otro lado si buscamos medir el intervalo temporal entre una señal y el clock entonces llamaremos al TDC *síncrono*, si el intervalo puede comenzar y terminar en cualquier momento entonces lo llamaremos *asíncrono*.

Las arquitecturas TDL pueden ser implementadas de distintas formas, como muestra la Figura 3.

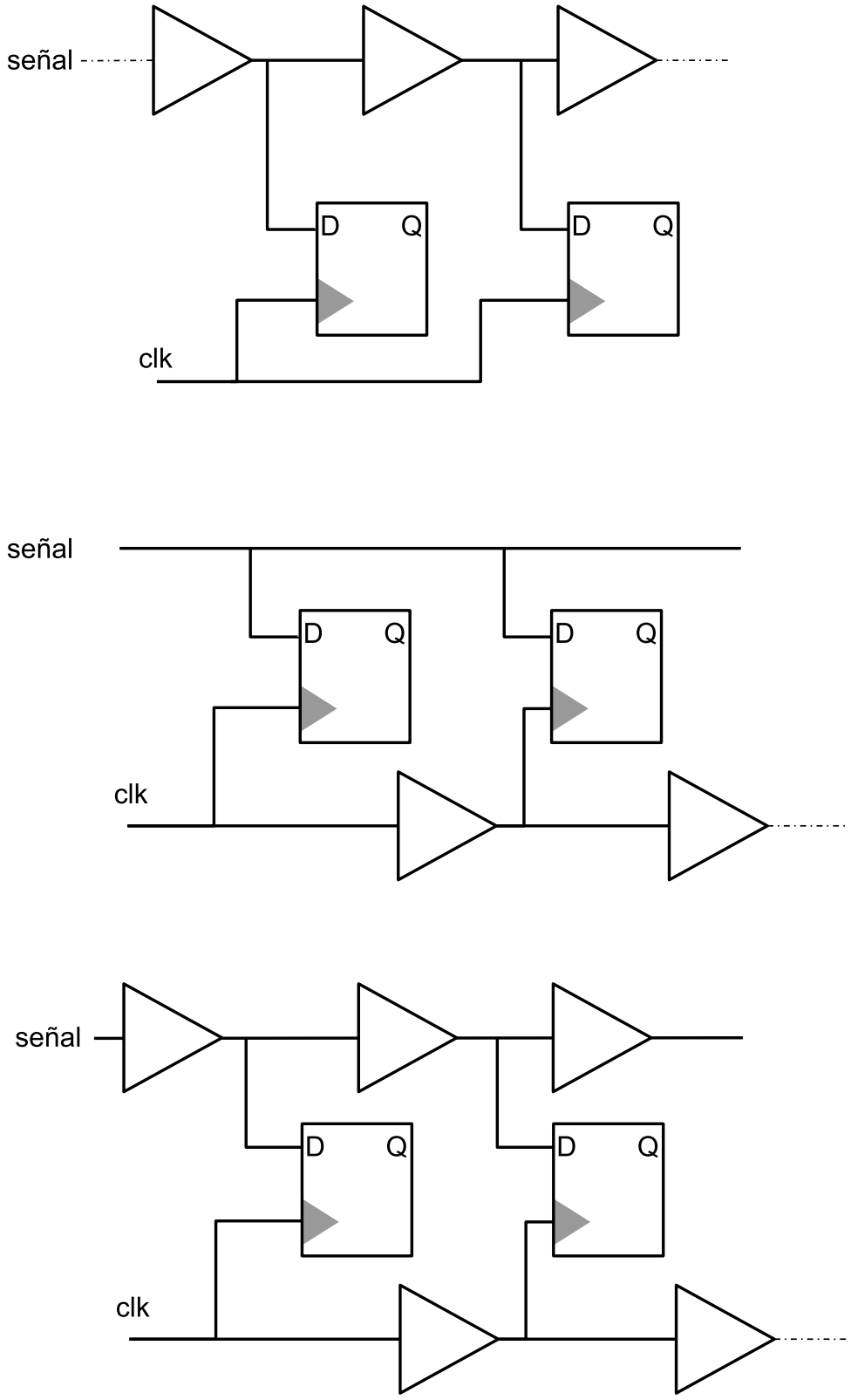


Figure 3: Diferentes formas de aplicar el método Vernier.

Específicamente se utilizó la primer opción, donde la señal alimenta una cadena de retardos, la cuál es registrada en cada clock. Cuando se detecta la entrada de un pulso, se genera una señal

de *Start*, y el valor de la cadena de flip-flops se guarda para luego ser procesado, representando el valor entre lineas rojas en la Figura 4. Cuando la señal egresa de la cadena de retardos, se genera la señal de *Stop*, con la que se vuelve a registrar el valor de la cadena. Finalmente, tal como implementa la topología Nutt, el resultado se compone de una parte gruesa y una parte fina, y puede calcularse como:

$$T_m = N_{coarse}\, T + \tau_{start} - \tau_{stop}$$

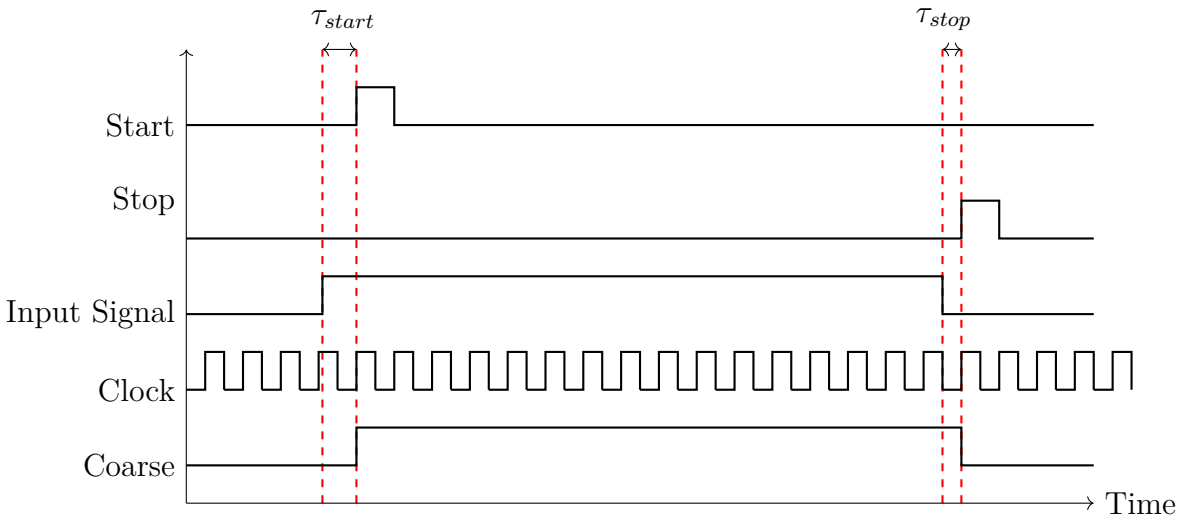


Figure 4: Esquema de medición fina de un Nutt-TDC.

2.2 Tests, mediciones y resultados

Particularmente en la arquitectura TDL, la opción predilecta en la bibliografía para calibrar el instrumento es mediante un "code-density test". Esta prueba consiste en ingresar una señal decorrelacionada del reloj del sistema, de forma que excite todos los elementos de retardo por igual y por lo tanto "golpee" a cada celda, generando un histograma de golpes (hits) en función del número de celda. Si la cadena tiene N_c elementos, mide un periodo de reloj T, se realiza el test con N muestras, y la i-ésima celda captura n_i mediciones entonces es claro que el tamaño de cada retardo es

$$\tau_i = n_i \, \frac{T}{N} \, , \tag{1}$$

y el retardo medio, es decir el retardo de cada celda si es equiprobable obtener un hit en cada bin, es

$$\hat{\tau} = \frac{T}{N_c} \, . \tag{2}$$

La explicación de esto es que a medida que la cadena se acerca a una cadena ideal, la probabilidad de golpear cada elemento debe ser la misma para todos los elementos y es la cantidad de veces que se obtiene un hit sobre la cantidad de experimentos totales, esto es n_i/N. Para una cantidad de mediciones grande, esta probabilidad debe acercarse a $1/N_c$. Al igual que lanzar un dado justo; como es equiprobable obtener cualquiera de sus seis caras entonces la ley de grandes números nos dice que para una cantidad suficientemente grande de experimentos, deberíamos obtener cada cara una misma cantidad de veces, esto es que cada cara salga 1/6 de las veces que se arroja el dado. Entonces, en el TDC debe ocurrir que

$$\frac{n_i}{N} \longrightarrow \frac{1}{N_c} \, ,$$

que es equivalente a que $\tau_i \longrightarrow \hat{\tau}$, a medida que $N \to \infty$. Con esto es posible calcular las no-linealidades de cada bin tomando como referencia el tiempo ideal $\hat{\tau}$. La DNL (Differential Non-Linearity) se puede calcular como

$$DNL_i = \tau_i - \hat{\tau} \ , \tag{3}$$

y con esto la INL (Integral Non-Linearity), es decir las no-linealidades acumuladas a medida que la señal atraviesa la cadena, como

$$INL_i = \sum_{i=0}^{N-1} \tau_i - \hat{\tau} \ . \tag{4}$$

Para obtener la DNL en función del bit menos significativo (LSB), poniendo en evidencia cuántos bits de incertidumbre representa cada no-linealidad, se puede calcular $DNL_i[LSB] = DNL_i[seg]/\hat{\tau}$.
Conociendo estas propiedades es posible generar una "tabla de calibración". Una vez la medición final se realiza en el TDC, se corrige mediante una tabla pre-cargada en memoria, entregando un resultado calibrado al instante de medición (on-line calibration [5]). Se puede contemplar una calibración exclusiva para el flanco *Start* y el flanco *Stop* ([6]), y corregir variaciones en las condiciones PVT ([7]). Con esto, es posible corregir limitaciones de diseño y hardware mediante una buena calibración.

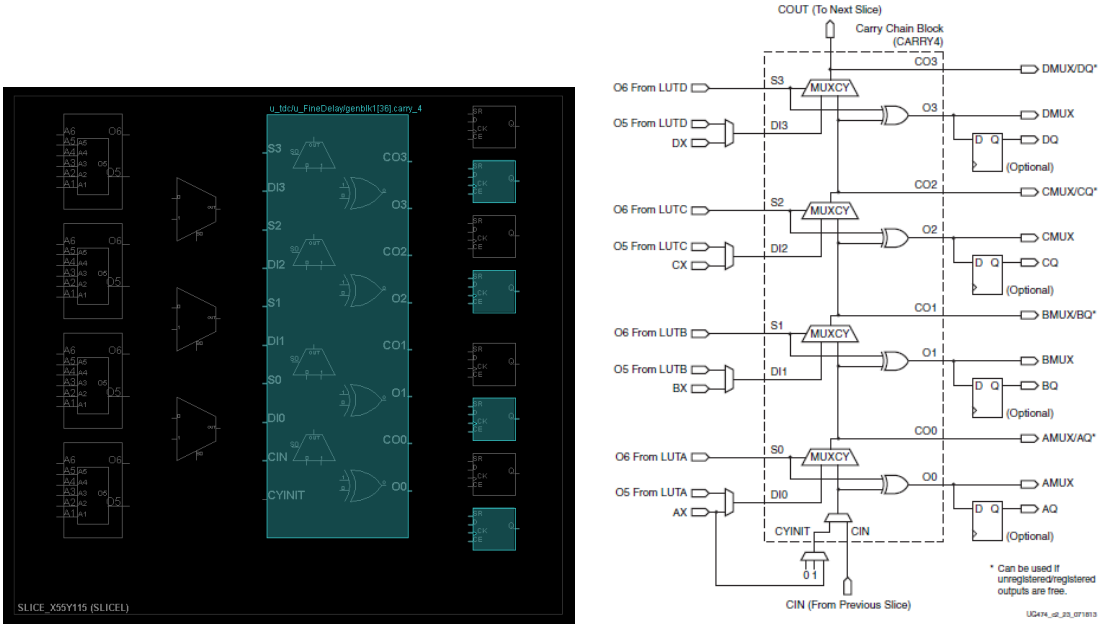
Arquitectura

A este tipo de estructuras digitales nos gusta llamarlas *ad-hoc*, ya que desde su concepción hasta su implementación podremos ver que no se puede enmarcar dentro de una estructura típica, sino que su diseño viene estrechamente ligado a su aplicación en particular. A simple vista puede verse que los elementos que se utilizarán, las etapas de procesamiento, y las técnicas implementadas son poco comunes para lo que es el mundo del diseño digital. Se da a continuación una explicación descriptiva de cada bloque fundamental que conforma la arquitectura del *TDC*.

3.1 Tapped Delay Line

El bloque principal de este diseño, encargado de realizar la *medición fina*, es una cadena de retardos registrados. Para implementar estos retardos existen distintas pero acotadas opciones en una implementación en FPGA. En sus inicios la solución predilecta era utilizar multiplexores en cadena [8], pero para aprovechar las ventajas propias de la estructura de una FPGA se evolucionó a utilizar *Carry4s*, posiblemente utilizado por primera vez en [9]. Estos elementos forman parte de la unidad mínima de trabajo en una FPGA, y su combinación con LUTs, flip-flops, y Multiplexores forman lo que se conoce como un *SLICE*[1]. Los *Carry4s* en su concepción fueron destinados para implementar sumadores, y tienen la ventaja de estar optimizados para tener el mínimo retardo entre compuerta y compuerta. Además cada salida cuenta con un flip-flop routeado inmediatamente a continuación, como se muestra en la Figura 5.

[1]La Artix-7 cuenta con un total de 33650 slices.



(a) SLICE L de una Artix-7 en Vivado 2023.2 (b) Carry 4

Figure 5: Elementos disponibles dentro de una SLICE.

Cada Carry4 cuenta con 4 salidas que pasan por una XOR, (O0-O3) y 4 salidas que pasan por cada MUXCY (CO0-CO3), además cuenta con un puerto *CIN* con el propósito de colocarlas en cadena y una entrada auxiliar *CYINIT* desde donde ingresará el pulso externo.
Como la implementación aquí presentada es asíncrona precisamos capturar la señal en su flanco de entrada y de salida de la cadena de retardos, por lo que luego de la colocación de la primer columna de flip flops (*First* Flip Flops) como se ve en 5, se utilizan otras dos columnas de flip-flops llamados *Start* flip-flops y *Stop* flip-flops. Estos se encargan de registrar la primer columna solamente cuando un flanco de subida o de bajada es detectado, a través de su *Clock Enable*. Así se genera la estructura que se ve a continuación en la Figura 6.

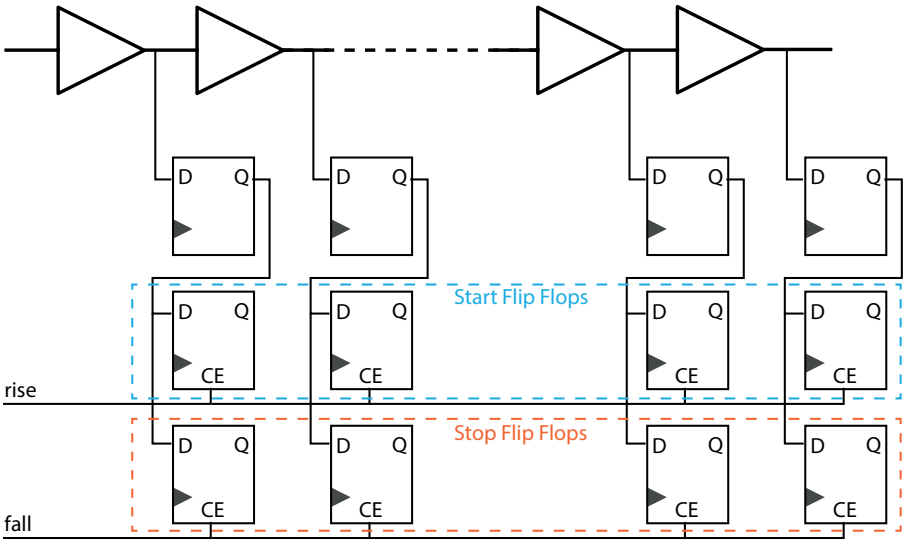


Figure 6: Esquema de la cadena del módulo de interpolación fina.

De esta estructura surgen dos aspectos importantes:

1) Es sabido que la transferencia de información analoga a digital puede producir metastabilidades si no se realiza correctamente. La configuración de doble flip flop para registrar los elementos de retardo soluciona este problema introduciendo lo que se conoce como

"sincronizador".
Esto es, si el tiempo de establecimiento de la señal externa es menor al tiempo de establecimiento requerido por un sólo flip-flop, este puede obtener un valor desconocido a la salida. Sin embargo introduciendo un segundo flip flop en serie, este no sufre de este problema, ya que en el próximo clock la señal es estable, y esta indeterminación se resuelve decantando hacia un estado. Además la técnica de doble registro reduce los problemas de burbujas en el resultado [10].

2) Una propiedad intrínseca del diseño tipo Nutt es que el tiempo del pulso a medir atraviese la cadena en un tiempo no menor al de un clock del sistema. Esta es una condición de diseño y se aprovecha en múltiples bloques, pero en particular aquí se puede ver que el registro de *Start* y *Stop* dependen de un flanco de subida o bajada para generar la señal de *Clock Enable*, el cuál tiene una duración de un clock.

Por otro lado, es importante prestar especial atención al routeo y colocación de cada elemento en el bitstream final, pues es preponderante en la calidad de los resultados del TDC. Si los retardos introducidos por el routeo son uniformes para cada elemento de retardo entonces podemos asegurar que la calibración es capaz de corregir en gran parte estos errores. Según [10] es importante colocar la cadena de retardos en las Slices que quedan a continuación de los Clock Buffers, y que los flip flops sean colocados también cerca de la cadena. Esto fue probado empíricamente y se obtuvieron mejores resultados cuando los extremos de la cadena de retardos quedaban simétricos respecto de los clock buffers.
Experimentalmente se observó que cuando el extremo inferior de la cadena es lejana a los buffers, se pueden observar huecos en el histograma resultante, mientras que al colocarlo simétrico, este espacio se reparte al inicio y al final de la cadena, como se ilustra en la Figura 7.

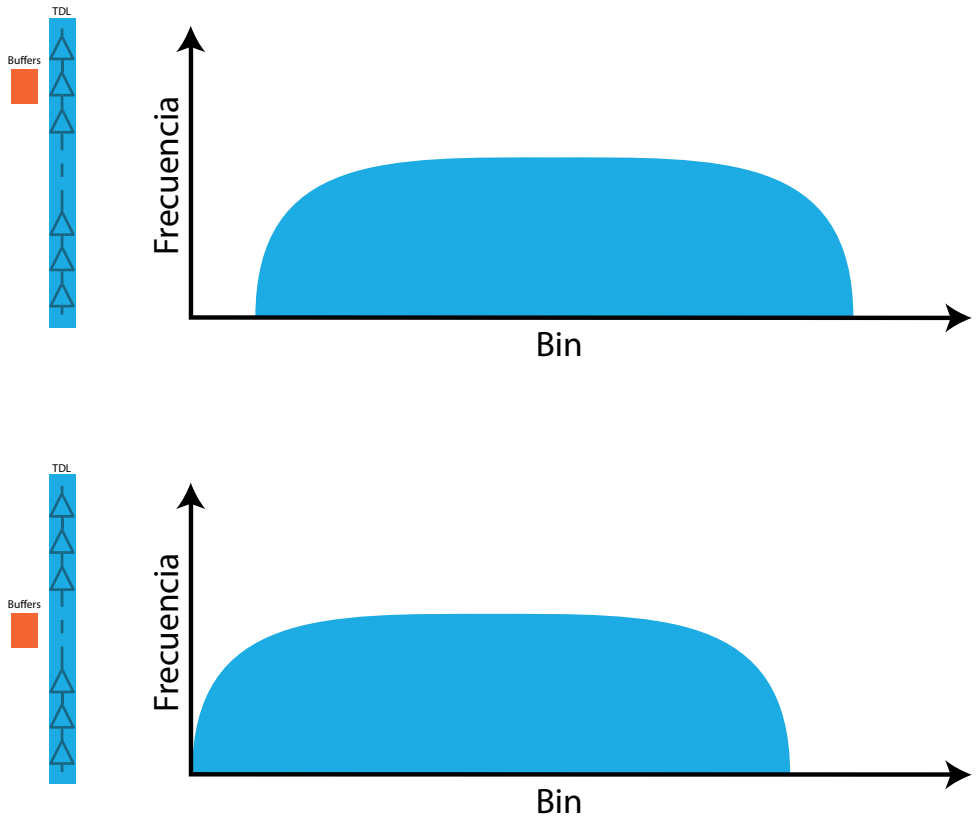


Figure 7: Ejemplo de histograma conseguidos al excitar el TDC con una frecuencia pseudo-aleatoria modificando la posición relativa de la cadena de retardos respecto a los buffers de clock.

La propagación de un pulso por la línea de retardos naturalmente genera un código unario sobre los registros. Esto es, a medida que el flanco descendente progresa sobre la línea de retardos deja a su paso un estado alto en cada Flip-Flop de la cadena, mientra que cuando egresa un flanco ascendente deja ceros a su paso, tal como muestra la Figura 8.

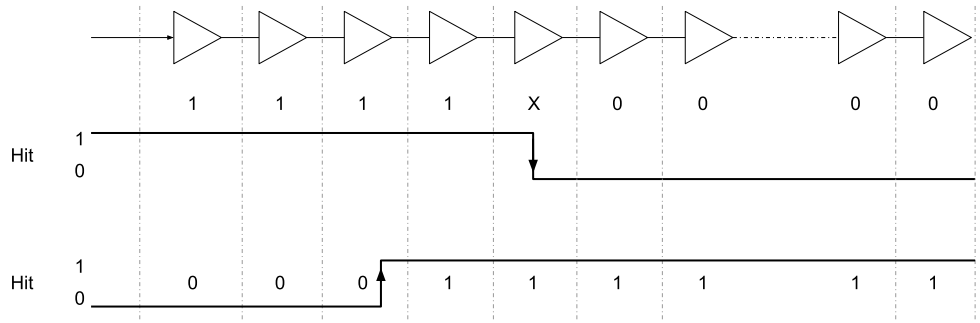


Figure 8: Registro de un pulso digital en el momento que ingresa a la línea, y en el momento que egresa de ella.

La misión principal del decoder es traducir estos valores leídos de los registros *Start* y *Stop* a datos procesables. Aquí se presenta un gran inconveniente presente en todas las soluciones propuestas por bibliografía, pues debido a las variabilidades propias de la arquitectura de una FPGA y de los órdenes de precisión que se pretenden es muy frecuente encontrar burbujas en los registros resultantes, imposibilitando la utilización de decodificadores unarios convencionales. Esto significa que en los registros *Start* o *Stop* se pueden encontrar registros con errores del tipo "11..111001000...", donde no es posible distinguir una transición de estados clara. Según la bibliografía ([4]) esto se debe principalmente al efecto de *clock skew* cada vez más presente en los últimas tecnologías, gracias al decrecimiento abrupto de tamaño de fabricación. Es posible que la diferencia temporal del flanco de clock (el clock skew) entre dos Flip Flops que registran elementos de retardo adyacentes sea mayor al tiempo de retardo de la celda a registrar, y por lo tanto se produce una burbuja en el registro. Esto se ejemplifica en la Figura 9. Si el flip-flop número uno captura la cadena en el momento t_1 y el flip-flop número dos lo hace un momento después en t_2, es posible entonces que el primero decante su estado en un valor 0, mientras que el segundo decante su estado en un valor 1 si $\Delta = t_2 - t_1 \geq \tau$ siendo τ el tiempo de una celda de retardo, y Δ el clock skew.

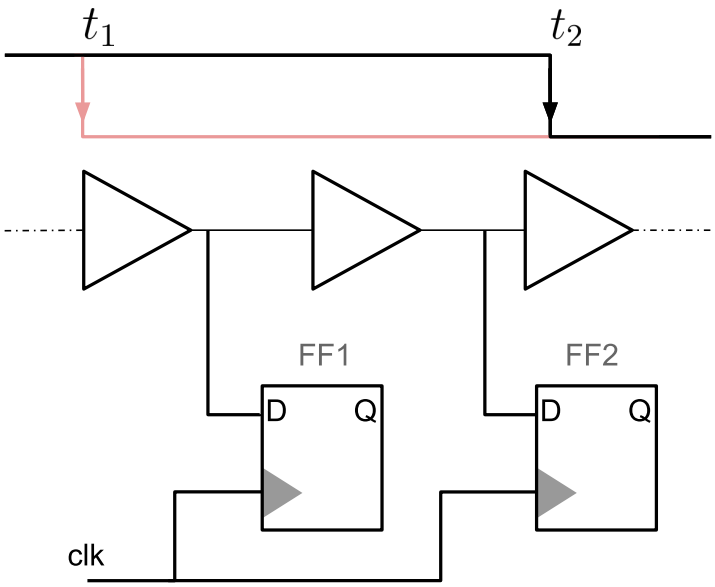


Figure 9: Retardo virtual de propagación que se puede observar cuando existe *clock skew* entre dos flip-flops aledaños.

Para implementar un decodificador se han propuesto distintas soluciones, en este sección resumiremos las ideas más usadas:

1) **Contador de unos (ones counter):** se propuso como una solución en el diseño de ADCs, y luego en [11] se utiliza por primera vez para el diseño de un TDC. La implementación consiste básicamente en contar la cantidad de estados altos en el registro para definir la posición aproximada del flanco. Si se puede asegurar que la cantidad de burbujas es despreciable entonces la respuesta del decodificador se acerca a la real. Esto es beneficioso porque se prescinde del ordenamiento de un registro, es decir la entrada "111100" se decodificaría en la misma respuesta que la entrada "111010", por lo tanto tiene la habilidad natural de corregir errores de burbuja.

2) **Buscador de flancos:** Consiste en realizar un circuito combinacional que busque secuencialmente (look-for) transiciones de estados del tipo "0-1" o "1-0" en el registro de entrada y devuelva la posición de este dentro la cadena. Por ejemplo, para un registro "0001111111" el decoder debe devolver tres, ya que es la posición donde se detecta la transición. Para eliminar el problema de burbujas se recurre a aumentar el tamaño de la secuencia buscada, es decir si se tiene un registro con burbujas del tipo "0001001111" podemos buscar la secuencia "0001" de forma que únicamente la primer transición es detectada y la burbuja se ignora ([12]). Rápidamente se hace evidente que a medida que el tamaño del filtro incrementa es capaz de corregir burbujas de mayor tamaño, a costa de descartar una mayor cantidad de muestras al principio y final de la cadena.

3) **Realineamiento de taps:** Una vez detectados los lugares del registro donde se producen burbujas, se intercambia la posición de salida de estos flip-flops con el de uno cercano ([5]), luego se utiliza un decoder del tipo "buscador de flancos". Para su implementación primero se debe hacer un "code-density test" que nos permita conocer cuáles son los bines de tamaño cero (burbujas), para luego realizar el reordenamiento de taps. Esta implementación depende fuertemente de las condiciones PVT[1] en las que se utiliza la FPGA.

[1]Proceso, voltaje y temperatura.

Una de las ventajas de la implementación tipo Nutt es que nos permite mantener un buen rango de medición sin perder precisión. Esto se logra separando la implementación en dos tareas, la medición gruesa y la medición fina, para esto se utiliza un contador grueso (coarse counter) con el que se mide la cantidad de periodos de clocks que tarda la señal en atravesar la cadena. Aunque esta propuesta parece sencilla precisa de una sincronización previo al procesamiento del resultado, ya que si la señal externa ingresa o egresa aproximadamente en sincronía con el clock del sistema es posible que el contador cuente o no un periodo de reloj menos, cometiendo un error de un periodo completo. En [10] se propone la utilización de más de un contador grueso como árbitros para decidir el resultado final del cálculo, el desarrollo es el siguiente:

Esto consiste en implementar tres contadores gruesos, cada uno alimentado por un clock derivado del clock del sistema y ligeramente desfasado, como se muestra en la Figura 10.

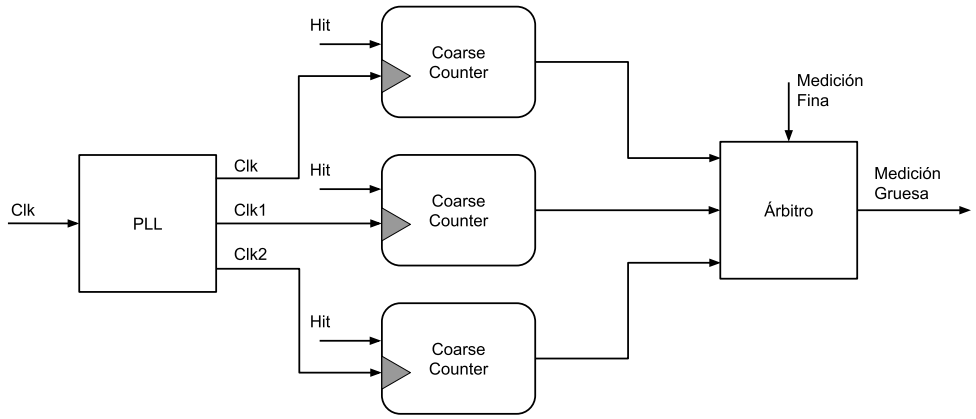


Figure 10: Arbitro implementado.

Estos tres clocks inducen cuatro casos a diferenciar, estos se plantearan para el flanco de entrada pero puede hacerse de la misma forma para el flanco de salida.
El caso base es suponer que el pulso llega antes del flanco de subida de todos los clocks, como muestra el Caso I de la Figura 11, o también llegar entre cada par de flancos como muestran los Casos II, III y IV. Para el primer caso, la línea de retardos captura la señal al inicio de la cadena pues el clock ocurre cerca de la llegada del pulso, sin embargo para el resto de casos donde el flanco de clock ya ocurrió es necesario esperar un periodo al próximo flanco de clock, en consecuencia se captura el flanco de entrada al borde de saturar la cadena de retardos, como muestra la Figura 12.

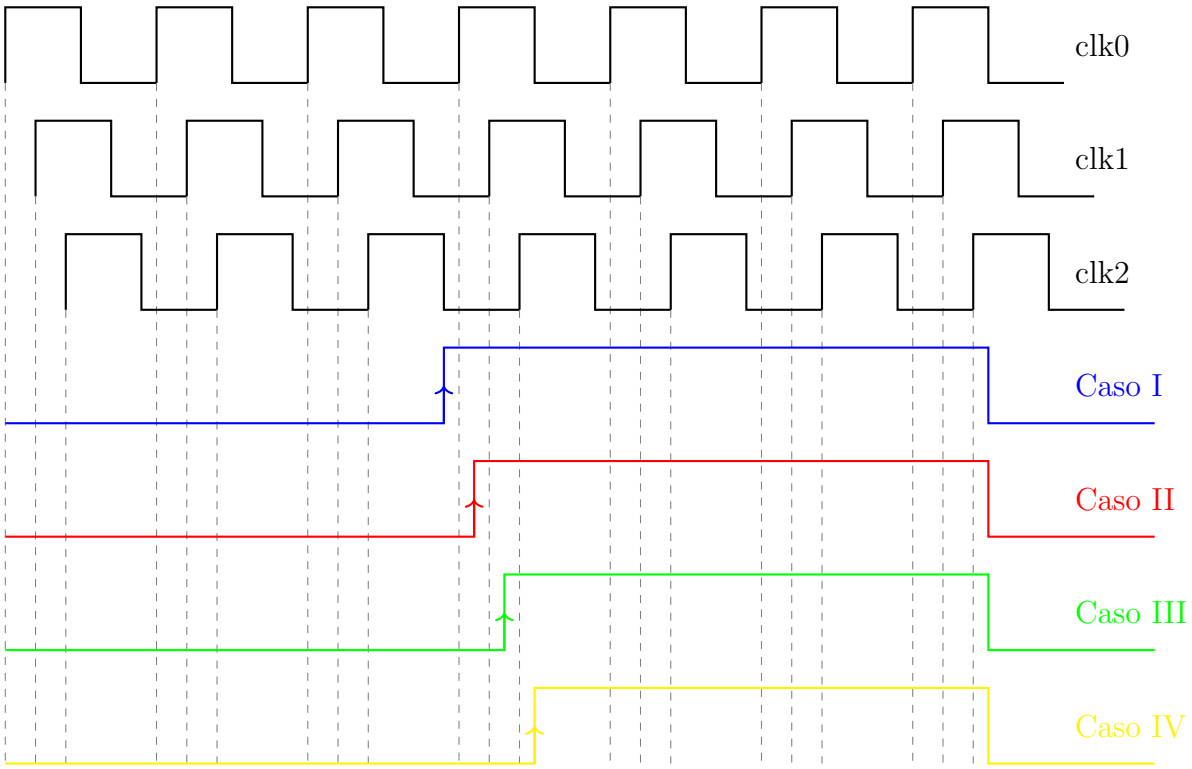


Figure 11: Todos los casos posibles de arbitraje para el flanco Start

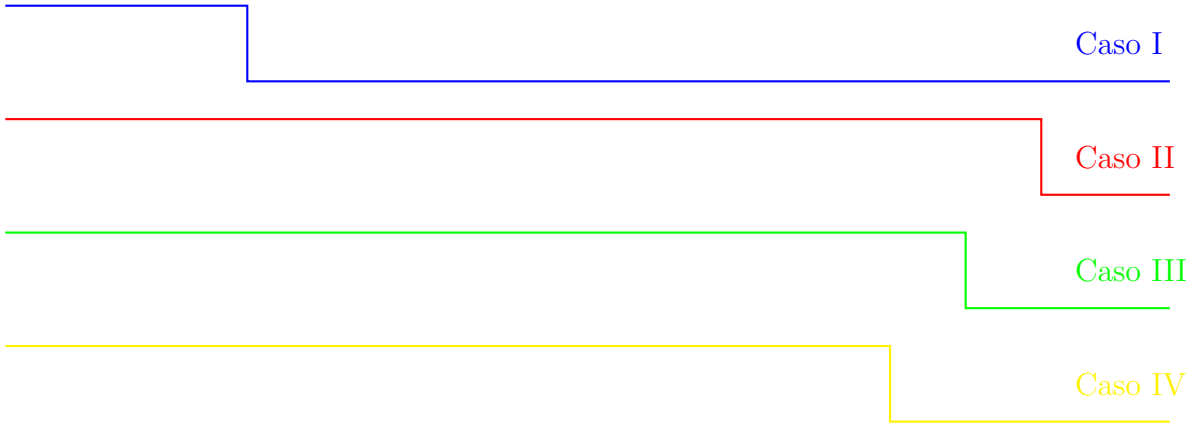


Figure 12: Registro *Start* de la cadena de retardos para cada caso de arbitraje.

Si analizamos cada caso con su coarse counter entonces obtendríamos los resultados que se presentan en la Tabla 1. Rápidamente nos podemos dar cuenta que si todos los Coarse Counters tienen la misma respuesta entonces el resultado de cualquiera de ellos es correcto, pero si no son iguales y el pulso es detectado en el extremo de la cadena, sabemos con certeza que la cuenta correcta es de tres periodos, pues el periodo incompleto está codificado en la respuesta fina. Es entonces posible conociendo las cuatro posibles condiciones de cuentas gruesas generar una respuesta apoyada en cuatro referencias: tres contadores y la respuesta fina.

Coarse counter según:	clk0	clk1	clk2
Caso I	4	4	4
Caso II	3	4	4
Caso III	3	3	4
Caso IV	3	3	3

Table 1: Cuenta gruesa del ejemplo presentado en la Figura 11, según cada clock generado por el PLL.

Finalmente se implementó la arquitectura presentada en [10] que se repite en la Figura 13.

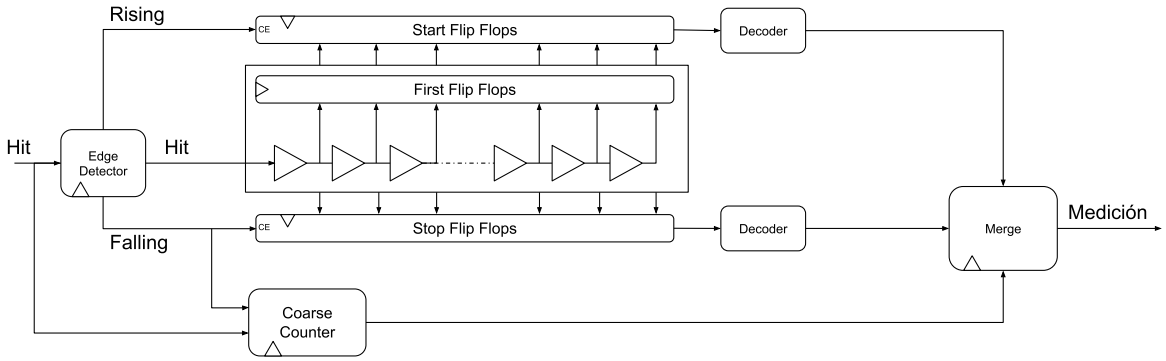


Figure 13: Arquitectura del TDC. Se ignora el routeo del clock.

Además para poder medir y procesar los datos se realizó el esquema de la Figura 14.

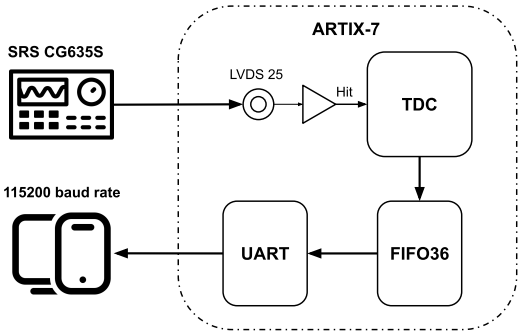


Figure 14: Esquema de medición

El generador de señales ingresa en diferencial a los conectores SMA de los puertos *GPIO* J33 y J34, de la placa de desarrollo AC701, que contiene una FPGA Xilinx Artix-7. Esta implementa el TDC que al ser activado realiza consecutivamente mediciones hasta llenar una memoria FIFO de 32 bits de largo, y 1024 palabras de profundidad. A su vez, cuando el usuario decide obtener los datos, se desactiva el TDC y la FPGA se conecta por puerte serie para descargar la información, para lo cuál se ha confeccionado un script en *python* que recibe los datos, separa cada palabra de 32bits en sus 10 bits de stop, 10 bits de start, y 12 bits de contador grueso, los transforma en el valor decimal correspondiente y los guarda en un archivo *".txt"*. Luego otro script se encarga de graficar una rápida visualización de los resultados. Tanto la conexión serie y la escritura/lectura de la memoria fueron probados en distintos casos para descartar errores.

El código, como algunos recursos de lectura utilizados, y detalles de implementación generales fueron publicados en el siguiente repositorio de Github.

Haciendo uso de un generador de clock "SRS CG635S" se realizó el "code-density test" con una frecuencia de 5.123456Mhz, obteniendo histogramas distribuidos para el flanco de start y de stop como muestra la Figura 15.

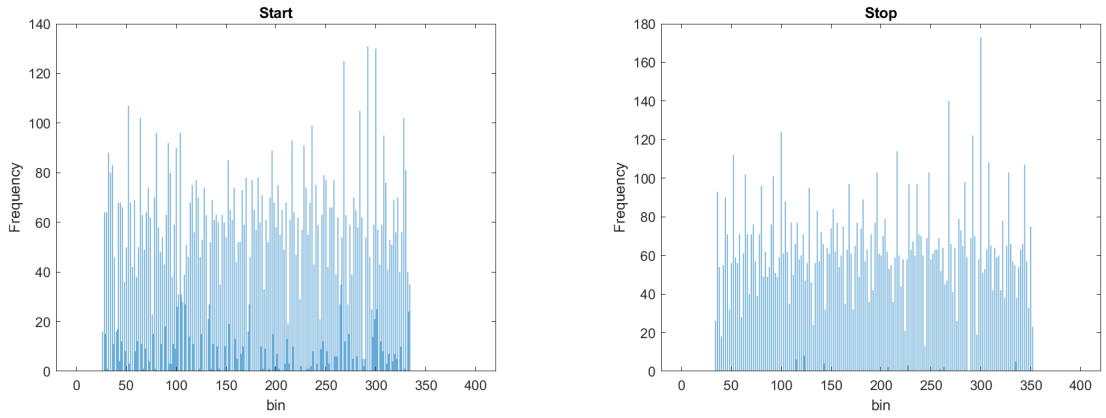


Figure 15: Histogramas del flanco de subida y flanco de bajada, para 10240 mediciones utilizando una frecuencia de 5.123456Mhz .

Es posible ver que los flancos de subida y bajada no se propagan de igual forma, pues no es la misma cantidad de bines para cada histograma. Podemos decir que el flanco stop se propaga de forma más rápida que el flanco start, pues alcanza un número de retardos mayor, lo que quiere decir que las celdas de retardo son más cortas cuando propagan una transición "1-0". Además, prestando especial atención al histograma stop se puede observar que los bines pares no son alcanzados, lo que sugiere que una única celda de retardo es demasiado pequeña para la velocidad de propagación del flanco.

Esto es completamente posible, ya que puede ocurrir que al implementar una compuerta los transistores complementarios no sean exactamente iguales. Tomemos como ejemplo la compuerta inversora 16. Cuando la entrada está en alto, el N-MOS entra en juego colocando un estado bajo a la salida. A la inversa, entra en juego el transistor PMOS. Esto pone en evidencia que si las características dinámicas de ambos transistores no son iguales, entonces la propagación de una transición alto-bajo/bajo-alto se dará de forma distinta.

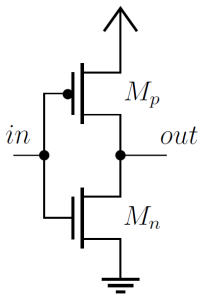


Figure 16: Circuito del buffer inversor.

Luego, para el flanco stop se utiliza desde el bin número 35, hasta el bin número 353, lo que nos da una cantidad de bines utilizados $N_c^{\text{stop}} = 318$, mientras que el flanco start ocupa desde el bin número 27, hasta el bin 335, utilizando un total de $N_c^{\text{start}} = 308$ elementos de retardo. Con esto se puede calcular que el retardo de cada celda es:

$$\hat{\tau}_{\text{stop}} = 15.72pseg\ , \qquad \hat{\tau}_{\text{start}} = 16.23pseg \tag{5}$$

Con esto se pueden construir los histogramas de linealidad, tal como muestran las Figuras 17, 18, utilizando las ecuaciones 3, 4, 2.

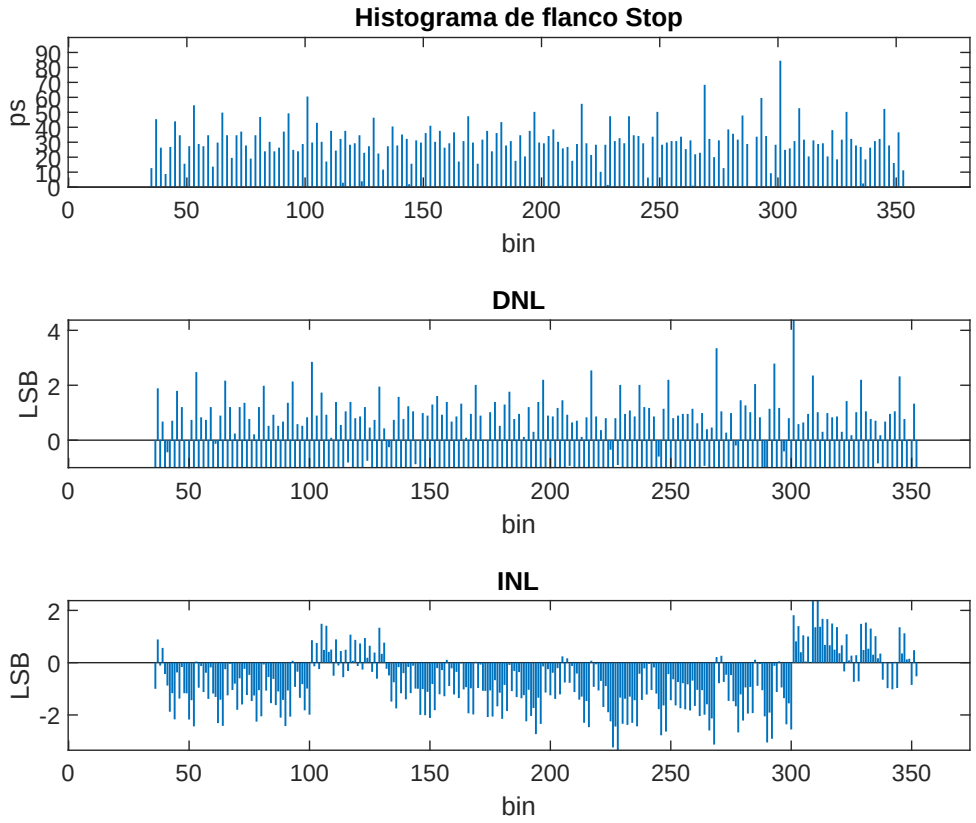


Figure 17: Histogramas de linealidad obtenidos en el code-density test para el flanco de stop.

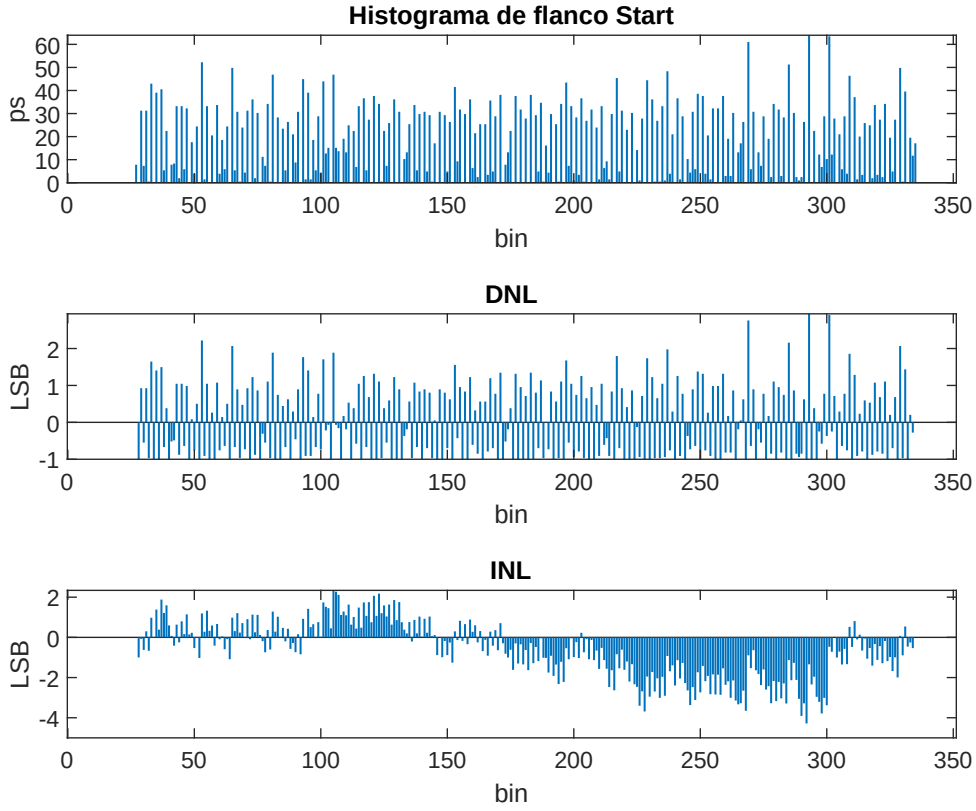
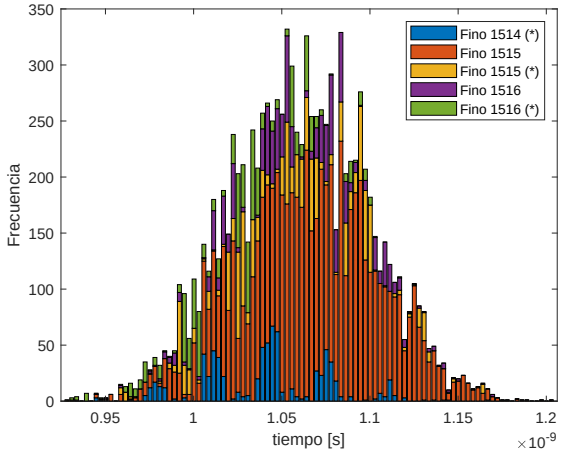


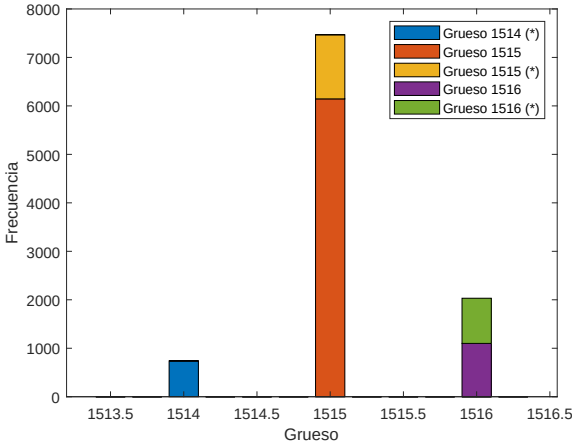
Figure 18: Histogramas de linealidad obtenidos en el code-density test para el flanco de start.

Se procedió a realizar un experimento de 10240 mediciones para seis frecuencias distintas. Para cada experimento, el tiempo fino es calculado tal como muestra el esquema 4, que induce:

$$t_{\text{fino}} = n_{\text{start}}\,\hat{\tau_{\text{start}}} - n_{\text{stop}}\,\hat{\tau_{\text{stop}}}\ . \tag{6}$$

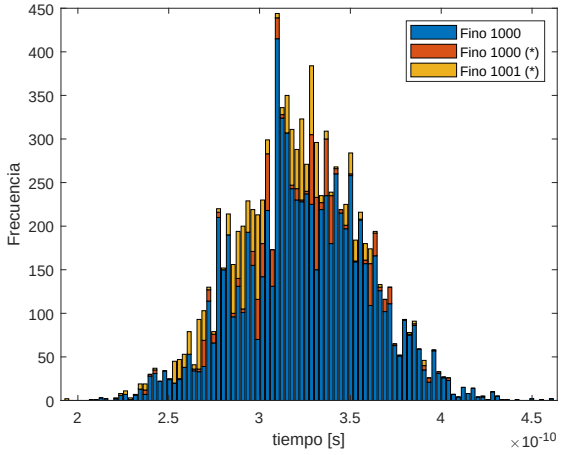


(a) Histograma de mediciones finas.

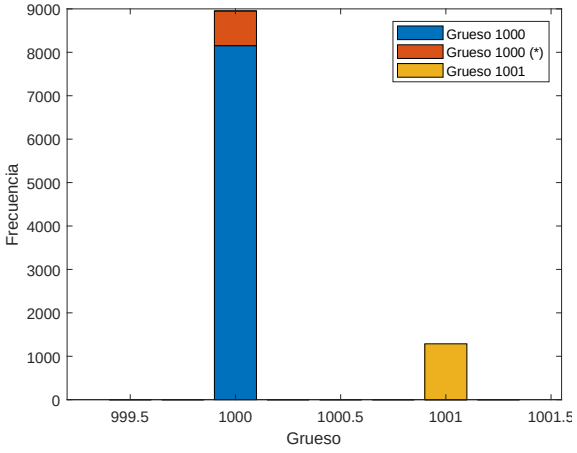


(b) Histograma de mediciones gruesas.

Figure 19: Histograma de 10240 mediciones utilizando una frecuencia de $f = 66k$Hz, $t_{\text{on}} = 7.57\ \mu$seg. Con asterisco aquellas mediciones cuyo contador grueso debió ser corregido.

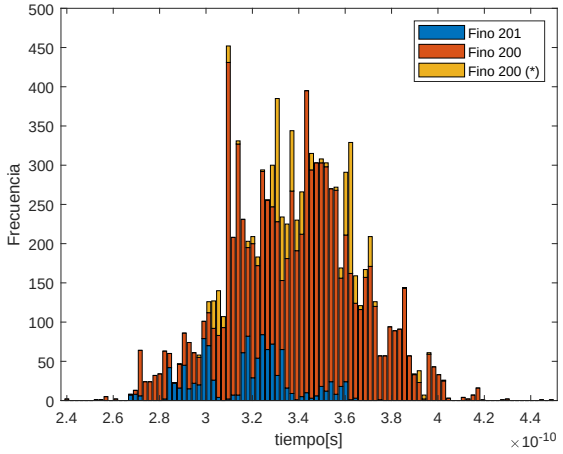


(a) Histograma de mediciones finas.

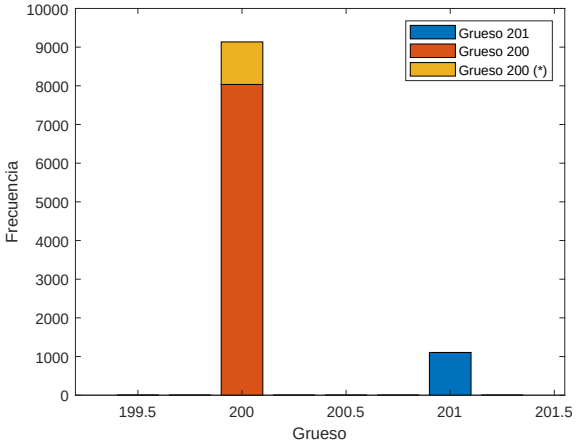


(b) Histograma de mediciones gruesas.

Figure 20: Histograma de 10240 mediciones utilizando una frecuencia de $f = 100k$Hz, $t_{\text{on}} = 5\ \mu$seg. Con asterisco aquellas mediciones cuyo contador grueso debió ser corregido.



(a) Histograma de mediciones finas.



(b) Histograma de mediciones gruesas.

Figure 21: Histograma de 10240 mediciones utilizando una frecuencia de f = 500kHz, t_{on} = 1 μseg. Con asterisco aquellas mediciones cuyo contador grueso debió ser corregido.

En resumen, se obtuvieron los resultados que muestra la tabla 2:

Frecuencia [Hz]	$E[X]$	$\sigma[X]$	Error [ps]
66k	1.06nseg	39.6pseg	303
100k	323pseg	35pseg	323
500k	338pseg	27.1pseg	338
1.225M	3.43nseg	29.8pseg	269
2M	271pseg	23.9pseg	271
3.3M	1.78nseg	26.7pseg	272
5M	286pseg	25.1pseg	286

Table 2: Parámetros de tiempo fino extraídos de cada experimento, cada uno con 10240 mediciones.

Queda en evidencia que existe un offset para todos los experimentos, que se calcula como $E[X] - t_{\text{on}}$, y da al rededor de 300ps. Aunque es difícil explicar este hecho, la primer conjetura es que este provenga del tiempo de *rising/fall* del generador, sin embargo este es $\leq$ 80ps para todo el rango de frecuencias, que se pudo constatar mediante un osciloscopio. No obstante es importante tener en cuenta la no linealidad de la cadena, que en particular afecta la forma de calcular el tiempo fino 6. Como no se utiliza el mismo tiempo de retardo (τ) medio para la propagación de subida y de bajada, entonces existe un error acumulado a medida que los flancos avanzan dentro de la cadena. Para verificar esto, a continuación se grafica un histograma de dos dimensiones para cada experimento, esto es: para cada valor n_{start} del eje-x se grafica el histograma de diferencia $n_{\text{start}} - n_{\text{stop}}$ en el eje-y, en otras palabras se grafica la diferencia de bines en función de dónde se encontró el bin start. Esto nos permite independizarnos de la temporalidad de los experimentos realizados, y observar el comportamiento de los retardos en función de la posición dentro de la cadena.

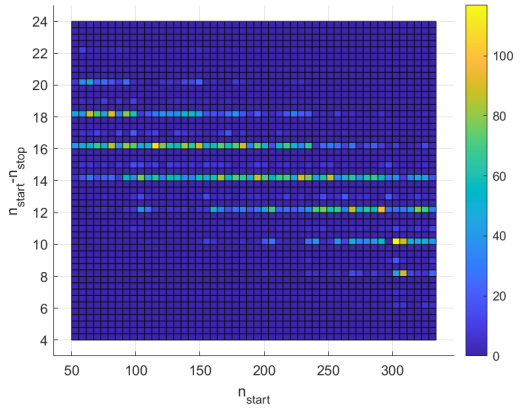


Figure 22: $f = 100k\text{Hz}$

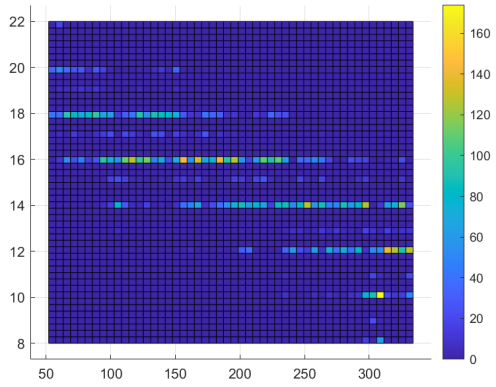


Figure 23: $f = 500k\text{Hz}$

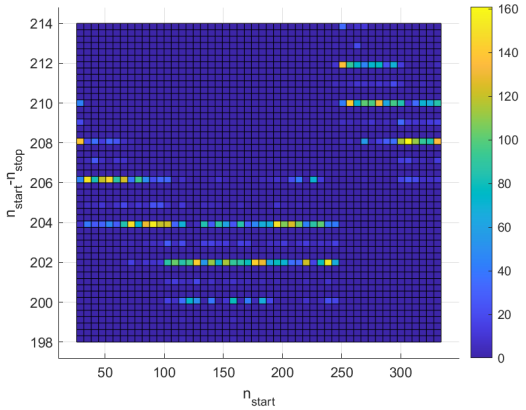


Figure 24: $f = 1.225M\text{Hz}$

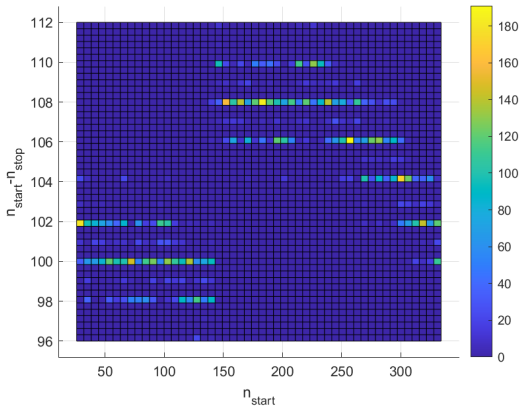


Figure 25: $f = 3.3M\text{Hz}$

Figure 26: Histograma que muestra la diferencia elementos de retardo start-stop en función de dónde se detectó el flanco de start dentro de la cadena, para cada experimento.

De estos resultados se realizaron dos conjeturas:

- Cuando la frecuencia es múltiplo de la frecuencia de clock (500kHz, 100kHz, 2MHz, 5MHz), se puede observar que la diferencias tienden a cero. Es posible entonces pensar que las no-linealidades acumuladas a medida que se avanza en la cadena tienden a compensarse, de forma que la diferencia tiende a cero, tal como se espera de una frecuencia exacta donde el contador grueso contiene toda la información necesaria.

- Cuando la frecuencia no es múltiplo del clock, y por lo tanto el tiempo fino a medir es distinto de cero, se pueden observar distintos patrones de dispersión. Si tomamos la tendencia de este histograma, se puede observar que la dispersión pico a pico es de al rededor de 10 bines. El corte que se observa ocurre principalmente porque dependiendo de la cantidad de tiempo fino que se debe medir, a medida que el pulso avanza en fase dentro de la cadena n_{stop} y n_{start} también avanzan en fase y por lo tanto se van transfieriendo cantidad de bines. Es decir, si tuvieramos una cadea de 300 elementos y el tiempo fino total a medir es de 200 bines, en un primer momento podemos encontrar que $n_{\text{start}} = 80$ y $n_{\text{stop}} = 280$, pero si siguen avanzando en fase podemos llegar a encontrar $n_{\text{start}} = 200$ y $n_{\text{stop}} = 100$ (27). Cuando existe un cruce de este tipo, las no-linealidades acumuladas en los extremos opuestos de la cadena son tan distintos que producen un salto en el histograma.

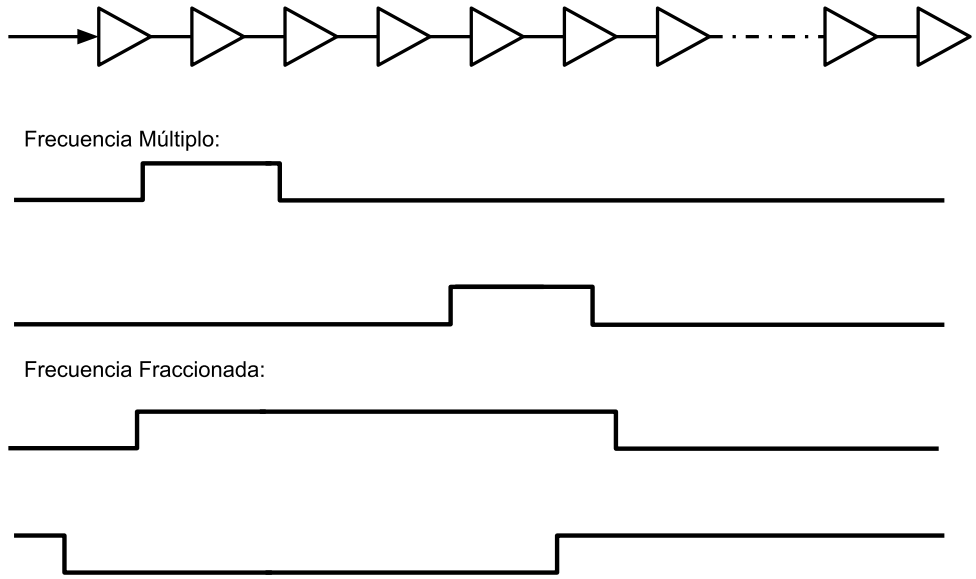


Figure 27: Explicación de la no-linealidad observada en los histogramas.

4.1 Calibración

Para mejorar la performance del TDC, se procedió a realizar cambios de implementación en el bitstream final. Originalmente el *placement & routing* realizado con la ayuda de la herramienta era el de la Figura 28. En principio se acortó la cadena y se la colocó a partir del slice $Y111$. Utilizando el parámetro de colocación fija `(* LOC = ``SLICE_X84Y111'' *)` en el primer *Carry4*, a continuación del buffer de clock como indica [10], la herramienta comprende que el el resto de elementos deben ser colocados en linea. Luego para que el routeo desde los retardos hasta sus registros sean lo más uniformes posibles, se generó la instanciación de cada uno de estos mediante un script en *python*, lo que permitió crear la columna de flip-flops de start y stop a continuación de estos, como muestra la Figura 29. Además utilizando constraints se logró excluir cualquier otro elemento dentro de la región donde se instanció la cadena de retardos, de forma tal que las conexiones no se vean intervenidas, obteniendo un routeo uniforme como muestra 30. Para lograrlo se muestran los *P_blocks* en la Figura 31, donde cada rectángulo pertenece a un constraint distinto. En el *Pblock_fine* se colocaron unicamente la cadena de retardos y sus respectivos registros, y el *Pblock_decode* sólo permite la colocación de compuertas que pertenezcan a los bloques de decodificación. Es así que los routeos desde *Pblock_fine* hacia *Pblock_decode* son lo más uniforme posibles sin recurrir a un routeo manual.

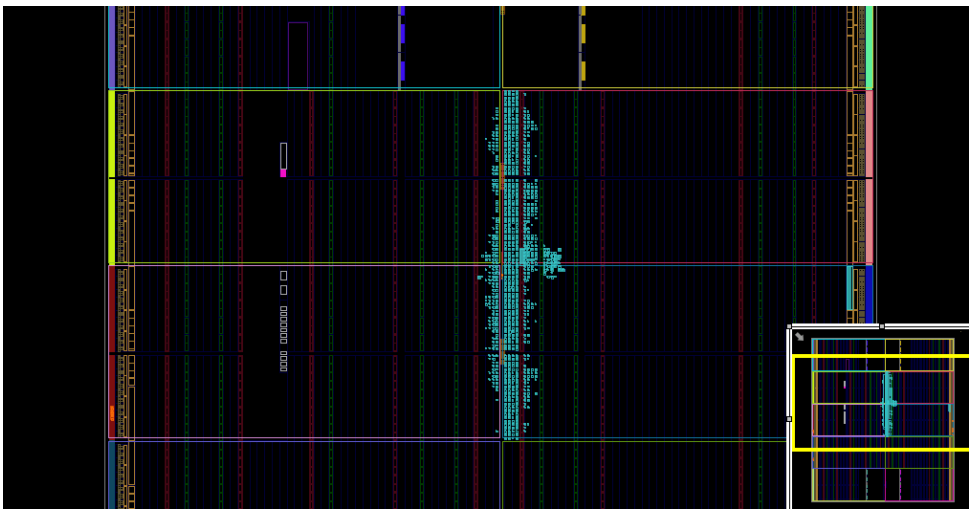


Figure 28: Implementación del TDC en la Artix-7.

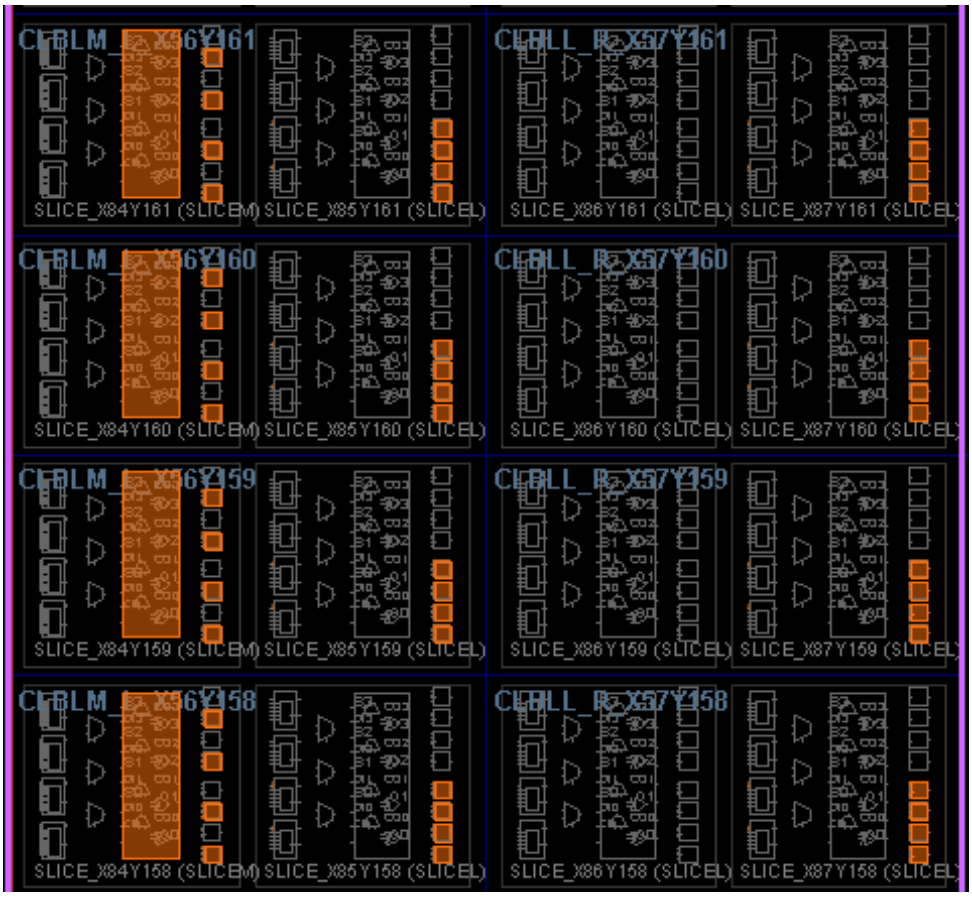


Figure 29: A la izquierda el slice contiene cada Carry4 y su registro. Un Slice después se colocan los registros start, y dos slices después los registros stop, tal como en [10].

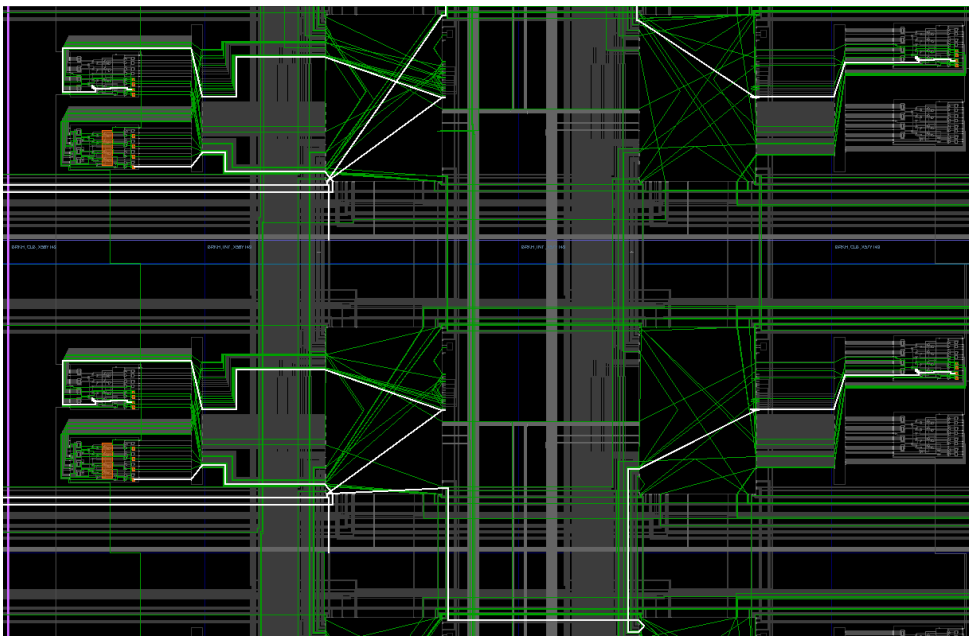


Figure 30: Routeo desde los retardos hacia sus registros.

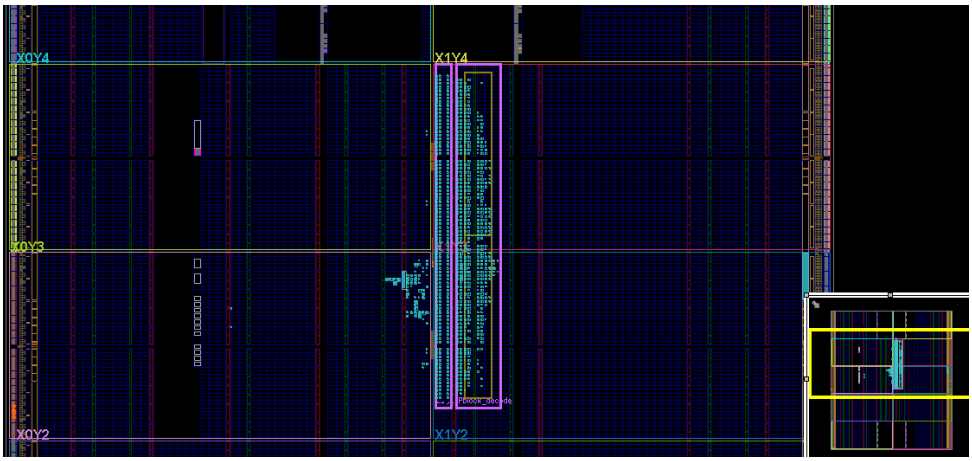


Figure 31: Floorplan del bitstream final. El primer rectángulo corresponde al *P_block* fino, y a su derecha el *P_block* para los decoders.

Con esta implementación, además se consiguió subir la frecuencia del sistema a $300MHz$, de forma que se acortó aún más el largo de la cadena. Las curvas de transferencia obtenidas son las que se muestran a continuación en las Figuras 33, 32

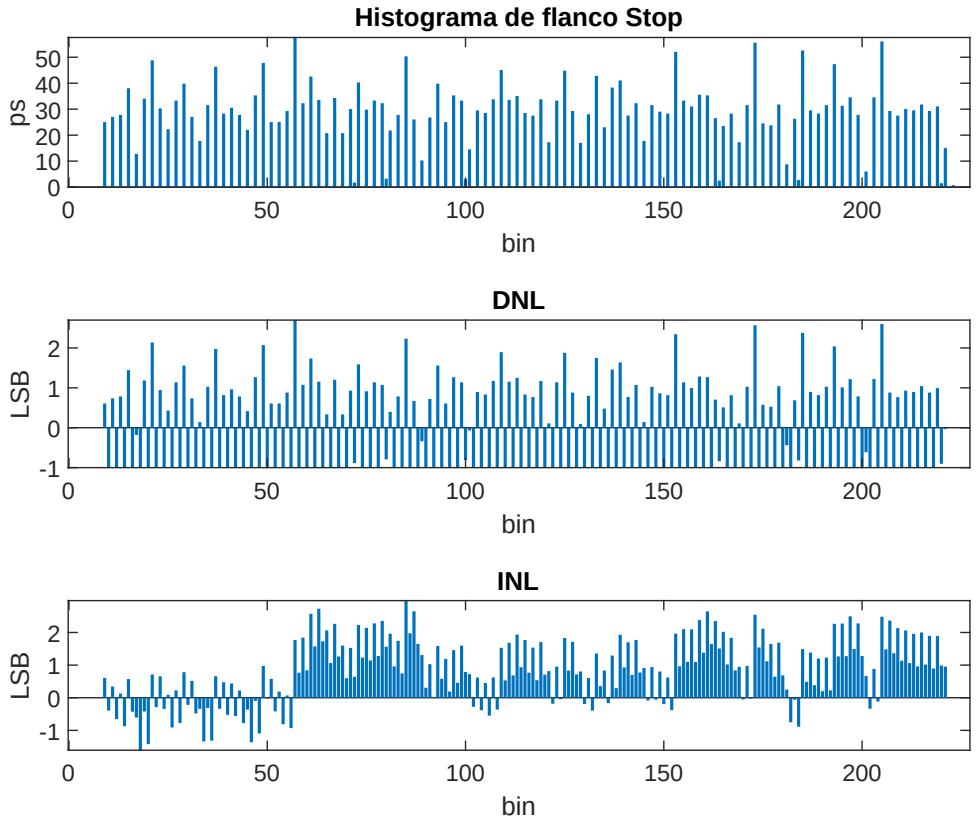


Figure 32: Histogramas de linealidad para la última implementación. Flanco stop.

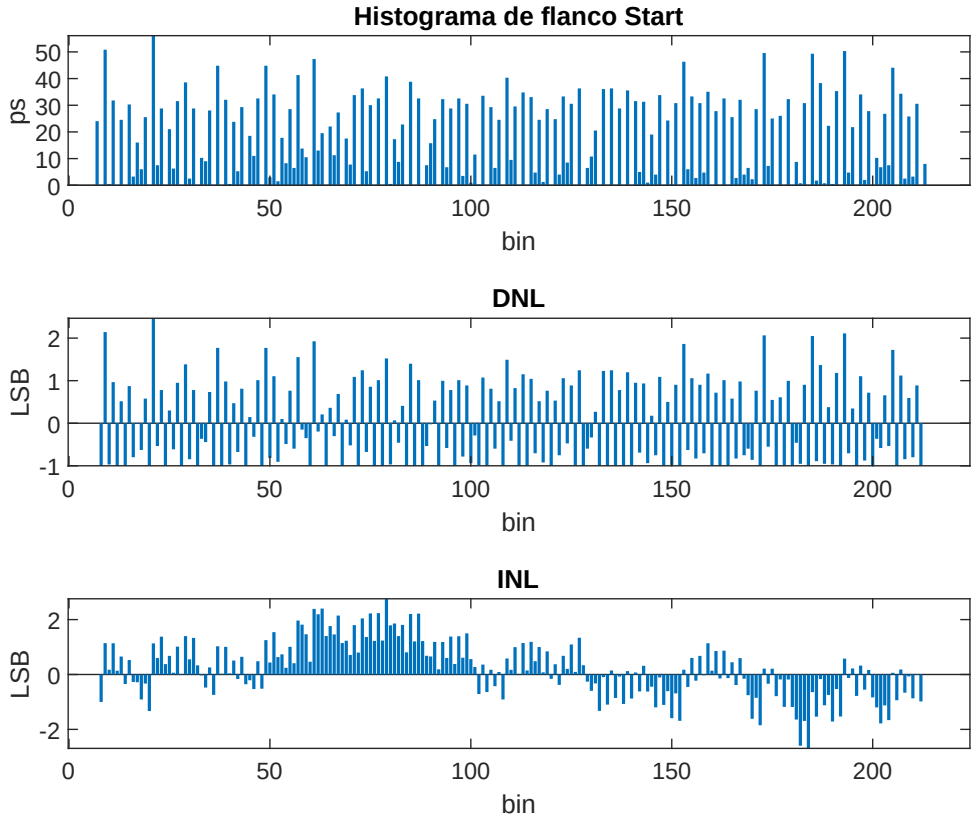


Figure 33: Histogramas de linealidad para la última implementación. Flanco start.

Por último es posible implementar una calibración en memoria, de forma tal que la respuesta del TDC tenga en cuenta las no-linealidades obtenidas en el "code-density test". Con esto, la idea principal es en vez de utilizar la ecuación 6, guardar en una tabla la curva de transferencia

obtenida al realizar la calibración, de forma tal que el tiempo medido en el i-ésimo bin es,

$$t_i = \frac{\tau_i}{2} + \sum_{k=0}^{i-1} \tau_k \ ,$$

donde la definición de τ_i proviene de la ecuación 1.
Tal como dice *Wu* en [12], es muy fácil caer en sólo computar la sumatoria cuando la implementación se vuelve engorrosa, pero sería un error omitir $\tau_i/2$ ya que esto implica calibrar a la mitad del bin en vez de hacerlo a los bordes, lo que disminuye el error RMS al mínimo.

4.2 Discusión

Se ha medido con la calibración final propuesta en este trabajo, con lo que se ha conseguido mejorar la precisión a valores de entre 19ps y 17ps (Figura 34). Aún así no se ha logrado corregir el offset, que se ha mantenido en el mismo rango. Se han reportado offsets de alrededor de 500ps, utilizando la misma topología en [13].

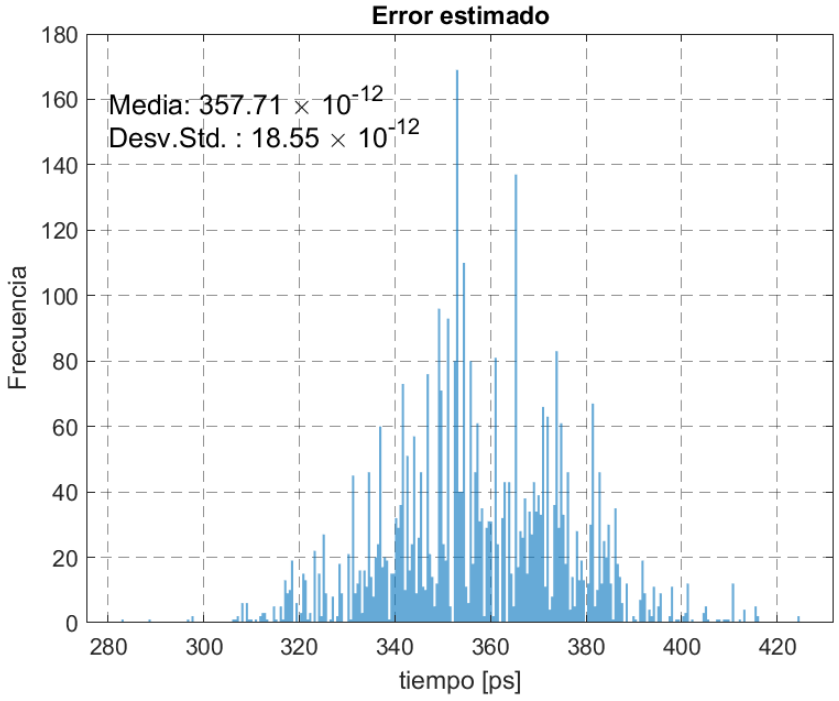


Figure 34: Error teórico al realizar 4096 mediciones con una señal de frecuencia f = 3.3MHz, t_{on} = 15.15μs.

Para desambiguar, es necesario seguir trabajando en caracterizar el buffer de entrada al ingresar la señal a través del estándar *LVDS_25*, y con esto conocer a ciencia cierta la dinámica de la señal que se distribuye a través de la cadena, o en su defecto, realizar un circuito de acondicionamiento de forma tal que, conocida su transferencia, se pueda tener en cuenta dentro de la calibración final.
A pesar de estos problemas, se ha logrado satisfactoriamente alcanzar una precisión del orden de la decena de picosegundos, con lo cuál es posible utilizar esta implementación en las aplicaciones propuestas. Creemos que los horizontes de trabajo para futuras implementaciones son:

- Sincronización de la medición fina y gruesa.
- Auto-calibración en hardware.

- Comparativa con la implementación de un decodificador *ones-counter*.
- Implementación multi-cadena.

Referencias

[1] Samuele Altruda et al. "PicoTDC: a flexible 64 channel TDC with picosecond resolution". In: *Journal of Instrumentation* 18.07 (July 2023), P07012. ISSN: 1748-0221. DOI: 10.1088/1748-0221/18/07/p07012. URL: http://dx.doi.org/10.1088/1748-0221/18/07/P07012.

[2] K. Maatta and J. Kostamovaara. "A high-precision time-to-digital converter for pulsed time-of-flight laser radar applications". In: *IEEE Transactions on Instrumentation and Measurement* 47.2 (Apr. 1998), pp. 521–536. ISSN: 0018-9456. DOI: 10.1109/19.744201. URL: http://dx.doi.org/10.1109/19.744201.

[3] Juan Ignacio Morales. "Diseño de sistemas microelectrónicos basados en alta resolución temporal". PhD thesis. Bahía Blanca: Universidad del Sur, 2020.

[4] Rui Machado, Jorge Cabral, and Filipe Serra Alves. "Recent Developments and Challenges in FPGA-Based Time-to-Digital Converters". In: *IEEE Transactions on Instrumentation and Measurement* 68.11 (2019), pp. 4205–4221. DOI: 10.1109/TIM.2019.2938436.

[5] Chong Liu and Yonggang Wang. "A 128-Channel, 710 M Samples/Second, and Less Than 10 ps RMS Resolution Time-to-Digital Converter Implemented in a Kintex-7 FPGA". In: *IEEE Transactions on Nuclear Science* 62.3 (June 2015), pp. 773–783. ISSN: 1558-1578. DOI: 10.1109/tns.2015.2421319. URL: http://dx.doi.org/10.1109/TNS.2015.2421319.

[6] Wassim Khaddour, Wilfried Uhring, Foudil Dadouche, Norbert Dumas, and Morgan Madec. "Calibration Methods for Time-to-Digital Converters". In: *Sensors* 23.5 (Mar. 2023), p. 2791. ISSN: 1424-8220. DOI: 10.3390/s23052791. URL: http://dx.doi.org/10.3390/s23052791.

[7] X. Qin et al. "A 1.15-ps Bin Size and 3.5-ps Single-Shot Precision Time-to-Digital Converter With On-Board Offset Correction in an FPGA". In: *IEEE Transactions on Nuclear Science* 64.12 (2017), pp. 2951–2957. DOI: 10.1109/TNS.2017.2768082.

[8] J. Kalisz, R. Szplet, J. Pasierbinski, and A. Poniecki. "Field-programmable-gate-array-based time-to-digital converter with 200-ps resolution". In: *IEEE Transactions on Instrumentation and Measurement* 46.1 (Feb. 1997). Conference Name: IEEE Transactions on Instrumentation and Measurement, pp. 51–55. ISSN: 1557-9662. DOI: 10.1109/19.552156. URL: https://ieeexplore.ieee.org/document/552156/citations#citations (visited on 07/16/2024).

[9] Claudio Favi and Edoardo Charbon. "A 17ps time-to-digital converter implemented in 65nm FPGA technology". In: *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. FPGA '09. New York, NY, USA: Association for Computing Machinery, Feb. 2009, pp. 113–120. ISBN: 978-1-60558-410-2. DOI: 10.1145/1508128.1508145. URL: https://doi.org/10.1145/1508128.1508145 (visited on 07/15/2024).

[10] Rui Machado, Luis A. Rocha, and Jorge Cabral. "A novel synchronizer for a 17.9ps Nutt Time-to-Digital Converter implemented on FPGA". In: *2018 AEIT International Annual Conference*. Oct. 2018, pp. 1–6. DOI: 10.23919/AEIT.2018.8577365. URL: https://ieeexplore.ieee.org/document/8577365 (visited on 07/16/2024).

[11] Yonggang Wang, Jie Kuang, Chong Liu, and Qiang Cao. “A 3.9-ps RMS Precision Time-to-Digital Converter Using Ones-Counter Encoding Scheme in a Kintex-7 FPGA”. In: *IEEE Transactions on Nuclear Science* 64.10 (Oct. 2017). Conference Name: IEEE Transactions on Nuclear Science, pp. 2713–2718. ISSN: 1558-1578. DOI: 10.1109/TNS.2017.2746626. URL: https://ieeexplore.ieee.org/document/8022888 (visited on 07/25/2024).

[12] Jinyuan Wu. “Several Key Issues on Implementing Delay Line Based TDCs Using FPGAs”. In: *IEEE Transactions on Nuclear Science* 57.3 (June 2010), pp. 1543–1548. ISSN: 1558-1578. DOI: 10.1109/tns.2010.2045901. URL: http://dx.doi.org/10.1109/TNS.2010.2045901.

[13] Rui Pedro Oliveira Machado. “Readout circuit for time-based automotive sensors”. por. Accepted: 2022-03-02T17:23:15Z Journal Abbreviation: Circuito de leitura para sensores automóveis baseados em tempo de voo. Tesis Doctoral. July 2020. URL: https://repositorium.sdum.uminho.pt/handle/1822/76359 (visited on 08/19/2024).