

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

Mayo 2018

Índice

1. Motivación y resumen	2
2. Introducción	4
2.1. FPGA	4
2.2. Procesamiento Planar	4
2.3. Coincidencias	5
2.4. Estado del arte	8
2.5. Codificación 8b10b	8
3. Implementación	13
3.1. Codificación 8b → 10b	13
3.2. Decodificación 10b → 8b	15
3.3. Transmisión LVDS	16
3.4. Recepción LVDS	16
4. Conclusión	17
Apéndices	18
Apéndice A. VHDL Codificador	18
Apéndice B. VHDL Decodificador	20
Apéndice C. VHDL LVDS TX	22
Apéndice D. VHDL LVDS RX	25

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 confinaran en Líneas de respuesta para una posterior reconstrucción y formado de imágenes diagnósticas.

Como ya veremos en el desarrollo del informe, el canal crítico de transferencia de información es el conjunto de líneas LVDS por donde los cabezales transfieren los eventos detectados hacia el procesador de coincidencias. Cada cabezal tiene la capacidad de procesar 1Mevento/s, y dado que el procesamiento es *On The Fly*, 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 lo estamos 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 procedo a presentar los bloques principales donde se planteó la problemática con el fin de crear una base de conocimientos del escenario.

2.1. FPGA

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].



Figura 3: SOM Artix-7

Fue largo el camino recorrido en el desarrollo del equipo hasta llegar a utilizar este tipo de tecnologías, pero las dificultades de la soldadura BGA en PCB de gran tamaño y los problemas de ESD introducidos por las grandes longitudes de cables entre procesadores , nos llevaron a dejarle el diseño o optimización delicada a la gente capaz de hacerlo, y nosotros poder dejar de pensar en los problemas y dedicarnos a las soluciones que exigía el equipo.

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 PMT (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 Angger) 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.

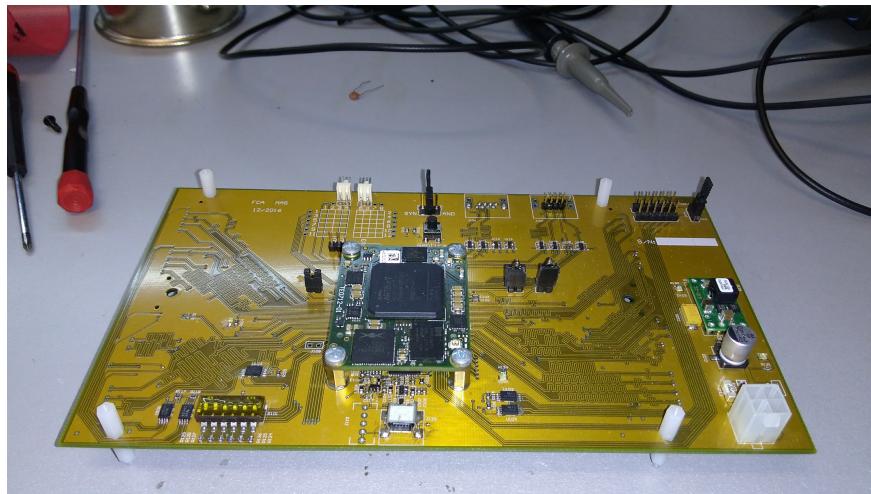


Figura 4: Procesamiento Planar

2.3. Coincidencias

La necesidad de trabajar con Positrones en un PET se concentra en que estas partículas, al interaccionar con un electrón, se aniquilan generando dos fotones de 511keV desfasados 180°, pudiendo estos ser detectados por dos cabezales enfrentados.

La unión de los dos puntos de interacción con los cabezales (coordenadas XY de cada uno) generan lo denominado LOR (Line Of Response), que con el correr del estudio se comienzan a acumular, logrando intensificaciones en aquellas zonas donde la actividad de positrones es más marcada, pudiendo así obtener las imágenes buscadas.

Al paciente en cuestión 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

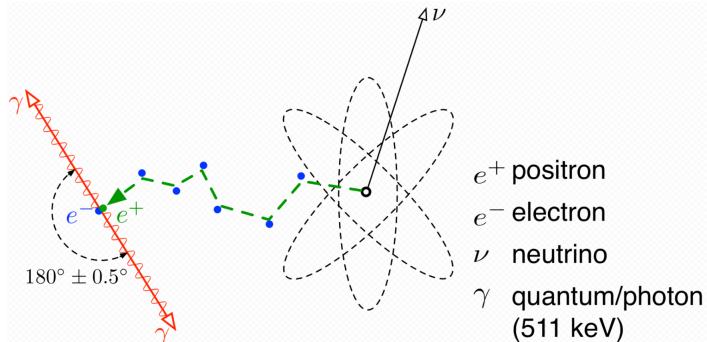


Figura 5: Aniquilación Positrón-Electrón

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, hígado y tumores celularmente activos. De esta manera, las desintegraciones radiactivas saldrán de estos lugares específicos del cuerpo, pudiendo así lograr su ubicación.



Figura 6: Coincidencias



Figura 7: 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 hubiésemos llegado a nada: sincronización.

Los pulsos que entregan los PMT, debido a la respuesta del centellador, es de unos 120ns, la ventana temporal intracabezal para determinar de que la información de 2 PMTs contiguos pertenecen al mismo evento es aproximadamente 1ns, y la ventana temporal intercabezal para determinar las coincidencias y así validar las LORs está alrededor de los 10ns. Es por esto que se trabaja con un reloj maestro de 20MHz, generado por la placa de Coincidencias, con el cual se sincronizan las 6 Artix-7 de Procesamiento Planar, y por lo tanto las 48 Spartan-3 de los PMT de cada uno de los cabezales, es decir 295 FPGAs perfectamente sincronizadas.

Con la finalidad de poder procesar 1Mevento/s se planteó que la descarga desde cada cabezal hacia Coincidencias debe ser de $1\mu s$, 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.

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 es la elegida ya que su incorporación de transiciones facilita la recuperación de reloj y no sobrecarga la línea de información extra. Lo que si aumenta es 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 ayudan 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), así como también incorporar palabras fuera del rango original, es decir, en 8b solo se podían enviar 256 posibles palabras, ahora con 10b, no solo se pueden enviar esas 256 posibilidades en ambas polaridades, sino que también palabras nuevas 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 de nuevas palabras 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.

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).

Ahora bien, se está pasando de un sistema de 256 valores a uno de 1024 valores posibles, esto da un exceso de 768 palabras sin uso. Para esto en primer lugar se añaden a las posibilidades los *Comma Character*, que son 12 palabras de 10 bits reservadas para comportamientos especiales en el receptor que con solo 8 bits no se podía.

Es decir que ahora sobran 756 palabras. Y aquí es donde se busca optimizar el canal de comunicación, utilizando aquellas palabras desbalanceadas (6/4 o 4/6) para optimizar la continua resultante en la línea de transmisión.

Agregadas todas estas consideraciones, 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. Supongamos que tenemos una palabra inicial conformada de la siguiente manera

$$W_{8b} = b_7b_6b_5b_4b_3b_2b_1b_0,$$

y definamos los *Data Character* (caracteres comunes) de la siguiente manera

$$D.x_2x_1.x_0$$

donde

$$[x_2x_1]_{10} = [b_4b_3b_2b_1b_0]_2$$

y

$$[x_0]_{10} = [b_7b_6b_5]_2.$$

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 nos 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 realizará 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 [2], salvo algunas excepciones que ya se presentarán.

En primer lugar se presentan las conversiones de los *Data Characters*. A continuación se muestra la tabla de la etapa 5b → 6b (prestar atención que hay palabras que su conversión no admite polaridad)

Data Character		RD-	RD+		Data Character		RD-	RD+
D.00	00000	100111	011000		D.16	10000	011011	100100
D.01	00001	011101	100010		D.17	10001		100011
D.02	00010	101101	010010		D.18	10010		010011
D.03	00011		110001		D.19	10011		110010
D.04	00100	110101	001010		D.20	10100		001011
D.05	00101		101001		D.21	10101		101010
D.06	00110		011001		D.22	10110		011010
D.07	00111	111000	000111		D.23	10111	111010	000101
D.08	01000	111001	000110		D.24	11000	110011	001100
D.09	01001		100101		D.25	11001		100110
D.10	01010		010101		D.26	11010		010110
D.11	01011		110100		D.27	11011	110110	001001
D.12	01100		001101		D.28	11100		001110
D.13	01101		101100		D.29	11101	101110	010001
D.14	01110		011100		D.30	11110	011110	100001
D.15	01111	010111	101000		D.31	11111	101011	010100

Tabla 1: Conversión 5b → 6b

Del mismo modo se plantea la tabla de la etapa 3b → 4b

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 podemos traducir con *RD-*

$$W_{8b} = D.11.3 = 01101011 \rightarrow W_{10b} = 1101001100,$$

o con *RD+*

$$W_{8b} = D.11.3 = 01101011 \rightarrow W_{10b} = 1101000011.$$

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

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

Si se convierte mecánicamente, se observa 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 posibilidades:

- 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	11000001011
K.28.1	00111100	0011111001	1100000110
K.28.2	01011100	0011110101	11000001010
K.28.3	01111100	0011110101	11000001010
K.28.4	10011100	0011110010	11000001101
K.28.5	10111100	0011111010	11000000101
K.28.6	11011100	0011110110	11000001001
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 → 10b

Tal como se describió, la conversión 8b → 10b se realiza casi automáticamente mediante tablas de conversión (memorias ROM). La problemática de no poder implementarlo en una 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 automática con la salida de las ROMs, y en el caso de un cambio de polaridad, se le intercala a esta nueva palabra un Negador de bit's.

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 los 5b como la de los 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 8 que sintetiza esta etapa y en el Apéndice A se encuentra el VHDL asociado.

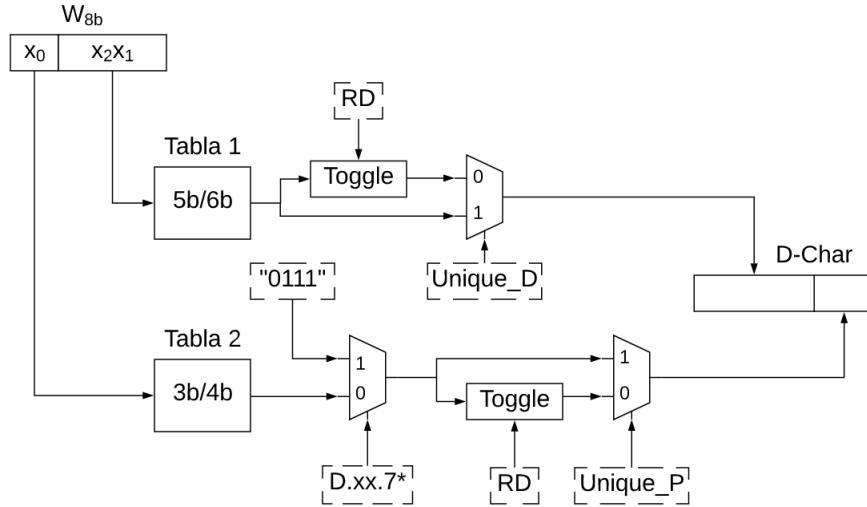


Figura 8: Implementación 8b → 10b en *Data Character*

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 8, la conversión es directa.

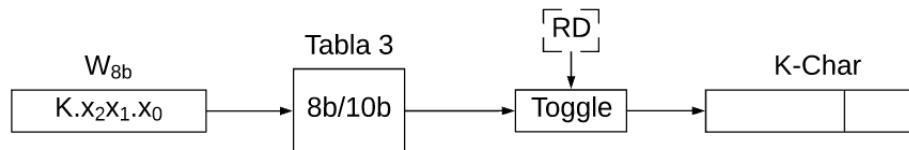


Figura 9: Implementación 8b → 10b en *Comma Character*

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 despachada, 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 consecutivas.

3.2. Decodificación 10b → 8b

Como es de esperar, la etapa de decodificación tiene las mismas consideraciones que la 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 10 se esquematiza esta etapa y en el Apéndice B se encuentra el VHDL asociado.

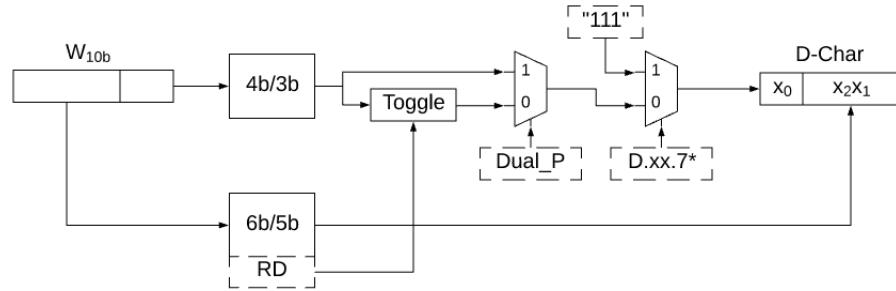


Figura 10: Implementación 10b → 8b en *Data Character*

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 11

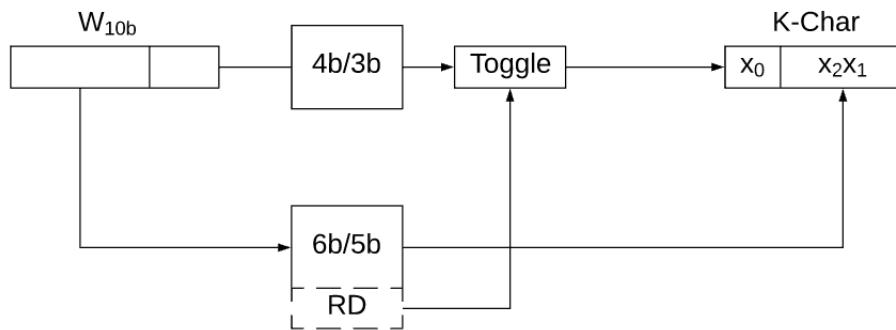


Figura 11: Implementación 10b → 8b en *Comma Character*

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. 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.

Ahora, de manera particular, en este trabajo se necesitan transferir 64 bits de información de cada evento localizado de cada cámara Gamma caracterizado 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 $1\mu s$.

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 en la que 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 este ingresara 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 Registro es ingresado al decodificador $10b \rightarrow 8b$ y su salida se compara constantemente esperando encontrar el *Comma Character* correspondiente de inicio de trama.

Una vez sincronizado, comienza con el llenado del Registro de datos final de 64 bits y posterior verificación de CheckSum para validarla, 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

Datos de color:

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

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://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);
            end if;
        end if;
    end if;
end if;
end process;

```

```

        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 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_Byt

```

```
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 process;

end Arq_LVDS_RX;
```