Técnicas Avanzadas de Diseño Digital

UNICEN

Implementación de un canal LVDS con codificación 8b10b

Ing. Federico De La Cruz Arbizu

Dictado por

Ing. Guillermo Jaquenod

Septiembre 2018

Índice

1. Motivación y resumen

La idea de la presente implementación nace en el marco del desarrollo del Tomógrafo por Emisión de Positrones Argentino (AR-PET), en el cual me encuentro trabajando hace más de 4 años. Gracias a este proyecto me sumergí en el mundo del VHDL.

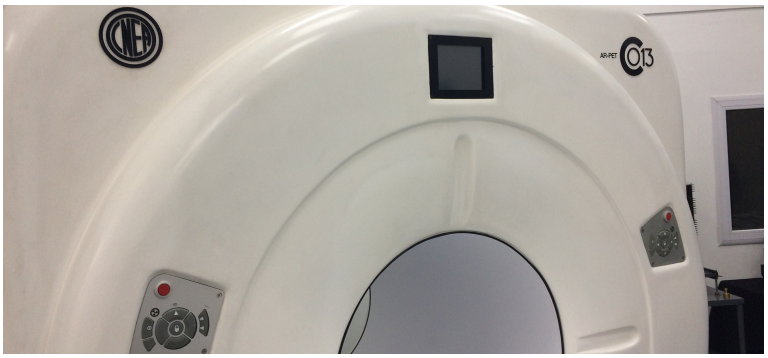

Figura 1: AR-PET

El AR-PET ya fue trasladado al Hospital de Clínicas José de San Martín en la Ciudad de Buenos Aires con el objetivo de llevar a cabo las validaciones de la técnica de adquisición y, si los resultados lo permiten, convertirse en el primer PET de diagnóstico en un Hospital Público en el país, con un futuro prometedor de difuminarse federalmente.

Este tomógrafo cuenta con 6 cámara Gamma (o cabezales) independientes, que en nuestro caso particular, funcionarán sincrónicas en búsqueda de eventos simultáneos (coincidencias) de aniquilación Positrón-Electrón, que de manera electrónica, se confinarán en Líneas de respuesta (LORs) para una posterior reconstrucción y formado de imágenes diagnósticas.

Como ya se verá en el desarrollo del informe, el canal crítico de transferencia de información es el conjunto de líneas LVDS (Low-Voltage Differential Signaling) por donde los cabezales transfieren los eventos detectados hacia el procesador de coincidencias. Cada cabezal tiene la capacidad de procesar 1 millón de eventos por segundo, y dado que el procesamiento es en tiempo de vuelo, no se puede perder tiempo en la descarga de los mismos.

Hoy en día el control de errores de este canal (LVDS a 80MHz) se lleva por medio de una simple paridad, lo cual nos ha demostrado su bajo rendimiento en intensos ensayos con el sistema en su peor condición (gran cantidad de eventos a transmitir mientras el equipo se encuentra girando).

El agregado de la codificación 8b10b no solo nos brinda su control de errores inherente sino también la posibilidad de introducir tramas de sincronismo y

mejorar el nivel de continua del canal. Además, y ya que se está implementando en la totalidad del sistema, se agrega un *CheckSum* de 8 bits en el espacio libre para validar las tramas.

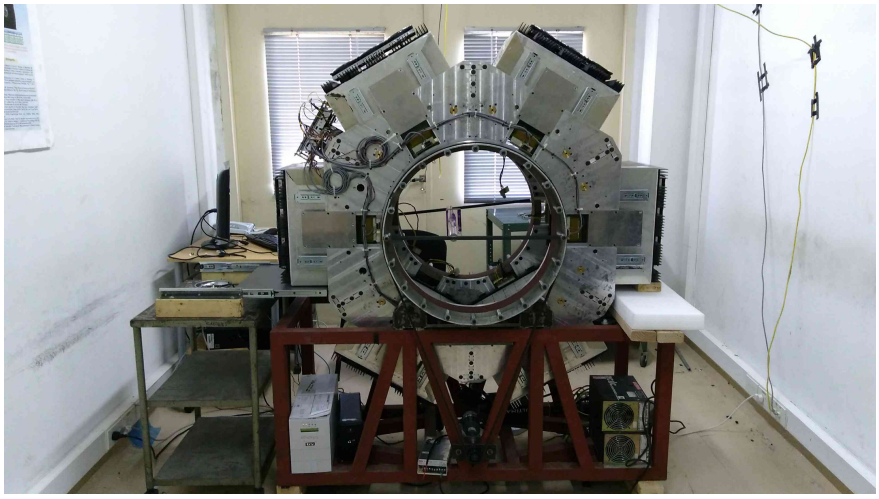

Figura 2: AR-PET al desnudo

2. Introducción

Solo a modo informativo se procede a presentar los componentes y características principales del equipo y sus funcionalidades para plantear una base de conocimientos de la técnica utilizada y las necesidades que surgen de cada una de ellas.

2.1. FPGA

La necesidad de procesar gran volúmenes de datos en tiempos extremadamente cortos y con procesamientos altamente sincrónicos han obligado a que la totalidad del pre-procesamiento de los eventos sean realizados en FPGA (Field-Programmable Gate Array) dado su versatilidad de bajo nivel para nuclear escenarios donde el Hardware y su alta densidad de señales necesitan ser manipulados de manera exclusiva a velocidades realmente elevadas. Recordar que el procesamiento completo de eventos se debe realizar en menos de 1μs.

Hoy en día y ya pensando en su versión final se está trabajando con un SOM (System On Module) de Trenz con una Artix-7 100T[1]. En particular en el proyecto se utiliza el TE-0712 que se muestra en la Figura 3.

Figura 3: SOM Artix-7

La versión anterior a este SOM fue un diseño convencional con Spartan-6, pero hubo tres razones por las cuales se decidió rediseñar las placas: conexiones no previstas en ese primer diseño que facilitarían la modularización del sistema, la soldadura de la FPGA era BGA (Ball Gris Array) que en una placa de gran tamaño los pandeos de la misma hace que este tipo de soldadura no resista esas fuerzas, y los problemas de ESD (ElectroStatic Discharge) introducidos por las grandes longitudes de cables entre procesadores y fue la culpable de la rotura de varios chips.

2.2. Procesamiento Planar

Cada uno de los 6 cabezales del PET cuenta con un cristal continuo de Ioduro de Sodio (NaI) de 30cm x 40cm que actúa como centellador, es decir, genera luz al ser impactado por un fotón.

Debajo del cristal continuo hay un array de 48 PMTs (PhotoMultiplier Tube), los cuales se encargarán de digitalizar, con un procesamiento sobre Spartan-3, la luz centellada a un valor energético proporcional a la porción de luz que le corresponde del total que generó el fotón en el centellador, y una marca temporal indicativa del momento en el cuál ese evento fue detectado. Toda esta información es recibida por una Artix-7, que se encargará de no solo filtrar temporalmente los eventos para descartar los que no participaron del mismo destello, sino de realizar la triangulación espacial (por promedio pesado de Anger[2]) y calcular la posición en la que el fotón impactó en el cristal. De esta manera se confecciona una cámara Gamma, que es sistema capaz de detectar no solo la energía de los fotones sino también la posición del mismo. En la Figura 4 se observa la placa de Procesamiento Planar.

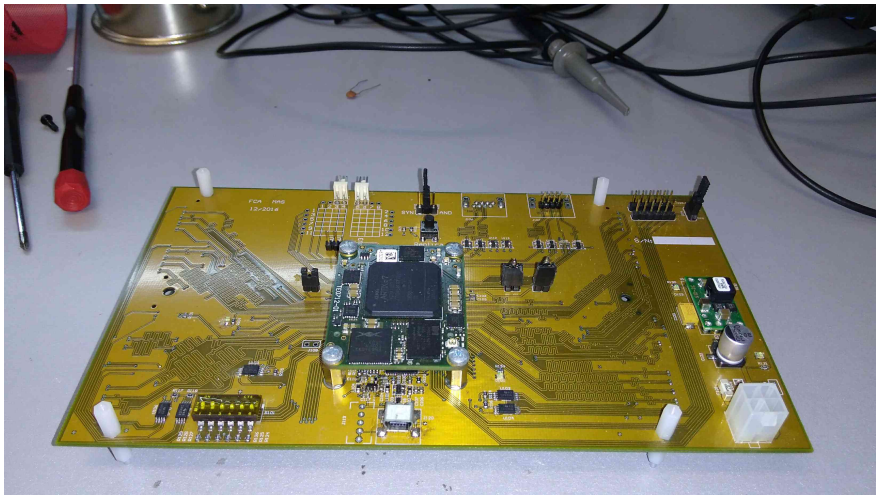

Figura 4: Procesamiento Planar

2.3. Coincidencias

La totalidad de la técnica de PET se basa en la utilización de Positrones como partículas de decaimiento radiactivo[3]. Su importancia se concentra en que estas partículas, al interaccionar con un electrón, se aniquilan generando dos fotones de 511keV desfasados 180°. En la Figura 5 puede visualizarse este fenómeno.

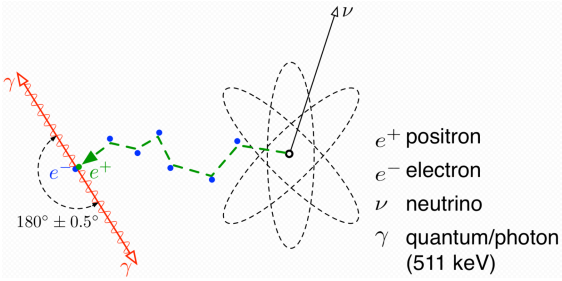

Figura 5: Aniquilación Positrón-Electrón

Esta característica física de la aniquilación Positrón-Electrón permite poder generar su línea de respuesta (LOR - Line Of Response) detectando ambos fotones con dos detectores enfrentados, como se ve en la Figura 6.

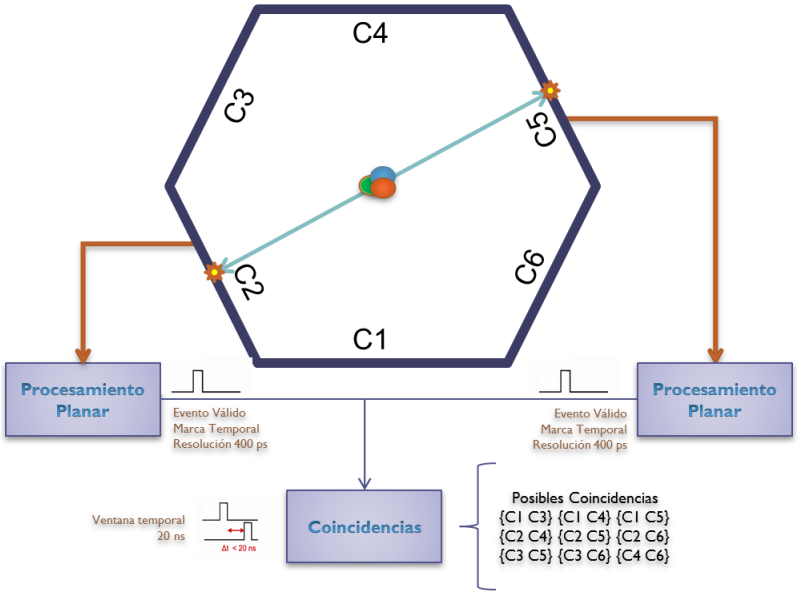

Figura 6: Generación de una LOR

Esta búsqueda se realiza dentro de una ventana temporal de 20ns, tiempo máximo en que se estima que un fotón puede tardar en viajar desde algún punto del FOV (Field Of View) hasta los cristales en sí, es decir, que si las marcas temporales de cada evento distan en menos de esta ventana temporal serán considerados coincidencia (es por esto que es crucial la sincronización de todo el sistema). En el caso específico del PET, será necesario detectarlos con dos cabezales enfrentados (coordenadas XY de cada uno), de manera de realizar una colimación electrónica de la coincidencia. La acumulación de estas LORs, registrarán intensificaciones en aquellas zonas donde la actividad de positrones está más concentrada, pudiendo así obtener las imágenes buscadas.

Para esto, al paciente se le inyecta una droga a base de glucosa marcada con un átomo radiactivo de Flúor (FDG). La metabolización de la misma logra que la glucosa se deposite en aquellas zonas orgánicas que demanden mayor cantidad de energía, es decir, que presenten mayor actividad celular, tanto en su dinámica funcional como reproductiva. Es por esto que la mayor parte del FDG inyectado se depositará finalmente en cerebro, corazón, tracto urinario y tumores celularmente activos[4]. De esta manera, las desintegraciones radiactivas emergerán de estos lugares específicos del cuerpo, pudiendo así, tras la acumulación de LORs, estimar su ubicación.

En las Figuras 7 y 8 se pueden ver ambos lados de la placa encargada de toda esta etapa de búsqueda de las coincidencias y posterior descarga de los datos pre-procesados a la PC mediante UDP por un enlace Ethernet de 100Mbps. Se utilizó este canal de comunicación ya que está incorporado el PHY dentro del SOM de Artix-7 y su implementación fue mucho más ágil para su uso que la implementación previa realizada por USB-HS.

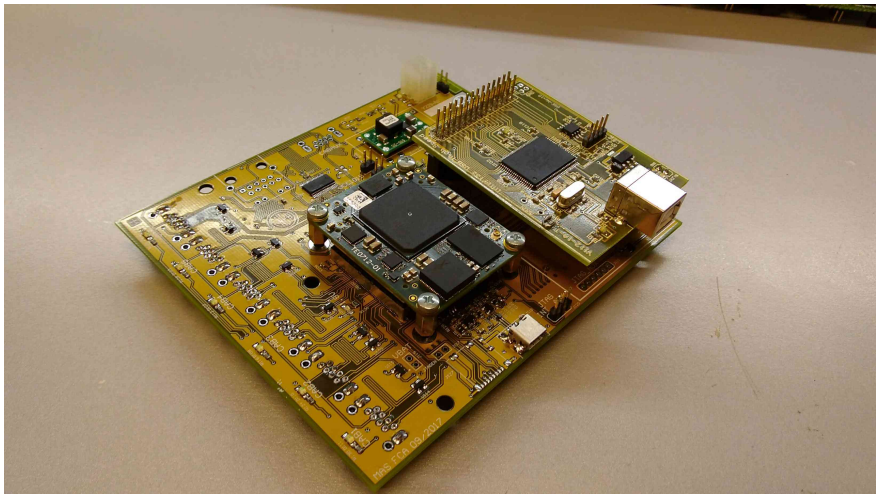

Figura 7: Coincidencias

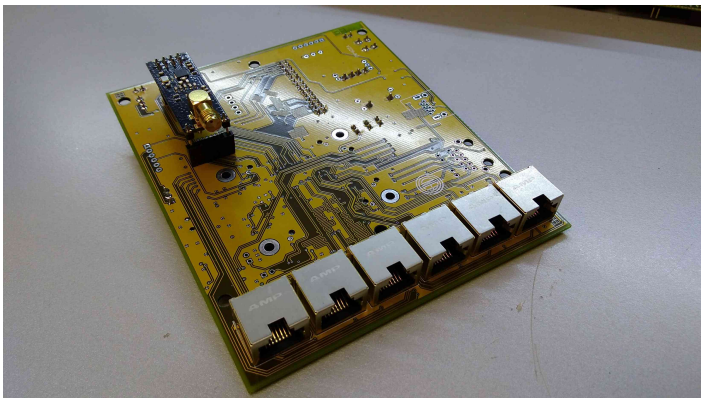

Figura 8: Coincidencias Bottom

2.4. Estado del arte

Para que todo esto sea posible hay un punto clave en el cual se estuvo trabajando la mayor cantidad de tiempo ya que sin eso no sería posible cumplir con las exigencias temporales de esta técnica: sincronización.

Los pulsos que entregan los PMTs, debido a la respuesta del centellador, tienen una constante temporal alrededor de los 120ns y la ventana temporal intracabezal (diferencia temporal entre los PMTs del mismo cabezal) para determinar que la información de 2 o más PMTs contiguos pertenecen al mismo evento es de aproximadamente 1ns. Por otro lado, la ventana temporal intercabezal (diferencia temporal de los eventos detectados por dos cabezales diferentes) para determinar las coincidencias entre detectores y así validar las LORs está alrededor de los 20ns. Es por esto que se trabaja con un reloj maestro bajo de 20MHz, generado por la placa de Coincidencias, con el cual se sincronizan las 294 FPGAs restantes (6 Artix-7 y 48 Spartan-3 por cada una de estas), generando cada una, en fase con este, sus relojes de funcionamiento.

Con la finalidad de poder procesar 1Mevento/s se planteó que la descarga desde cada cabezal hacia la placa de Coincidencias debe ser de 1μs, y debido a que la información necesaria ocupa 64 bits (marca temporal, energía y coordenadas XY), se planteó un canal de 80MHz, que con el Header, paridad y bits de finalización se completaban los 80bit para cumplir con el tiempo indicado.

Si bien todo el equipo trabaja en el mismo dominio de reloj, el canal de 2 metros de LVDS con un equipo girando y alimentado por escobillas nos ha mostrado que es propenso a errores aleatorios en la decodificación de los pulsos LVDS que no pueden detectarse con un simple control de errores de paridad.

En este trabajo lo que se plantea es la posibilidad de incorporar un control

de errores más robusto que nos permita detectar y rechazar eventos erróneos afectados exclusivamente por las condiciones del canal de comunicación.

En la Figura 9 se intenta graficar un resumen del esquema entero de procesamiento, partiendo de las Spartan-3, pasando por el Procesamiento Planar que calcula tanto la Energía como las coordenadas XY del evento, la comunicación con Coincidencias (canal LVDS donde se pretende implementar la codificación), la búsqueda y generación de las LORs y por último la descarga por Ethernet hacia la PC interna (que posteriormente enviará los archivos de adquisición hacia la PC de reconstrucción por WiFi).

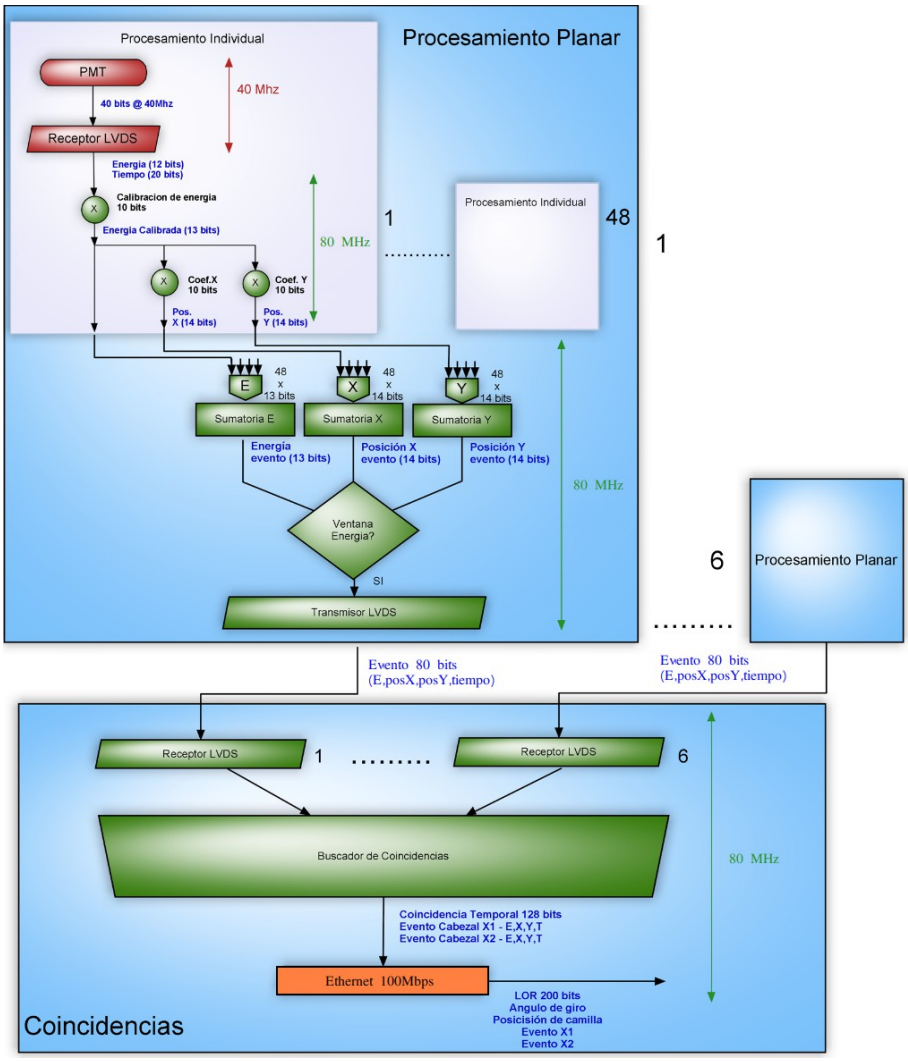

Figura 9: Cadena de procesamiento de eventos

2.5. Codificación 8b10b

A la hora de transmitir información, se trata de elegir un protocolo tal que no sea tan grande la sobrecarga de información que agregue, permita recuperación de reloj y asegure confiabilidad para detección y corrección de errores.

Por ejemplo, en las comunicaciones BASE-100T de Ethernet (100 Mbps) la codificación Manchester[5] es la elegida ya que su incorporación de transiciones facilita la recuperación de reloj y no sobrecarga la línea de información extra. Como desventaja, esto hace que aumente el ancho de banda de la línea al doble, pero así y todo es una propiedad que puede ser soportada debido a su baja velocidad.

En cambio, en el caso de BASE-1000T de Ethernet (1 Gbps), el doble de ancho de banda respecto al original se volvería inmanejable o contraproducente, por lo que agregar flancos por cada valor de bit no sería una opción viable.

Es aquí donde la codificación 8b10b toma relevancia ya que con una sobrecarga del 25 % en bits de codificación (lo mismo sucede con el ancho de banda si se desea mantener la tasa de transferencia) se permite tener un control en los bits a transmitir, sin perder las propiedades de recuperación de reloj (ya que se limita la cantidad de 0's o 1's consecutivos) y aportando nuevas herramientas para la detección de errores y control de flujo.

El hecho de agregar 2 bits a la palabra original a emitir (4 veces más de palabras posibles), no solo permite eliminar las palabras que no aportan a la sincronización de las partes, sino que se puede elegir la polaridad neta de cada *burst* de bits (controlando el valor de continua permanente del canal). Otra ventaja es agregar símbolos adicionales además de los 256 símbolos originales, tales como palabras de control que se incorporan para establecer eventos especiales en la comunicación, así como ayudas de sincronización, inicios de tramas, aviso de inactividad, etc. Además esta incorporación suma a la lista aquellas palabras que no cumplen el pre-requisito de cantidad de 0's o 1's consecutivos, ayudando a la detección y corrección de errores en aquellos casos donde estas palabras prohibidas se hagan presentes en algún instante de la comunicación.

Por lo tanto, la etapa de codificación se encarga de transformar una palabra de 8 bits en una de 10 bits respetando principalmente dos consideraciones básicas:

- Las palabras codificadas podrán ser 6/4 (6 unos y 4 ceros), 5/5 o 4/6.
- No podrán contener más de 4 símbolos (unos o ceros) iguales consecutivos a no ser que se trate de un *Comma Character* (caracteres especiales).

Sabiendo esto y antes de comenzar con la manipulación de bits de la conversión en sí, se hará un pequeño recuento de la disponibilidad de palabras real

que se tiene en este sistema.

Un sistema de 8 bits da la posibilidad de utilizar 256 posibilidades, y uno de 10 bits ofrece 1024 opciones, es decir, hay 768 posibilidades extra en este pasaje que falta definir cómo se van a utilizar.

El nuevo sistema de 10 bits contará con la incorporación de caracteres especiales para funcionalidades específicas. Estas palabras se las denomina *Comma Character*. Existen 12 de estas, y dependiendo el sistema donde se aplique tienen diferentes funcionalidades. A modo de versatilizar su utilización existen en sus dos versiones de polaridad, con lo que se estarán utilizando 24 palabras de 10 bits para este tipo de caracteres.

De las 1000 palabras posibles restantes, 461 palabras serán generadas desde las 256 originales, combinando ambas polaridades en aquellas que puedan ayudar a balancear la polaridad neta de la línea de transmisión.

Es así como quedan finalmente sin utilizar 563 palabras de las 1024 posibles, es decir, que casi la mitad de las posibilidades en 10 bits serán consideradas errores de transmisión.

A continuación se presenta la formación de las 461 palabras válidas, mostrando en primer instancia la nomenclatura utilizada en todo este proceso.

Considerando que se tiene una palabra de 8 bits inicial conformada de la siguiente manera

$$W_{8b} = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0,$$

definiendo los *Data Character* (caracteres comunes) como

$$D.x_2 x_1.x_0$$

donde

$$[x_2 x_1]_{10} = [b_4 b_3 b_2 b_1 b_0]_2$$

y

$$[x_0]_{10} = [b_7 b_6 b_5]_2.$$

Por ejemplo

$$D.11.3 = 01101011.$$

De la misma manera se generarán los *Comma Character*

$$K.28.5 = 11011100.$$

Una vez definidas las nomenclaturas a utilizar resta establecer cómo serán las conversiones a realizar para pasar de una palabra de 8 bits a otra de 10 bits con las consideraciones ya mencionadas.

Dicha conversión se realiza en dos etapas. En primer lugar una conversión de 5 bits (x_2x_1) a 6 bits y luego los 3 bits (x_0) restantes a 4 bits, completando así los 10 bits totales.

Estas conversiones se realizan por tabla[6], salvo algunas excepciones que ya se mostrarán más adelante.

En la Tabla 1 se presentan las conversiones 5b → 6b y en la Tabla 2 las conversiones 3b → 4b de los *Data Characters* (prestar atención que hay palabras que su conversión no admiten polaridad).

Data Character		RD-	RD+
D.00	00000	100111	011000
D.01	00001	011101	100010
D.02	00010	101101	010010
D.03	00011	110001	
D.04	00100	110101	001010
D.05	00101	101001	
D.06	00110	011001	
D.07	00111	111000	000111
D.08	01000	111001	000110
D.09	01001	100101	
D.10	01010	010101	
D.11	01011	110100	
D.12	01100	001101	
D.13	01101	101100	
D.14	01110	011100	
D.15	01111	010111	101000

Data Character		RD-	RD+
D.16	10000	011011	100100
D.17	10001	100011	
D.18	10010	010011	
D.19	10011	110010	
D.20	10100	001011	
D.21	10101	101010	
D.22	10110	011010	
D.23	10111	111010	000101
D.24	11000	110011	001100
D.25	11001	100110	
D.26	11010	010110	
D.27	11011	110110	001001
D.28	11100	001110	
D.29	11101	101110	010001
D.30	11110	011110	100001
D.31	11111	101011	010100

Tabla 1: Conversión 5b → 6b

Data Character		RD-	RD+
D.x.0	000	1011	0100
D.x.1	001	1001	
D.x.2	010	0101	
D.x.3	011	1100	0011
D.x.4	100	1101	0010
D.x.5	101	1010	
D.x.6	110	0110	
D.x.7	111	1110	0001
D.x.7*	111	0111	1000

Tabla 2: Conversión 3b → 4b

Por ejemplo, siguiendo esto se puede traducir con *RD-*

$$W_{8b} = D.10.3 = 01101010 \rightarrow \ W_{10b} = 0101011100,$$

o con *RD+*

$$W_{8b} = D.10.3 = 01101010 \rightarrow \ W_{10b} = 0101010011.$$

Hay excepciones que hay que tener en cuenta ya que hay ocasiones en las que no se cumple la condición de no haber más de 4 unos o 4 ceros consecutivos en un *Data Character*. Un caso podría ser el siguiente

$$W_{8b} = D.24.7 = 11111000 \rightarrow \ W_{10b} = 0011000001.$$

Esto sucede cuando se aplican las reglas estrictas, observando que aparecen 5 ceros seguidos rompiendo con la condición recién mencionada. Para estos casos se utiliza el último elemento de la Tabla 2 *D.x.7**, el cual modifica su conversión para salvar las siguientes singularidades:

- D.0.7 en *RD-* y *RD+*
- D.11.7 en *RD+*
- D.13.7 en *RD+*
- D.14.7 en *RD+*
- D.15.7 en *RD-* y *RD+*
- D.16.7 en *RD-* y *RD+*
- D.17.7 en *RD-*
- D.18.7 en *RD-*
- D.20.7 en *RD-*
- D.24.7 en *RD-* y *RD+*
- D.31.7 en *RD-* y *RD+*

De la misma manera todo este mecanismo de conversión se repite para los *Comma Character*. En este caso se muestra la Tabla 3, la cual es una tabla general ya que la metodología es la misma y los casos son menos

Comma Character		RD-	RD+
K.23.7	11110111	1110101000	0001010111
K.27.7	11111011	1101101000	0010010111
K.28.0	00011100	0011110100	1100001011
K.28.1	00111100	0011111001	1100000110
K.28.2	01011100	0011110101	1100001010
K.28.3	01111100	0011110101	1100001010
K.28.4	10011100	0011110010	1100001101
K.28.5	10111100	0011111010	1100000101
K.28.6	11011100	0011110110	1100001001
K.28.7	11111100	0011111000	1100000111
K.29.7	11111101	1011101000	0100010111
K.30.7	11111110	0111101000	1000010111

Tabla 3: Conversión *Comma Character*

3. Implementación

3.1. Codificación 8b $\rightarrow$ 10b

Tal como se describió, la conversión 8b $\rightarrow$ 10b se realiza casi automáticamente mediante tablas de conversión (memorias ROM). La problemática de no poder implementarlo en una sola memoria ROM directamente es que hay que considerar algunos casos importantes. Sin embargo, la mayoría de los casos, como lo son los *Comma Character*, se trabajan de manera directa con la salida de las memorias, y en el caso de un cambio de polaridad, se le intercala a esta nueva palabra un negador de bits.

En primer lugar se debe recordar que en el caso de los *Data Character* hay muchas palabras que no admiten codificación en ambas polaridades, es decir, la transformación es única independiente de *RD*. En estos casos simplemente se saltea la etapa negadora, tanto para la conversión de la etapa de 5b como la de 3b.

Y en segundo lugar, están los casos que se etiquetaron como *D.xx.7**, que son aquellos que superan la cantidad de ceros o unos consecutivos permitidos en la palabra convertida, en estos casos (que permiten polaridad) se cambia la conversión del *D.xx.7* por *D.xx.7** según la Tabla 2.

En la Figura 10 se sintetiza esta etapa, mostrando todas las consideraciones a tener en cuenta para cumplir con todos los requisitos del protocolo, y en el Apéndice A se encuentra el VHDL asociado. Las señales marcadas como *Unique_D* y *Unique_P* referencian los casos que no permiten polaridad, es decir, que no existe su versión negada como palabra posible.

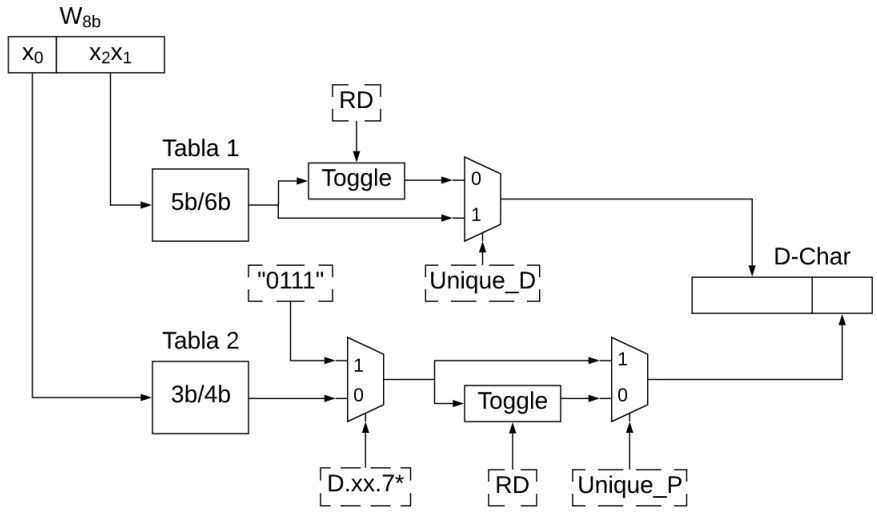

Figura 10: Implementación 8b → 10b en *Data Character*

Como se mencionó, en el caso de los *Comma Character*, no hay excepciones de polaridad ni de casos *D.xx.7**, por lo que, como se muestra en Figura 11, la conversión es directa.

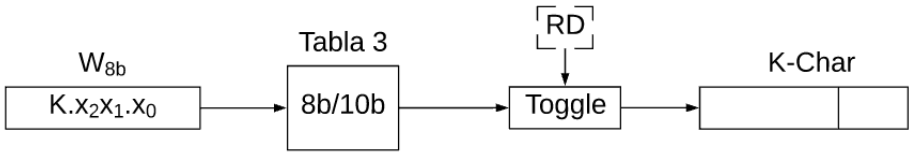

Figura 11: Implementación 8b → 10b en *Comma Character*

Como último, y es algo que se aplica tanto en *Data Character* como en *Comma Character*, se encuentra el control de polaridad, es decir, el control paralelo que se mantiene controlando el balance de unos y ceros que se van transmitiendo. Para resolver esto, al momento de cargar el *Buffer* de salida, se tienen disponibles ambas polaridades de la palabra a ser enviada y se decide en el momento cuál será conveniente despar, no solo para mantener la continua nula (acoplamiento alterna), sino también para no romper con la premisa de que no puede haber más de 5 unos o ceros consecutivos en ninguna instancia de la transmisión, caso que podría sucederse con el final y principio de dos palabras válidas consecutivas.

3.2. Decodificación 10b → 8b

Como es de esperar, la etapa de decodificación tiene las mismas consideraciones que la de codificación. La problemática de la polaridad se resuelve con el agregado de un bit extra en la tabla inversa 6b/5b. Este bit indica que la palabra entrante tiene polaridad inversa, pudiendo de aquí extraer información para la decodificación 4b/3b. Además de esto se compara la palabra entrante con la lista de palabras que no admiten polaridad (para no negar innecesariamente) y aquellas con el caso especial de *D.xx.7**. En la Figura 12 se esquematiza esta etapa y en el Apéndice B se encuentra el VHDL asociado.

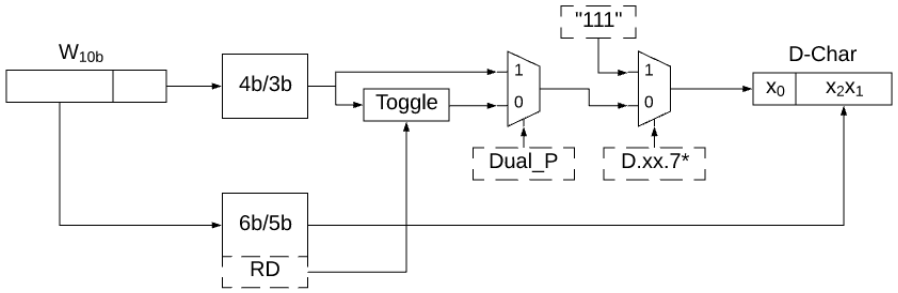

Figura 12: Implementación 10b → 8b en *Data Character*

Y al igual que en la codificación, los *Comma Character* son simplemente decodificados aplicando la tabla inversa y el bit de aviso de polaridad, como se muestra en la Figura 13

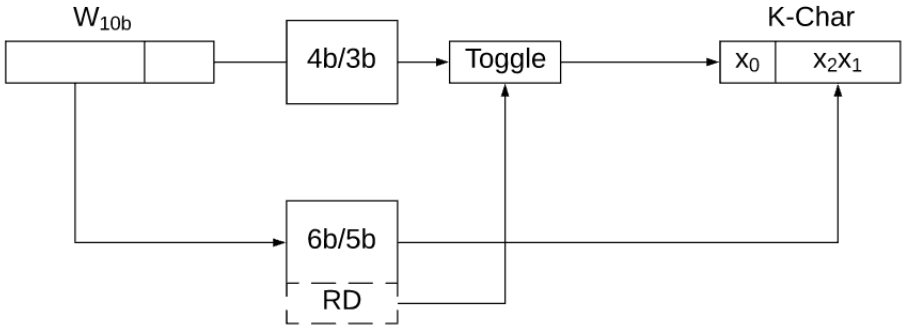

Figura 13: Implementación 10b → 8b en *Comma Character*

3.3. Transmisión LVDS

Volviendo a la idea principal de mejorar el control de errores en un canal LVDS, aquí se presenta dicha implementación.

De manera general, para codificar y enviar un byte se sabe que una vez obtenida la palabra de 8 bits codificada en una de 10 bits, se la carga en un Registro, y con un *Shift Register* se van desplazando los bits al ritmo del reloj de transmisión, ingresando el MSB a un último *Buffer* que convierte la señal Single-Ended a LVDS, como muestra la Figura 14. Este Registro se trabaja de manera cíclica, es decir que mientras se está despachando el LSB ya se está cargando la próxima W_{10b}, de manera que en el próximo CLK ya se esté enviando el MSB de la siguiente palabra.

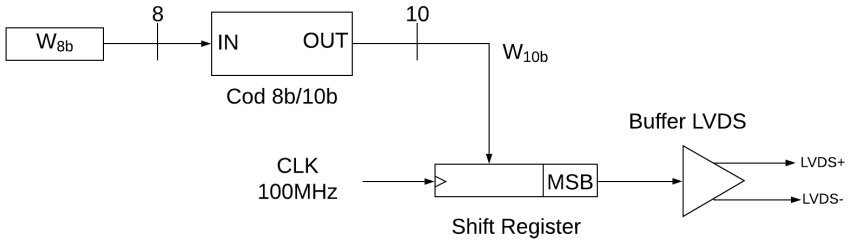

Figura 14: Diagrama transmisor LVDS

Ahora, de manera particular, en este trabajo se necesitan transferir 64 bits de información de cada evento localizado en cada cámara Gamma, caracterizado cada uno en cada placa de Procesamiento Planar, es decir, 8 W_{8b}, que una vez convertidas serán 8 W_{10b}, es decir, 80 bits. Si a esto se le suma un *Comma Character* al inicio como sincronización de tramas y un *CheckSum* al final, la trama total a transmitir tendrá 10 W_{10b}, y si se desea mantener la velocidad de transmisión de datos desde los Cabezales, el reloj de transmisión será de 100MHz para así permitir tener eventos cada 1μs.

A modo de primera implementación se plantearon dos *Comma Character* a modo de control de flujo

- K.28.5: se pensó a modo de comienzo de cada trama por posible desincronización en la máquina de transmisión
- K.28.2: cada 2 segundos hay un Reset general que se propaga por la totalidad de las 295 FPGAs a modo de recuperación de reloj en caso de algún tipo de asincronía. En estos casos, la primera trama luego de cada Reset que se envía por LVDS comenzará de esta manera, indicando la correcta recepción de dicha señal

Por último, es en esta etapa donde se realizó el balance de polaridad, es decir, una vez codificado cada byte, se realiza el cálculo de la polaridad que esta nueva palabra introduce en el canal con el fin de que siempre esté lo más cercana a cero posible, y así perturbar lo menos posible la estabilidad del canal. Este balance se realiza byte a byte y puede verse en el Apéndice C.

3.4. Recepción LVDS

Análogamente a la etapa de transmisión, el receptor también contará con un *Buffer* de entrada que convierte la señal LVDS en Single-Ended. La salida de éste ingresará en un *Shift Register* cíclico de 10 bits que se irá cargando desde el LSB al ritmo del CLK de 100MHz. En cada pulso de CLK el *Shift Register* es leido por el decodificador $10\text{b} \rightarrow 8\text{b}$ y su salida se compara constantemente esperando encontrar el *Comma Character* correspondiente de inicio de trama, mediante la señal *SYNC*. Un esquema general de esto puede visualizarse en la Figura 15.

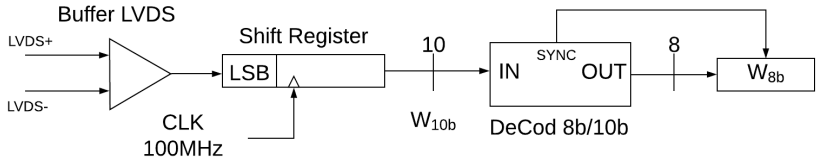

Figura 15: Diagrama receptor LVDS

Una vez sincronizado, se comienza con el llenado del Registro de datos final de 64 bits y posterior verificación de *CheckSum* para validarlo, que en caso de ser correcto es entregado a la próxima etapa para su correspondiente análisis.

Esta etapa de la implementación es más simple ya que tiene un comportamiento totalmente pasivo en lo que respecta a control de polaridad y generación de palabras. El código de la misma se encuentra en el Apéndice D.

4. Conclusión

Se ha podido implementar de manera satisfactoria una codificación 8b/10b sobre un canal LVDS con el fin de mejorar el control de errores del mismo y evitar así los falsos positivos.

En primera instancia se comprobó que la codificación cumplió con las condiciones temporales exigidas siguiendo los informes de síntesis del módulo aislado, obteniendo los siguientes resultados

- Minimum period: 7.797ns (Maximum Frequency: 128.263MHz)
- Minimum input arrival time before clock: 1.734ns
- Maximum output required time after clock: 1.661ns

Aprobada esta restricción se reemplazaron y adaptaron los tiempos del Firmware que trabajaba con Paridad simple. Esto hubo que hacerlo porque la totalidad de las máquinas de estado del manejo de datos funciona a 80MHz, y el hecho de ahora manejar las señales a 100MHz hizo tener especial cuidado en el manejo de los semáforos, los cuales se encargan de cargar en los registros de salida las nuevas tiras de datos a ser codificadas y enviadas del lado del transmisor, como así el aviso de llegada de nuevos datos decodificados del lado del receptor.

Una vez realizado esto se logró la misma versatilidad de funcionamiento, es decir, sin detectar diferencias considerables a la hora de realizar adquisiciones. Sin embargo, no se pudieron realizar comparaciones en lo que respecta a tasas de error ya que no se pudo someter al sistema a condiciones para forzar las fallas. Debido a que nunca se había puesto énfasis en el sistema de paridad, el canal LVDS en cuestión fue diseñado y cuidado de tal manera de mantener una muy buena relación señal a ruido, manteniéndose muy estable en condiciones normales de funcionamiento. Como trabajo futuro queda someter al equipo a un campo de interferencias, tanto artificiales como naturales de funcionamiento con alta actividad de eventos, para analizar las diferencias de ambas implementaciones en condiciones más ruidosas.

Respecto a la diferencia en recursos utilizados, cabe destacar que el algoritmo de síntesis trabajó mejor con la solución de codificación dado su mayor nivel de orden, es decir, que al ser más modularizada su implementación fue más simple la distribución los recursos. El caso de Paridad se trataba de una sola máquina de estados recursiva y sistemática que no permitía opción de resumen. Tanto en la Figura 16 como en la Figura 17 se pueden ver el resumen de utilización y consumo relativo en el diseño con Paridad y con codificación 8b10b respectivamente.

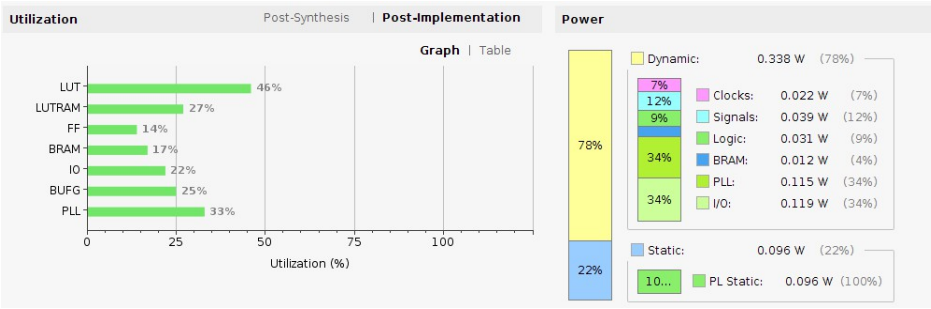

Figura 16: Recursos utilizados modelo con paridad

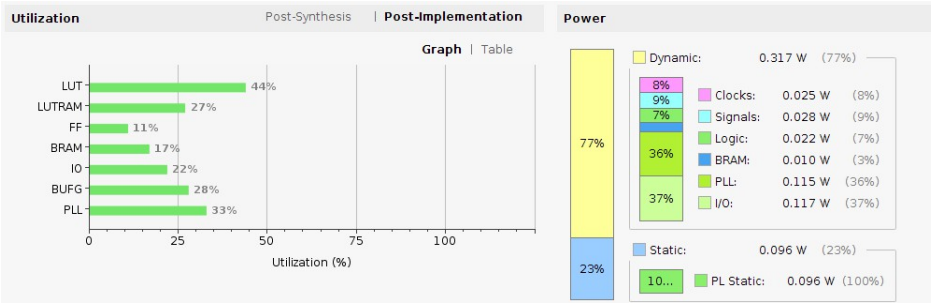

Figura 17: Recursos utilizados modelo con codificación 8b10b

A nivel personal, los conocimientos adquiridos en el desarrollo de este trabajo como en la totalidad del curso han superado mis expectativas y me ha hecho crecer profesionalmente en la generación y optimización de código.

Referencias

[1] *https://shop.trenz-electronic.de/en/TE0712-02-100-2C-Artix-7-100T-Micromodule-with-Xilinx-XC7A100T-2C-Size-4-x-5-cm-com.temp.range.*

[2] *https://www.researchgate.net/publication/267559533.*

[3] *https://es.wikipedia.org/wiki/Aniquilaci %C3 %B3n_electr %C3 %B3n-positr %C3 %B3n.*

[4] *https://www.anmm.org.mx/GMM/2008/n2/58_vol_144_n2.pdf.*

[5] *https://en.wikipedia.org/wiki/Manchester_code.*

[6] *https://es.wikipedia.org/wiki/8b/10b.*

Apéndices

A. VHDL Codificador

```
begin

Dato <= Dato_In(4 downto 0);
Plus <= Dato_In(7 downto 5);

Ins_DCh   : DCh   PORT MAP(a => Dato, spo => Out_DatoD);
Ins_Dplus : Dplus PORT MAP(a => Plus, spo => Out_PlusD);
Ins_KCh   : KCh   PORT MAP(a => Dato, spo => Out_DatoK);
Ins_Kplus : Kplus PORT MAP(a => Plus, spo => Out_PlusK);

Toggle6 <= "111111" when nRD = '1' else "000000";
Toggle4 <=  "1111"  when nRD = '1' else  "0000";

Dxx_7_Case <= DnK when
      Dato_In = "11100000" or                -- D.0.7 - y +
     (Dato_In = "11101011" and nRD = '1') or -- D.11.7 solo +
     (Dato_In = "11101101" and nRD = '1') or -- D.13.7 solo +
     (Dato_In = "11101110" and nRD = '1') or -- D.14.7 solo +
      Dato_In = "11101111" or                -- D.15.7 - y +
      Dato_In = "11110000" or                -- D.16.7 - y +
     (Dato_In = "11110001" and nRD = '0') or -- D.17.7 solo -
     (Dato_In = "11110010" and nRD = '0') or -- D.18.7 solo -
     (Dato_In = "11110100" and nRD = '0') or -- D.20.7 solo -
      Dato_In = "11111000" or                -- D.24.7 - y +
      Dato_In = "11111111"                   -- D.31.7 - y +
  else '0';

Unique_DatoD  <= '1' when
     (Dato = "00011" or Dato = "00101" or
      Dato = "00110" or Dato = "01001" or
      Dato = "01010" or Dato = "01011" or
      Dato = "01100" or Dato = "01101" or
      Dato = "01110" or Dato = "10001" or
      Dato = "10010" or Dato = "10011" or
      Dato = "10100" or Dato = "10101" or
      Dato = "10110" or Dato = "11001" or
      Dato = "11010" or Dato = "11100") and DnK = '1'
  else '0';
```

```
Unique_PlusD  <= '1' when (Plus = "001" or Plus = "010" or
                           Plus = "101" or Plus = "110") and DnK = '1'
                    else '0';

Out_DatoD_Aux <= Out_DatoD XOR Toggle6 when Unique_DatoD = '0' else Out_DatoD;

Out_PlusD_Aux <=    "0111" XOR Toggle4 when Dxx_7_Case = '1'
            else Out_PlusD XOR Toggle4 when Unique_PlusD = '0'
            else Out_PlusD;

Out_DatoK_Aux <= Out_DatoK XOR Toggle6;
Out_PlusK_Aux <= Out_PlusK XOR Toggle4;

Dato_Out <= Out_DatoD_Aux & Out_PlusD_Aux when DnK = '1' else
            Out_DatoK_Aux & Out_PlusK_Aux;

Error <= '1' when Out_DatoD = "00000" or Out_DatoK = "00000"
             else '0';

end Arq_Cod_8b10b;
```

B. VHDL Decodificador

```
begin

Dato <= Dato_In(9 downto 4);
Plus <= Dato_In(3 downto 0);

Dual_D <= '1' when
    Dato = "001011" or Dato = "001101" or
    Dato = "001110" or Dato = "010011" or
    Dato = "010101" or Dato = "010110" or
    Dato = "011001" or Dato = "011010" or
    Dato = "011100" or Dato = "100011" or
    Dato = "100101" or Dato = "100110" or
    Dato = "101001" or Dato = "101010" or
    Dato = "101100" or Dato = "110001" or
    Dato = "110010" or Dato = "110100"
    else '0';

Dual_Plus <= '1' when
    Plus = "0101" or Plus = "0110" or
    Plus = "1001" or Plus = "1010"
    else '0';

Ins_DCh : Inv_Ch PORT MAP(a => Dato, spo => Out_DatoD);

nRD_Aux_D <= Out_DatoD(Out_DatoD'high) when Dual_D = '0' else '0';

Toggle4   <= "1111" when nRD_Aux_D = '1' else "0000";

Plus_D    <= Plus XOR Toggle4 when Dual_Plus = '0' else Plus;
Plus_K    <= Plus XOR Toggle4;

Ins_Dplus : Inv_Dplus PORT MAP(a => Plus_D, spo => Out_PlusD);
Ins_Kplus : Inv_Kplus PORT MAP(a => Plus_K, spo => Out_PlusK);

Dxx_7_Case <= '1' when Out_PlusD = "1101" or Out_PlusD = "1001" else '0';

nRD_Aux_P  <= '1' when Out_PlusD = "1001" else Out_PlusD(Out_PlusD'high);

Dato_Out_Aux_D <= "111" & Out_DatoD(Out_DatoD'high-1 downto 0) when Dxx_7_Case = '1'
    else Out_PlusD(2 downto 0) & Out_DatoD(Out_DatoD'high-1 downto 0);

Dato_Out_Aux_K <= Out_PlusK(2 downto 0) & Out_DatoD(Out_DatoD'high-1 downto 0);
```

```
DnK_Aux <= '0' when
-- K.28.0 000 11100
(Dato_Out_Aux_K = x"1C" and (Dato_In = "00"&x"F4" or Dato_In = not ("00"&x"F4"))) or
-- K.28.1 001 11100
(Dato_Out_Aux_K = x"3C" and (Dato_In = "00"&x"F9" or Dato_In = not ("00"&x"F9"))) or
-- K.28.2 010 11100
(Dato_Out_Aux_K = x"5C" and (Dato_In = "00"&x"F5" or Dato_In = not ("00"&x"F5"))) or
-- K.28.3 011 11100
(Dato_Out_Aux_K = x"7C" and (Dato_In = "00"&x"F3" or Dato_In = not ("00"&x"F3"))) or
-- K.28.4 100 11100
(Dato_Out_Aux_K = x"9C" and (Dato_In = "00"&x"F2" or Dato_In = not ("00"&x"F2"))) or
-- K.28.5 101 11100
(Dato_Out_Aux_K = x"BC" and (Dato_In = "00"&x"FA" or Dato_In = not ("00"&x"FA"))) or
-- K.28.6 110 11100
(Dato_Out_Aux_K = x"DC" and (Dato_In = "00"&x"F6" or Dato_In = not ("00"&x"F6"))) or
-- K.28.7 111 11100
(Dato_Out_Aux_K = x"FC" and (Dato_In = "00"&x"F8" or Dato_In = not ("00"&x"F8"))) or
-- K.23.7 111 10111
(Dato_Out_Aux_K = x"F7" and (Dato_In = "11"&x"A8" or Dato_In = not ("11"&x"A8"))) or
-- K.27.7 111 11011
(Dato_Out_Aux_K = x"FB" and (Dato_In = "11"&x"68" or Dato_In = not ("11"&x"68"))) or
-- K.29.7 111 11101
(Dato_Out_Aux_K = x"FD" and (Dato_In = "10"&x"E8" or Dato_In = not ("10"&x"E8"))) or
-- K.30.7 111 11110
(Dato_Out_Aux_K = x"FE" and (Dato_In = "01"&x"E8" or Dato_In = not ("01"&x"E8")))
else '1';

Dato_Out <= Dato_Out_Aux_K when DnK_Aux = '0' else Dato_Out_Aux_D;

DnK      <= DnK_Aux;
nRD      <= nRD_Aux_D or nRD_Aux_P;

Error_D1 <= '1' when Out_DatoD = "010101" else '0';
Error_D2 <= '1' when Out_PlusD = "1010" else '0';

Error_D  <= (Error_D1 or Error_D2) when DnK_Aux = '1' else '0';
Error_K  <= '1' when Out_PlusK = "1000" and DnK_Aux = '0' else '0';

Error <= error_D or error_K;

end Arq_Dec_8b10b;
```

C. VHDL LVDS TX

```
begin

Ins_Cod_Menos: entity work.En_Cod_8b10b(Arq_Cod_8b10b)
port map(
    Dato_In  => Cod_8b_In,
    DnK      => DnK_Cod,
    nRD      => '0',
    Dato_Out => Cod_10b_Out_Menos,
    Error    => open
);

Ins_Cod_Mas: entity work.En_Cod_8b10b(Arq_Cod_8b10b)
port map(
    Dato_In  => Cod_8b_In,
    DnK      => DnK_Cod,
    nRD      => '1',
    Dato_Out => Cod_10b_Out_Mas,
    Error    => open
);

LVDS_Out: process (Clk)
begin
if rising_edge (Clk) then
    if ResetTX = '1' then
        DnK_Cod     <= '0';
        Cod_8b_In   <= "10111100"; -- K.28.5
        Buff_10b    <= Cod_10b_Out_Menos; -- K.28.5
        Index       <= 9;
        Count       <= 0;
        Next_Data   <= '0';
        SumCheck    <= CONV_UNSIGNED(0,SumCheck'length);
        Polarity    <= 2; -- K.28.5 5 unos - 3 ceros
    else
        Next_Data   <= '0';
        LVDSout     <= Buff_10b(Index);
        Index       <= Index - 1;
        if index = 1 then
            Count   <= Count + 1;
            if Count = 0 then
                DnK_Cod     <= '0';
                Cod_8b_In   <= "10111100"; -- K.28.5
                Buff_Data   <= Data_A;
                SumCheck    <= CONV_UNSIGNED(0,SumCheck'length);
```

```
        elsif Count = 9 then
            DnK_Cod     <= '1';
            Cod_8b_In   <= CONV_STD_LOGIC_VECTOR(SumCheck,SumCheck'length);
            Next_Data   <= '1';
            Count       <= 0;
        else
            DnK_Cod     <= '1';
            Cod_8b_In   <= Buff_Data(Buff_Data'high - (Count-1)*8 downto
                                     Buff_Data'high - (Count-1)*8 - 7);
            SumCheck    <= SumCheck +
                           unsigned(Buff_Data(Buff_Data'high - (Count-1)*8 downto
                                    Buff_Data'high - (Count-1)*8 - 7));
        end if;
    elsif index = 0 then
        if Polarity + (CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(9),1)) +
         CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(8),1)) +
         CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(7),1)) +
         CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(6),1)) +
         CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(5),1)) +
         CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(4),1)) +
         CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(3),1)) +
         CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(2),1)) +
         CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(1),1)) +
         CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(0),1)))*2 - 10 < 0 then
            if Buff_10b(2 downto 0) = Cod_10b_Out_Menos(Cod_10b_Out_Menos'high
                                      downto Cod_10b_Out_Menos'high - 2) then
                Polarity <= Polarity +
                   (CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(9),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(8),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(7),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(6),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(5),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(4),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(3),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(2),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(1),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(0),1)))*2 - 10;
                Buff_10b <= Cod_10b_Out_Mas;
            else
                Polarity <= Polarity +
                   (CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(9),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(8),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(7),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(6),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(5),1)) +
                    CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(4),1)) +
```

```
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(3),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(2),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(1),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(0),1)))*2 - 10;
                    Buff_10b <= Cod_10b_Out_Menos;
                end if;
            else
                if Buff_10b(2 downto 0) = Cod_10b_Out_Mas(Cod_10b_Out_Mas'high
                                          downto Cod_10b_Out_Mas'high - 2) then
                    Polarity <= Polarity +
                       (CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(9),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(8),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(7),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(6),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(5),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(4),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(3),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(2),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(1),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Menos(0),1)))*2 - 10;
                    Buff_10b <= Cod_10b_Out_Menos;
                else
                    Polarity <= Polarity +
                       (CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(9),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(8),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(7),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(6),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(5),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(4),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(3),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(2),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(1),1)) +
                        CONV_INTEGER(CONV_UNSIGNED(Cod_10b_Out_Mas(0),1)))*2 - 10;
                    Buff_10b <= Cod_10b_Out_Mas;
                end if;
            end if;
            Index                <= 9;
        end if;
    end if;
end if;
end process;

end Arq_LVDS_TX;
```

D. VHDL LVDS RX

```
begin

Dec_10b_In  <= Dec_10b_In(8 downto 0) & LVDS_IN when rising_edge(CLK);

SeisCeros   <= '1' when
    Dec_10b_In(Dec_10b_In'high downto Dec_10b_In'high - 5) = "000000"
    else '0';

SeisUnos    <= '1' when
    Dec_10b_In(Dec_10b_In'high downto Dec_10b_In'high - 5) = "111111"
    else '0';

Ins_Dec: entity work.En_Dec_8b10b(Arq_Dec_8b10b)
port map(
    Dato_In          => Dec_10b_In,
    DnK              => DnK_Dec,
    nRD              => open,
    Dato_Out         => Dec_8b_Out,
    Error            => Error_Dec
);

LVDS_Acquire: process (Clk)
begin
if rising_edge (Clk) then
    if RST = '1' then
        Init        <= '0';
        Count_bit   <= 1;
        Count_byte  <= 1;
        SumCheck    <= to_unsigned(0,SumCheck'length);
    else
        Ready <= '0';
        if Init = '0' and Dec_8b_Out = "10111100" and DnK_Dec = '0' and
           Error_Dec = '0' then
            Init        <= '1';
            Count_bit   <= 1;
            Count_byte  <= 1;
            SumCheck    <= to_unsigned(0,SumCheck'length);
        end if;

        if Init = '1' then
            if Count_bit = 8 and Count_Byte = 10 then
                Init    <= '0';
            elsif Count_bit = 10 then
```

```
                Count_bit   <= 1;
                Count_byte  <= Count_byte + 1;
                if Count_byte = 9  then
                    if SumCheck = unsigned(Dec_8b_Out) then
                        Data_Out    <= sData_Out;
                        Ready       <= '1';
                    end if;
                else
                    if DnK_Dec = '1' and Error_Dec = '0' then
                        sData_Out   <= sData_Out(sData_Out'high - 8 downto 0)
                                       & Dec_8b_Out;
                    else
                        Init        <= '0';
                    end if;
                    SumCheck    <= SumCheck + unsigned(Dec_8b_Out);
                end if;
            else
                Count_bit   <= Count_bit + 1;
            end if;
        end if;
    end if;
end if;
end process;

end Arq_LVDS_RX;
```