UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DESARROLLO DE CONTROLADOR PARA INTERFAZ DE VIDEO DE ALTA DEFINICIÓN EN FPGA

INGENIERÍA DE TELECOMUNICACIÓN

JUAN DAVID HEREDIA GRANADOS

MÁLAGA, 2017

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

Titulación: Ingeniería de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por:

D./Dª. _____

D./Dª. _____

D./Dª. _____

para juzgar el Proyecto Fin de Carrera titulado:

DESARROLLO DE CONTROLADOR PARA INTERFAZ DE VIDEO DE ALTA DEFINICIÓN EN FPGA

del alumno/a D./Dª. *Juan David Heredia Granados*

dirigido por D./Dª. *Martín González García*

ACORDÓ POR _____ OTORGAR LA

CALIFICACIÓN DE _____

Y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia

Málaga, a _____ de _____ de _____

El/La Presidente/a El/La Vocal El/La Secretario/a

Fdo.: _____ Fdo.: _____ Fdo.: _____

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

DESARROLLO DE CONTROLADOR PARA INTERFAZ DE VIDEO DE ALTA DEFINICIÓN EN FPGA

REALIZADO POR:

Juan David Heredia Granados

DIRIGIDO POR:

Martín González García

DEPARTAMENTO DE: *Tecnología Electrónica*

TITULACIÓN: Ingeniería de Telecomunicación

PALABRAS CLAVE: *Controlador, video, alta definición, Vivado, Zynq, VHDL, FPGA, HDMI, TMDS, banco de pruebas*

RESUMEN: En este proyecto se diseña, implementa y se verifica en placa un *core* sintetizable para el control del *Interfaz Multimedia de Alta Definición* (HDMI), particularmente la salida de video, embebido en el dispositivo reconfigurable APSoC (*All Programmable System on Chip*) Zynq™-7000 presente en la placa de desarrollo Digilent ZYBO. El desarrollo se realiza a nivel RTL (*Nivel de Transferencia de Registro*) utilizando el lenguaje de descripción hardware VHDL sobre el entorno de desarrollo Vivado®.

Málaga*, noviembre de 2017*

Agradecimientos

En primer lugar, agradecer a mis padres, Juan y Charo, su apoyo durante todos estos años, por haberme dado la posibilidad de realizar esta Ingeniería y por depositar en mi toda su confianza en este y en todos mis proyectos.

Agradecer a mis hermanos, Samuel e Ismael, los ánimos sin los cuales nada de esto hubiera sido posible. "Siempre juntos, siempre fuertes".

Agradecer también a mi tutor en este Proyecto, Martín González García, el interés prestado y por facilitar el soporte necesario para su conclusión.

Y en resumen, gracias a todas esas personas que de una forma u otra me han aportado su granito de arena durante estos años.

A mi familia,

por su apoyo incondicional desde el inicio...

Índice general

Lista de Acrónimos

APSoC	All Programmable System on Chip
BNC	Bayonet Neill-Concelman
CLK	Clock
DDR	Double Data Rate
DEN	Data Enabled
DVI	Digital Visual Interface
FPGA	Field Programmable Gate Array
HDMI	High-Definition Multimedia Interface
HD-SDI	Serial Digital Interface
HDTV	High Definition Television
LSB	Least Significant Bit
MMCM	Mixed-Mode Clock Manager
MSB	Most Significant Bit
NTSC	National Television System Committee
PAL	Phase Alternating Line
PL	Programmable Logic
PLL	Phase-Locked Loop

PS	Processing System
RGB	Red-Green-Blue
RTL	Register-Transfer Level
SDI	Serial Digital Interface
SECAM	Séquentiel Couleur Avec Mémoire
TERC4	Transition Minimized Differential Signaling Error Reduction Coding
TMDS	Transition Minimized Differential Signaling
VESA	Video Electronics Standards Association
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
WHUXGA	Wide Hex Ultra Extended Graphics Array

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción

1.1. Contexto histórico y tecnológico

En la actualidad, los sistemas de visión artificial han adquirido una gran importancia en el campo científico-técnico debido a sus posibilidades de aplicación. Podemos encontrar sistemas de visión artificial en la automoción, la industria del cine, aplicaciones de alta velocidad, aplicaciones de tráfico, la biomecánica, la bioingeniería, etc.

El desarrollo de estos sistemas de visión artificial, no habría sido posible sin la evolución que se ha producido en el campo de la transferencia de la señal de video. Desde sus inicios, los sistemas de video han estado limitados en resolución y frecuencia, los ya antiguos sistemas analógicos pusieron de manifiesto la necesidad de dar un paso hacia adelante en la búsqueda de un interfaz digital de altas prestaciones que fuera adaptable y con capacidad de evolucionar.

1.1.1. Evolución del interfaz de vídeo

Previamente al análisis del *Interfaz Multimedia de Alta Definición (HDMI)*, es necesario un repaso básico a la evolución de los interfaces de vídeo:

Interfaces de video analógicos

1. **Video Compuesto**: Interfaz de video que codifica mediante los estándares NTSC, PAL o SECAM, la señal de video diferenciando las siguientes componentes:

   - Crominancia: Información sobre el color.
   - Luminancia: Información sobre la luz.
   - Sincronismo: Información sobre el barrido de la imagen.

   La transmisión se realiza sobre un cable coaxial de 75 Ohmios con conectores RCA, normalmente amarillos para diferenciar el canal de video de los canales de audio (ver Figura 1.1).

   La máxima resolución de pantalla que nos permite esta tecnología viene determinada por una resolución horizontal de 400 píxeles y 525 líneas verticales, siendo necesario un ancho de banda de 4MHz para conseguir una frecuencia de barrido de 30Hz.

Figura 1.1. Conector de Video Compuesto.

2. **Separated Video (S-Video)**: Interfaz de video con el mismo principio de funcionamiento que el interfaz de video compuesto, consigue una mejora en la calidad de la señal separando la información de crominancia y luminancia en dos pares señal-tierra sincronizados (ver Figura 1.2), lo que permite resoluciones de 720

píxeles por 480 líneas si se utiliza NTSC o 720 píxeles por 576 líneas según el estándar PAL y SECAM.

Figura 1.2. Conector de S-Video.

3. **Video por Componentes**: Interfaz de video con el mismo principio de funcionamiento que las dos anteriores, consigue una mejora en la calidad de la señal separando la información en tres pares, uno para la luminancia y dos para la crominancia o una descomposición de la señal en las componentes RGB (Red, Green, Blue) sincronizadas (ver Figura 1.3), consiguiendo resoluciones por encima del HDTV, siendo esta resolución de 1920 píxeles por 1080 líneas.

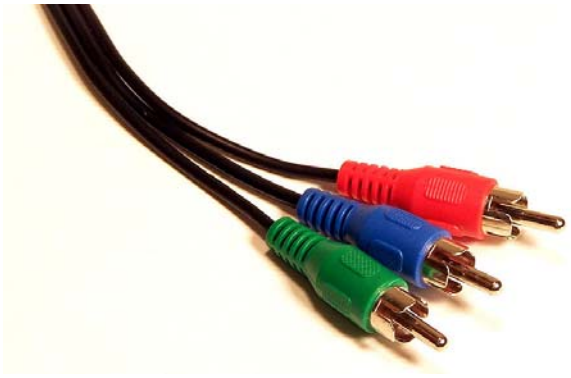


Figura 1.3. Conectores de Video por Componentes.

4. **Video Graphics Array (VGA)**: Interfaz de video basado en la transferencia de señal de video por componentes RGB dedicando además canales particulares para las componentes de sincronismo, consiguiendo así una mejora en la calidad de la señal. Se utiliza el conector D-Subminiature de 15 pines (ver Figura 1.4) y aunque en su diseño original la resolución máxima

era de 800 píxeles por 600 filas, 4 bits de color por píxel y hasta 70Hz de frecuencia de refresco, ha ido evolucionando hasta resoluciones de 7680 píxeles por 4800 líneas (WHUXGA - Wide Hex Ultra Extended Graphics Array).

Figura 1.4. Conector D-Subminiature (VGA).

Interfaces de video digital:

1. **Serial Digital Interface (SDI)**: Interfaz de video de uso profesional, existen dos versiones single-link y dual-link según se utilicen uno o dos enlaces. Su principal característica es la transmisión de video digital sin comprimir en serie a través de cable coaxial con conectores BNC estándar (ver Figura 1.5). Su versión original soportaba resoluciones de hasta 565p mientras que su versión HD-SDI permite la transmisión de señales de video de hasta 1080p.

Figura 1.5. Conectores BNC (SDI).

2. **FireWire**: Interfaz para transferencia de datos serie a gran velocidad desarrollado por Apple (ver Figura 1.6). Según su versión dispone de anchos de banda de entre 400 y 3200 Mbits

por segundo. Como interfaz de video su uso más común es la interconexión de dispositivos digitales, principalmente video cámaras.

Figura 1.6. Conectores Firewire.

3. **Digital Visual Interface (DVI)**: Interfaz de video digital de alta velocidad. Nace con el objetivo de proporcionar un interfaz digital entre un ordenador y una pantalla. Consiste en un cable de cuatro pares trenzados: uno para cada color primario (rojo, verde y azul) y otro para el "reloj" (que sincroniza la transmisión) (ver Figura 1.7). La sincronización de la señal es casi igual que la de una señal analógica de vídeo.

   Sus características principales son:

   - Conector diseñado por el propio *Digital Display Working Group*.
   - Diseñado para maximizar la calidad visual de dispositivos de video con pantalla plana.
   - Conectividad "Plug&Play".
   - Posibilidad de transmisión analógica o digital según el tipo de conector: DVI-A, DVI-D o DVI-I
   - Se utiliza el formato de datos del estándar *Transition Minimized Differential Signaling* (TMDS), el cual no utiliza ningún tipo de compresión.

- Resolución mínima de 640 píxeles por 480 filas a 60Hz, siendo necesario un reloj de 25.715MHz.
- Frecuencia máxima de reloj de 165MHz, permitiendo resoluciones de 1280 píxeles por 1024 filas a 85 Hz en enlace único y 2560 píxeles por 1600 líneas a doble enlace.
- El tamaño de su conector, sus limitaciones de distancia de enlace y la no integración de audio, han sido las principales razones por las que se está dejando de utilizar.

Figura 1.7. Conector DVI.

4. **Display Port**: Interfaz de video libre de licencias y cánones, desarrollado por la *Video Electronics Standards Association* (VESA) tiene un funcionamiento y conector muy similar al HDMI y nace como alternativa a este (ver Figura 1.8). El DisplayPort admite un flujo de datos de hasta 10.8 Gbit/s, permitiendo una resolución de hasta 2560 píxeles por 1600 filas sobre un cable de 15 metros.

Figura 1.8. Conector Display Port.

1.2. Motivación y objetivo del proyecto

En el desarrollo de los sistemas de visión artificial, es habitual el uso de dispositivos basados en *Field Programmable Gate Array* (FPGA), esto se debe a la capacidad de estos dispositivos de realizar operaciones de procesamiento lógico en paralelo, con lo que se mejora el rendimiento de los algoritmos de procesamiento de imagen.

En la fase de diseño de estos sistemas de visión, es de gran utilidad tener la posibilidad de visualizar lo que se está "leyendo" (desde sensor, cámara,...), y es aquí donde radica la motivación para la realización de este proyecto, proporcionar la posibilidad de exportar la imagen procesada desde la FPGA a un monitor externo al diseño en desarrollo.

El objetivo principal de este proyecto es el diseño e implementación de un *core* sintetizable para el control de un interfaz HDMI embebido en un dispositivo reconfigurable FPGA, nos centraremos en diseñar el protocolo para la señal de video.

Existen diferentes soluciones para controlar el interfaz HDMI en el mercado, como es el caso del integrado ADV7511, incluido en la placa de desarrollo Zedboard, con lo que el uso del interfaz se reduce a una adaptación de formatos.

Para el desarrollo de este Proyecto, se hará uso de la placa de desarrollo Digilent ZYBO, donde disponemos del APSoC (All Programmable System on Chip) Zynq™-7000 directamente conectado a un puerto HDMI, lo que hace necesario el desarrollo de un controlador HDMI dentro de la FPGA. Dicho desarrollo se realizará a nivel RTL (Nivel de Transferencia de Registro) utilizando el lenguaje de descripción hardware VHDL, obteniendo un *core* exportable a otros chips tipo FPGA más económicos como el Artix-7. Las posibilidades evolutivas del *core* y la realización de Proyectos futuros con la ZYBO, han sido también determinantes en la elección.

El entorno de desarrollo utilizado será Vivado®, que nos dotará de las herramientas necesarias para sintetizar, comprobar y depurar nuestro controlador.

Habrá que realizar un estudio del estándar TMDS (Señal Diferencial de transición minimizada) cuyo protocolo para la transmisión de datos en serie a alta velocidad se utiliza en el interfaz de video HDMI.

1.3. Estructura de la memoria

La documentación generada durante la realización de este Proyecto, se ha resumido en seis capítulos cuyos aspectos más importantes se detallan a continuación:

- **Capítulo 1: Introducción.** Breve repaso al contexto histórico y tecnológico realizando un repaso de los diferentes interfaces de video.
- **Capítulo 2: High Definition Multimedia Interface.** Análisis de las principales características de la interfaz HDMI así como del estándar TMDS.
- **Capítulo 3: Elementos y tecnologías utilizados.** Descripción y justificación de medios hardware y software utilizados en la realización de este Proyecto.
- **Capítulo 4: Desarrollo del sistema.** Descripción del desarrollo de la interfaz así como de los módulos que la componen.
- **Capítulo 5: Fase de pruebas.** Descripción del entorno de pruebas y de los procesos necesarios para verificar el correcto funcionamiento del sistema.
- **Capítulo 6: Conclusiones y líneas futuras de trabajo.** Exposición de conclusiones obtenidas de la realización del Proyecto y posibles líneas de trabajo futuras para la ampliación y uso del mismo.
- **Apéndice A: Código del Sistema.** Se incluyen los códigos de descripción hardware generados que componen el sistema.

Capítulo 2. High Definition Multimedia Interface

El HDMI se trata de un interfaz de video y audio multicanal digital capaz de soportar todos los estándares de video y alta definición sin comprimir, incluye un flujo de información de control y estado bidireccional entre transmisor y receptor.

La transferencia de información está basada en el estándar TMDS, el cual será descrito en este capítulo.

El tamaño de pixel de video tiene un valor por defecto de 24 bits para lo cual la frecuencia del TMDS-CLK será la misma que la frecuencia de pixel, siendo capaz de soportar también tamaños de 30, 36 o 48 bits. Se pueden codificar los formatos de video RGB, YCBCR 4:4:4 e YCBCR 4:2:2.

Las diferentes prestaciones o requerimientos del sistema dependen de la versión de HDMI que se utilice. En la Tabla 2.1. podemos observar las principales características según la versión.

Versión HDMI	*1.0*	*1.1*	*1.2*	*1.3*	*1.4*	*2.0*	*2.1*
Máximo ancho de banda (Gbps)	*4.95*	*4.95*	*4.95*	*10.2*	*10.2*	*18*	*48*
Resolución máxima	*1920x1200p 60Hz*	*1920x1200p 60Hz*	*1920x1200p 60Hz*	*2560x1600p 60Hz 2048x1536p 75Hz*	*3840x2160p 30Hz 4096x2160p 24Hz*	*4096x2160p 60Hz*	*3840x2160p 120Hz 7680x4320p 60Hz 9600x5400p 60Hz*
Canales de audio	*8*	*8*	*8*	*8*	*8*	*32*	*--*
Máxima frecuencia de audio (KHz)	*768*	*768*	*768*	*768*	*768*	*1536*	*--*

Tabla 2.1. Principales características de las versiones HDMI. [1]

2.1. Principio de funcionamiento

Lo primero que se va a analizar es el funcionamiento básico de un sistema HDMI, además de dar una perspectiva eléctrica. Para ello se hará uso del gráfico de la Figura 2.1, donde vienen definidas todas las líneas de comunicación entre el transmisor y el receptor.

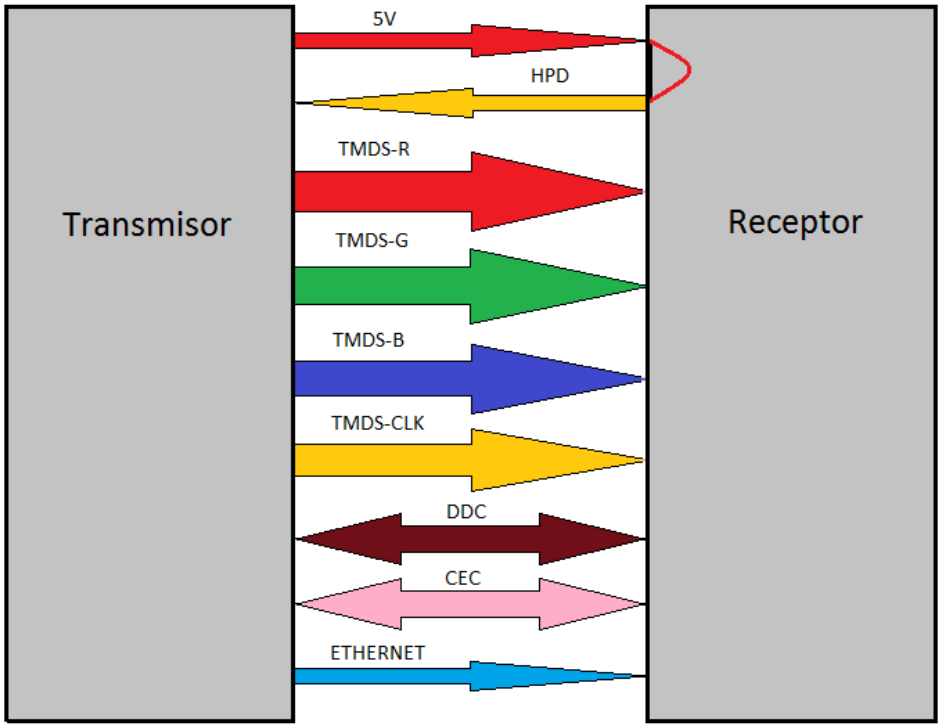


Figura 2.1. Enlace HDMI. [2]

- **5 Voltios**: Toda comunicación HDMI se inicia con una señal de alimentación de 5 V de continua de transmisor a receptor, supone la alimentación del sistema HDMI en el receptor incluso si este está apagado. Estos 5 V tendrán que estar regulados para no salir del margen de entre 4.8 y 5.3 V, esta condición es totalmente necesaria para iniciar y mantener el enlace.
- **HPD (Hot Plug Detect)**: El segundo paso es la activación de una señal de 5 V de continua de receptor a transmisor como confirmación de haber recibido la alimentación inicial. Es necesario un retardo mínimo de 100ms. desde que se ha recibido la señal inicial de 5 V.

- **Canales TMDS**: Una vez se ha creado el enlace, se envía la información de video, audio, control, sincronización y temporización a través de los canales 0, 1, 2 (RGB) y CLK. Toda esta información está codificada y serializada siguiendo la configuración del estándar TMDS tal y como se analizará en apartados posteriores.
- **DDC (Display Data Channel)**: Enlace doble de comunicación serie que porta señales de datos y relojes con información sobre los datos que se están enviando o sobre las capacidades del receptor, tales como resoluciones o frecuencias.
- **CEC (Consumer Electronics Control)**: Conexión utilizada por sistemas que incorporan "One touch control" o en enlaces en los que se desea implementar utilizando el HDMI. Se utiliza también como canal de mensajes de sistema en comunicaciones de varios dispositivos interconectados en serie mediante HDMI.
- **Ethernet**: Utilizado a partir de la versión de HDMI 1.4, se utiliza como segunda línea de un sistema de doble enlace Ethernet haciendo uso de la línea HPD. Se utiliza también como ARC (Audio Return Channel).

2.2. Características físicas y eléctricas

Conectores

Existen 5 tipos de conectores HDMI tal y como se muestra en la Figura 2.2

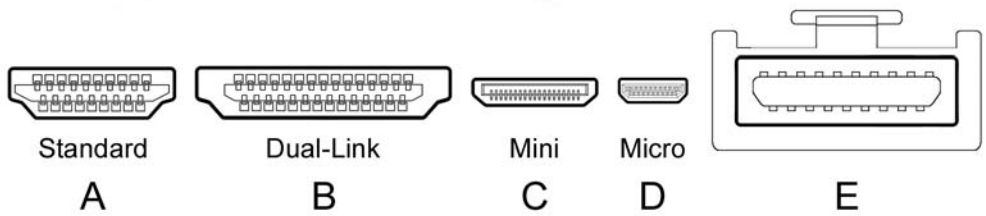


Figura 2.2. Tipos de conectores HDMI.

- **Tipo A**: Conector más habitual y comercializado, utilizado en la realización de este Proyecto. Está formado por 19 pines (Ver Figura 2.3, Figura 2.4, y Tabla 2.2).

Figura 2.3 Conector HDMI Tipo A.

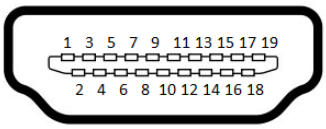


Figura 2.4. Pinout HDMI Tipo A.

Pin	Señal	Pin	Señal
1	TMDS_2 +	2	Protección TMDS_2
3	TMDS_2 -	4	TMDS_1 +
5	Protección TMDS_1	6	TMDS_1 -
7	TMDS_0 +	8	Protección TMDS_0
9	TMDS_0 -	10	TMDS_CLK +
11	Protección TMDS_CLK	12	TMDS_CLK -
13	CEC	14	N.C.
15	SCL	16	SDA
17	DDC/CEC Ground	18	+5 Power
19	HPD		

Tabla 2.2. Pinout del conector HDMI Tipo A. [3]

- **Tipo B**: Versión de conector preparado para uso futuro no muy extendido hasta el momento. Sus principales características son:
    - Está formado por 29 pines.
    - Incluye tecnología "Dual-link" para soportar señales DVI con regímenes de datos superiores a 165Mpixels por segundo. Para ello, se añade un nuevo canal TMDS además de los RGB y CLK.
    - Tamaño considerablemente mayor que el Tipo A.
- **Tipo C y Tipo D**: Versiones del conector Tipo A mini y micro, se trata de conectores con la misma tecnología que el Tipo A pero de menor tamaño. Diseñados para integrar la tecnología HDMI en dispositivos de tamaño reducido.
- **Tipo E**: Conector de uso minoritario, diseñado para soportar altas temperaturas y posibles movimientos. Se crea con el objetivo de utilizar la tecnología HDMI en la industria automotriz.

Especificaciones del estándar TMDS

El estándar TMDS indica una serie de requisitos eléctricos de obligado cumplimiento para una correcta transmisión diferencial. Los más importantes son una tensión AV_{CC} de 3.3V ±5% y una resistencia R_T de 50ohms ±10% (ver Figura 2.5)

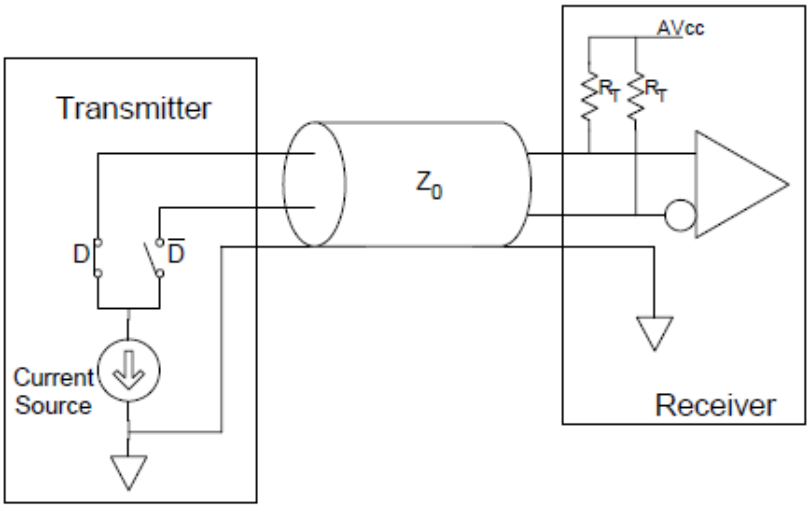


Figura 2.5. Enlace TMDS básico. [3]

Por otro lado, se especifican condiciones de fluctuaciones de relojes y datos necesarias para la sincronización y temporización de los diferentes canales de datos que permitirán una correcta recuperación de los datos enviados.

Las más importantes son las referentes a "jitters" y tamaños del "ojo" del canal diferencial y se resumen en la Figura 2.6 para transmisor y receptor..

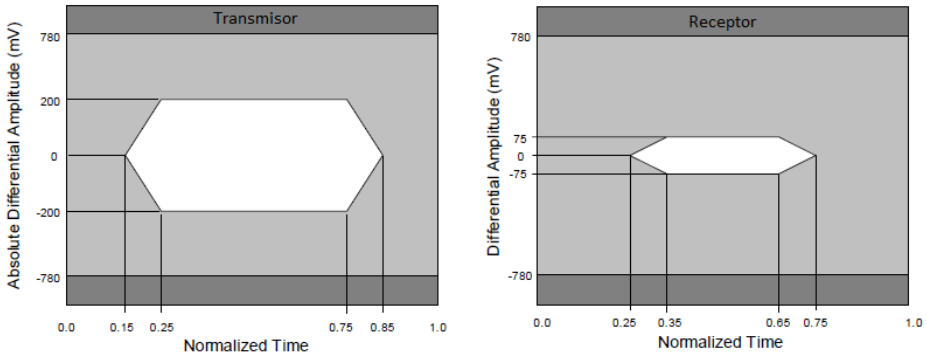


Figura 2.6. Requisitos de tamaño del "ojo" en transmisor y receptor TMDS. [3]

2.3. Modos de operación y codificación

El flujo de datos en un enlace HDMI se realiza a través de los tres canales de datos y el canal de reloj. Por el canal de reloj se transmite de forma diferencial una señal de frecuencia proporcional a la del reloj de píxel de la señal de video que se está transmitiendo (ver Figura 2.7). En cada ciclo del canal de reloj, se transmite por cada uno de los canales TMDS-0, TMDS-1 y TMDS-2 un dato en serie de diez bits fruto de la codificación correspondiente.

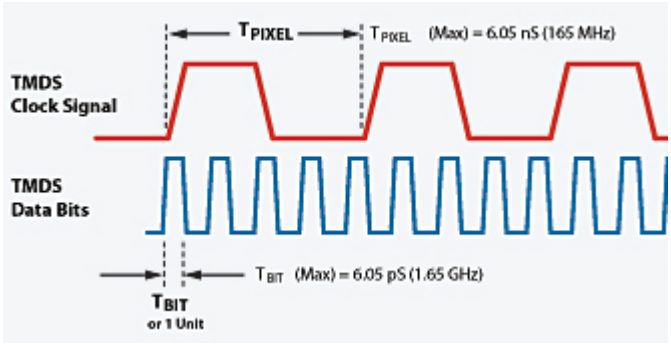


Figura 2.7. TMDS-CLK y CLK para transmisión serializada. [2]

El envío de la información en el enlace HDMI se realiza sobre tres posibles modos de funcionamiento según se esté enviando señal de video, señal de audio /señales auxiliares o señales de control. Estos tres periodos son, respectivamente: Periodo de Video Activo, Periodo de Isla de Datos y Periodo de Control.

Periodo de Video Activo

La transmisión de datos de video se realiza haciendo uso del algoritmo de codificación TMDS. Dicho algoritmo toma como dato de entrada el valor del pixel de alguna de las componentes de video de 8 bits y genera un dato de salida codificado de 10 bits.

El principal objetivo de esta codificación es reducir el número de transiciones en la secuencia de envío de datos y conseguir una transición DC balanceada.

En una primera etapa, se genera un dato interno de 9 bits basado en las transiciones del dato de entrada y encabezado por un bit que indica el método de generación utilizado, este método dependerá del número de unos del dato de entrada y del valor de un registro donde se guarda la disparidad del dato anterior. En ambos casos, el bit menos significativo coincide con el del dato de entrada y se consigue la reducción de las transiciones mediante secuencias XOR o XNOR según corresponda (ver Figura 2.8).

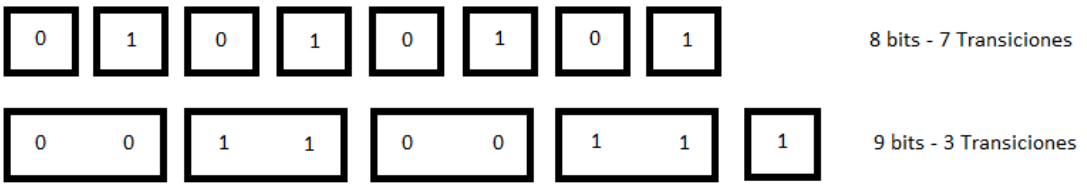


Figura 2.8. Ejemplo de reducción de transiciones en la primera parte del algoritmo TMDS. [2]

En la segunda etapa, se genera el dato de salida de 10 bits. Inicialmente se realiza un balanceo DC selectivo invirtiendo los 8 bits de datos incluidos en el dato de 9 bits generado en la primera etapa (ver Figura 2.9), este balanceo se realiza en función de la disparidad, tanto del último dato enviado como del dato que se está codificando. El objetivo es evitar largas cadenas de datos sin transiciones, ya que esto mejora la eficiencia del sistema, la relación señal a

ruido y reduce las emisiones. Por último, se añade un decimo bit que indica si se ha realizado o no dicha inversión.

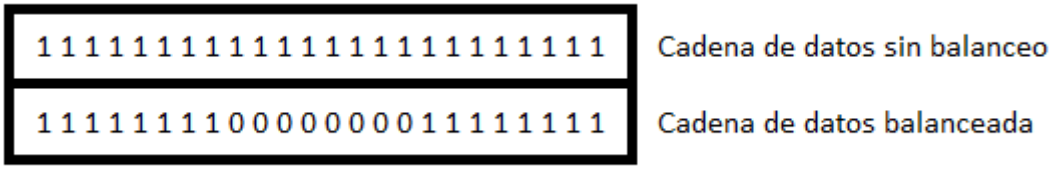


Figura 2.9. Ejemplo de balanceo DC. [2]

Para una mejor compresión de la codificación, se muestra el algoritmo en forma de diagrama de flujo (ver Figura 2.10).

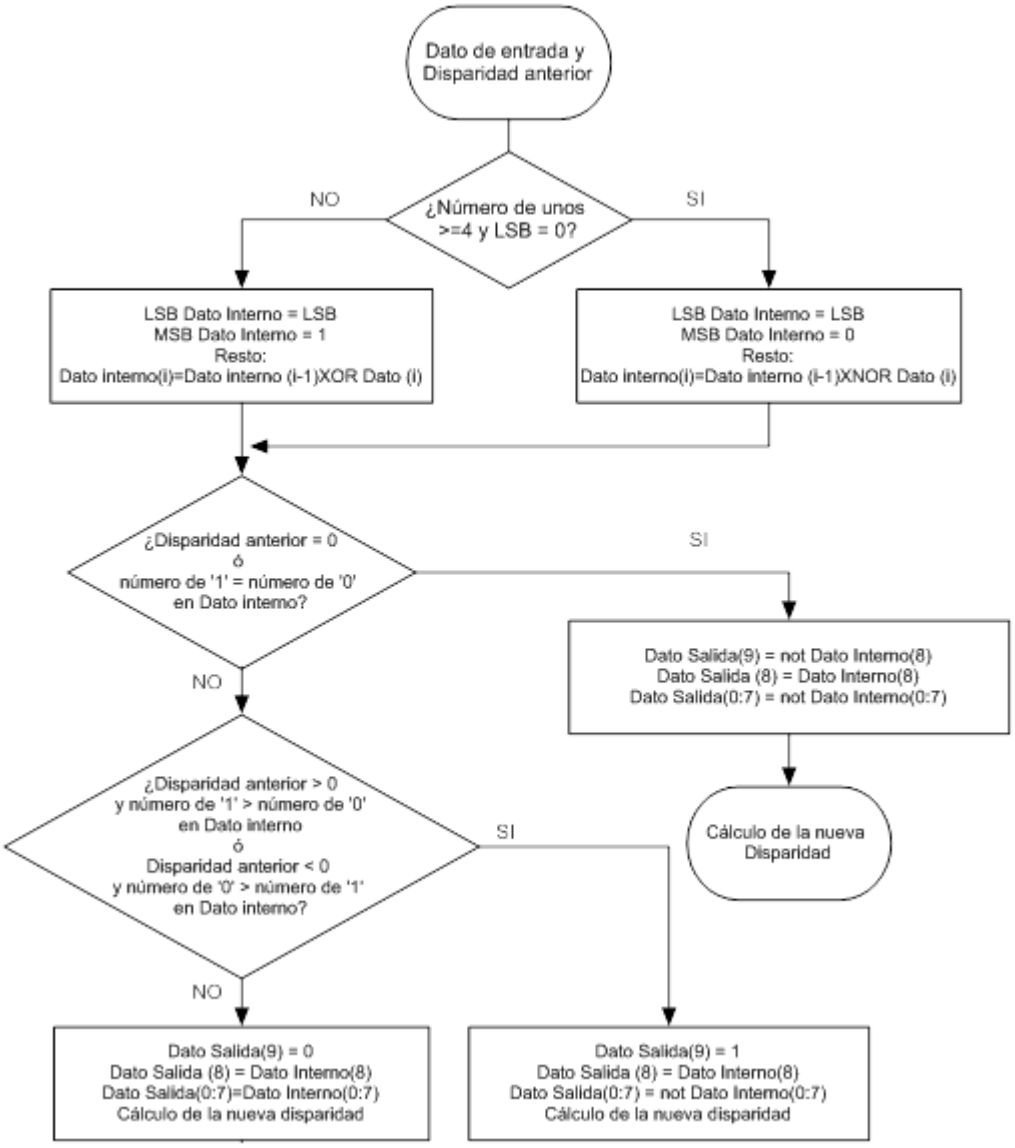


Figura 2.10. Algoritmo de codificación de video TMDS. [3]

Todos los periodos de video activo están precedidos de una banda de guarda durante dos ciclos de TMDS-CLK con un patrón de datos de salida fijo. Este patrón es diferente según el canal (ver Tabla 2.3).

Canal TMDS	Patrón de salida
0	1011001100
1	0100110011
2	1011001100

Tabla 2.3. Patrones de banda de guarda previos al envío de información de video. [3]

Periodo de Isla de Datos

Los periodos de islas de datos son utilizados para el envío de datos de audio y datos auxiliares. La transmisión de datos en este periodo se realiza haciendo uso del algoritmo de codificación TERC4 (TMDS Error Reduction Coding), dicho algoritmo toma como entrada un dato de 4 bits que asigna de forma directa a un dato de salida codificado de 10 bits (ver Tabla 2.4).

En el caso del canal TMDS-0, la configuración de los 4 bits de entrada al algoritmo se realizará de la siguiente forma:

1. El bit 0 (LSB) transmite la información correspondiente a la sincronización de video horizontal.
2. El bit 1 transmite la información correspondiente a la sincronización de video vertical.
3. El bit 2 transmite la información de cabecera del paquete.
4. El bit 4 (MSB) será cero en el primer ciclo y uno en los siguientes.

Los canales TMDS-1 y TMDS-2 transmiten la información del paquete.

Dato de entrada	Codificación TERC4
0000	1010011100
0001	1001100011
0010	1011100100
0011	1011100010
0100	0101110001
0101	0100011110
0110	0110001110
0111	0100111100
1000	1011001100
1001	0100111001
1010	0110011100
1011	1011000110
1100	1010001110
1101	1001110001
1110	0101100011
1111	1011000011

Tabla 2.4. Asignación de valores TERC4. [3]

Todos los periodos de isla de datos están precedidos y seguidos de una banda de guarda durante dos ciclos de TMDS-CLK con un patrón de datos de salida fijo, este patrón es diferente según el canal (Ver Tabla 2.5).

Canal TMDS	Patrón de salida
0	Codificación TERC4 de "1-1-Vsync-Hsync"
1	0100110011
2	0100110011

Tabla 2.5. Patrones de banda de guarda previos y posteriores al envío de una Isla de Datos. [3]

Los paquetes de información enviados tienen un tamaño fijo de 32 ciclos y cada Periodo de Isla de Datos puede contener un número entero de paquetes entre 0 y 18. Es aconsejable incluir al menos una Isla de Datos cada dos Periodos de Video Activo.

Para la realización de este Proyecto, solo se hará uso del Paquete de Datos NULL a modo de muestra de la codificación TERC4 y la inclusión de Islas de Datos en el flujo de información (ver Tabla 2.6). El resto de paquetes disponibles para transmisión de audio y señales auxiliares no serán motivo de estudio.

Número de ciclo de reloj	TMDS-0	TMDS-1	TMDS-2
0	00VsyncHsync	0000	0000
1	10VsyncHsync	0000	0000
...	...	...	...
31	10VsyncHsync	0000	0000

Tabla 2.6. Valores de entrada TERC4 del paquete de datos NULL. [3]

Periodo de Control

El Periodo de Control se utiliza como preámbulo de la información que se va a enviar posteriormente y tiene un tamaño mínimo de 12 ciclos de reloj.

La transmisión de datos en este periodo se realiza mediante codificación directa, tomando como entrada un dato de 2 bits se asigna un dato de salida de 10 bits (ver Tabla 2.7).

El dato de entrada estará formado por las señales de control C1 y C0 cuya asignación dependerá del canal TMDS (ver Tabla 2.8).

C1	C0	Dato de Salida
0	0	1101010100
0	1	0010101011
1	0	0101010100
1	1	1010101011

Tabla 2.7. Asignación de valores de patrones de salida según señales de control C0 y C1. [3]

Canal TMDS	C1	C0
0	Vsync	Hsync
1	0	1
2	0	0 si Pre-Video 1 si Pre-Isla

Tabla 2.8. Asignación de valores a las señales de control C1 y C0 según el canal. [3]

A modo de síntesis de este Capítulo podemos observar la secuencia de envío de información en el interfaz HDMI (ver Figura 2.11).

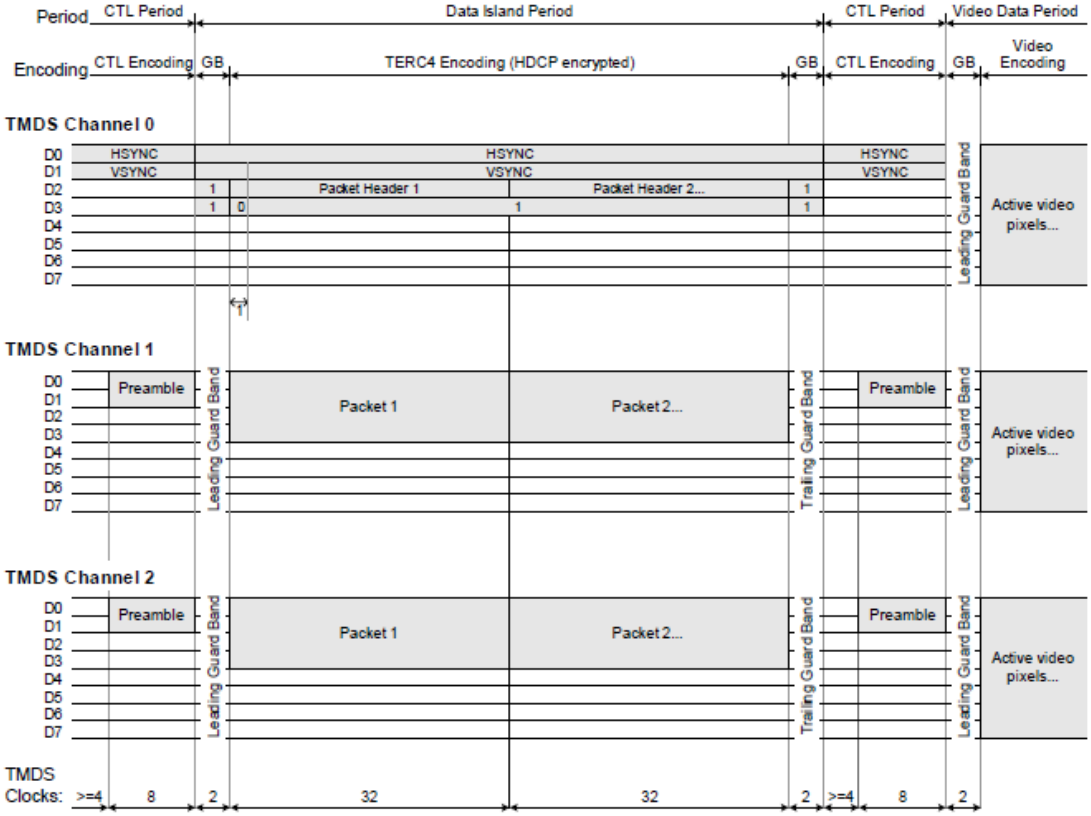


Figura 2.11. Secuencia de envío de información HDMI. [3]

Capítulo 3. Elementos y tecnologías utilizados

En este capítulo se describen las herramientas necesarias para la realización y prueba de este Proyecto.

Los medios materiales utilizados han sido:

- Ordenador ASUS N53J con sistema operativo Windows 7
- Placa de desarrollo Digilent ZYBO Zynq™-7000
- Entorno de desarrollo Xilinx Vivado® versión 2016.4
- Monitor HP EliteDisplay E232
- Cable de interconexión HDMI Tipo A

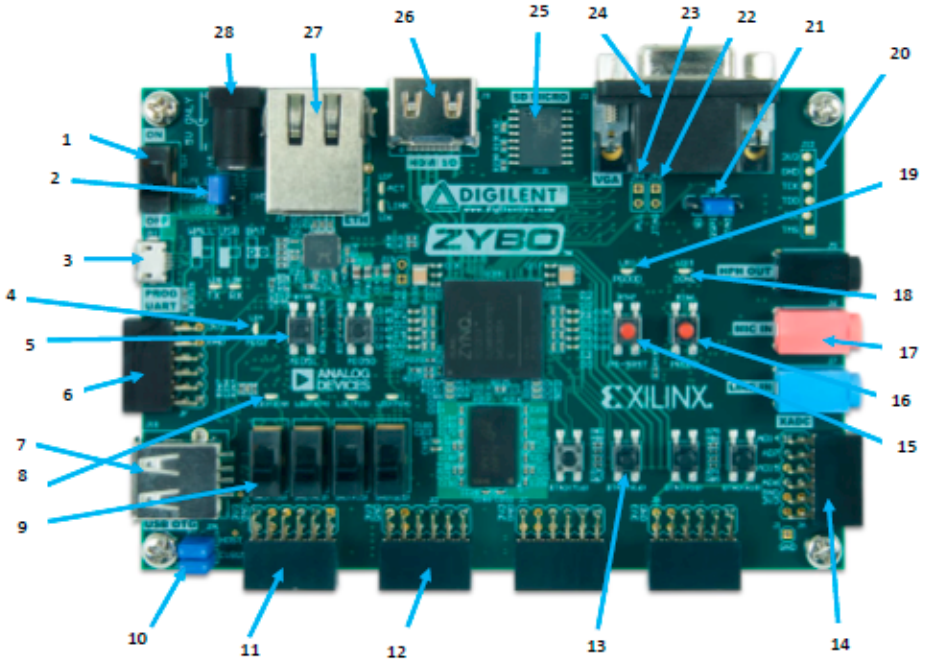
3.1. Placa de desarrollo Digilent ZYBO Zynq™-7000

La placa de desarrollo ZYBO está basada en la arquitectura Xillinx APSoC, que integra un procesador dual-core ARM Cortex-A9 con la lógica FPGA.

La ZYBO incluye:

- Procesador dual-core Cortex-A9 650Mhz.
- Controlador de memoria DDR3 con 8 canales DMA.
- Controladores de periféricos: 1G Ethernet, USB 2.0, SDIO, SPI, UART, CAN, I2C.
- Lógica reprogramable:
    - 4.400 segmentos lógicos, cada uno de ellos con 4 LTUs de 6 y 8 flip-flops.
    - 240 KB de RAM.
    - Dos gestores de reloj, cada uno con un PLL (Phase-Locked loop) y un MMCM (Mixed-Mode Clock Manager).
    - 80 DSP.
    - Reloj interno con velocidades por encima de 450MHz.
    - Convertidor analógico-digital (XADC).
- APSoC ZYNQ XC7Z010-1CLG400C.
- 32 DDR3 w/ de 512MB y 1050Mbps de ancho de banda.
- HDMI.
- VGA de 16 bits por píxel.
- Ethernet (1Gbit/100Mbit/10Mbit).
- MicroSD.
- USB 2.0.
- EEPROM externa.
- Jack's de audio para altavoz, micrófono y auxiliar.

- Flash serie de 128Mb.
- Programación JTAG y conversor UART-USB.
- GPIO: 6 interruptores pulsadores, 4 interruptores selectores y 5 LED's.
- 6 conectores Pmod.



Callout	Component Description	Callout	Component Description
1	Power Switch	15	Processor Reset Pushbutton
2	Power Select Jumper and battery header	16	Logic configuration reset Pushbutton
3	Shared UART/JTAG USB port	17	Audio Codec Connectors
4	MIO LED	18	Logic Configuration Done LED
5	MIO Pushbuttons (2)	19	Board Power Good LED
6	MIO Pmod	20	JTAG Port for optional external cable
7	USB OTG Connectors	21	Programming Mode Jumper
8	Logic LEDs (4)	22	Independent JTAG Mode Enable Jumper
9	Logic Slide switches (4)	23	PLL Bypass Jumper
10	USB OTG Host/Device Select Jumpers	24	VGA connector
11	Standard Pmod	25	microSD connector (Reverse side)
12	High-speed Pmods (3)	26	HDMI Sink/Source Connector
13	Logic Pushbuttons (4)	27	Ethernet RJ45 Connector
14	XADC Pmod	28	Power Jack

Figura 3.1. Placa de Desarrollo ZYBO. [4]

Nos centraremos en la descripción de los elementos que han sido utilizados en la realización de este Proyecto.

Alimentación

La ZYBO se puede alimentar mediante el puerto USB-JTAG-UART a través del puerto J11 (ver Figura 3.1, Callout 3), mediante alimentación de red eléctrica a través del conector J15 (ver Figura 3.1, Callout 28) o mediante una batería externa. La selección de alimentación se realiza a mediante la posición del jumper JP7 (ver Figura 3.1, Callout 2).

Ha sido necesaria la alimentación a red eléctrica debido a que el nivel de tensión del puerto USB, unido a la longitud y calidad del cable utilizado, no permitía niveles de salida acordes a las especificaciones del estándar TMDS.

Zynq Z-7010

El APSoC Zynq está dividido en dos subsistemas: PS (Processing System) y PL (Programmable Logic) (ver Figura 3.2). El PS constituye un sistema formado por los procesadores y conexiones a las memorias y distintos periféricos, así como interfaces a los dispositivos físicos externos de cada placa. La PL la constituye una FPGA con comunicación física directa con el exterior y la posibilidad de interconexión entre la lógica implementada y la PS a través de buses AXI.

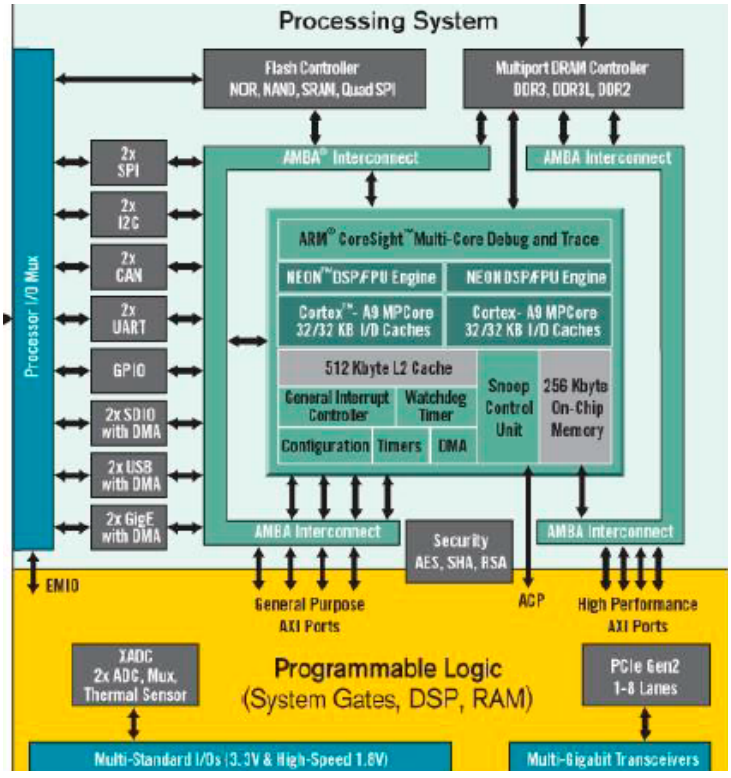


Figura 3.2. Esquema PS y PL del APSoC Zynq Z-7010. [4]

HDMI

Además de las múltiples posibilidades de la ZYBO, tratándose de una buena herramienta en futuros proyectos, una de sus principales características por la cual ha sido la placa de desarrollo elegida en la realización de este Proyecto, es la inclusión de un puerto HDMI directamente conectado a la FPGA, con todo lo que esto supone, descrito en el Capítulo 1.

Se han testeado resoluciones de hasta 720p (1280x720) en la ZYBO [].

La descripción de los pines de entrada/salida se define en la Tabla 3.1, el puerto HDMI_OUT_EN será el que defina el funcionamiento como transmisor ('1') o como receptor ('0').

Señal HDMI	PIN
HDMI_CLK_N	H17
HDMI_CLK_P	H16
HDMI_D0_N	D20
HDMI_D0_P	D19
HDMI_D1_N	B20
HDMI_D1_P	C20
HDMI_D2_N	A20
HDMI_D2_P	B19
HDMI_CEC	E19
HDMI_HPD	E18
HDMI_OUT_EN	F17
HDMI_SCL	G17
HDMI_SDA	G18

Tabla 3.1 Pinout del conector HDMI de la Placa de Desarrollo ZYBO. [5]

Señales de Reloj

La ZYBO provee de una señal de reloj de 50MHz a la entra de reloj del PS, permitiendo que el procesador opere a frecuencias de hasta 650MHz y los DDR3 hasta 525MHz. Se incluye además una señal de reloj externa de 125MHz directamente conectada en el PIN L16 de la parte PL (ver Figura 3.3).

Se dispone de dos MMCMs y dos PLLs que permiten la posibilidad de generar señales de reloj de diferentes frecuencias. Alguno de los 4 relojes de referencia del PS o la señal de reloj de 125MHz pueden ser utilizadas como entradas en la generación de estas señales de reloj.

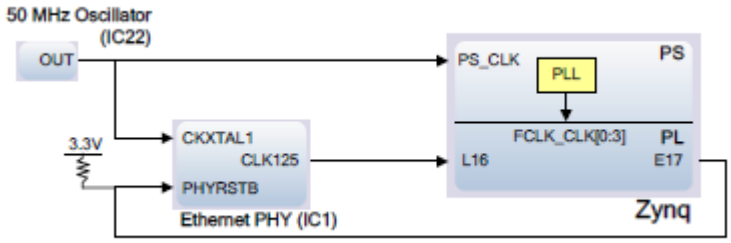


Figura 3.3. Señales de reloj de la Placa de Desarrollo ZYBO. [4]

Entrada/Salida básica

Se incluyen 6 interruptores pulsadores, 4 interruptores selectores y 5 LEDs de uso genérico para entrada y salida básica. Todos ellos divididos en parte PS y PL (ver Figura 3.4).

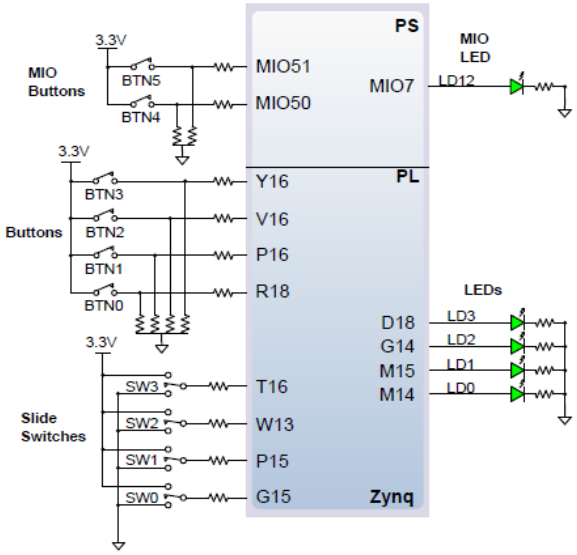


Figura 3.4. Entradas y Salidas básicas de la Placa de Desarrollo ZYBO. [4]

3.2. Entorno de desarrollo Xilinx Vivado®

Herramienta para diseño hardware que permite tanto la configuración y personalización de la parte del PS como la adición e interconexión de bloques de lógica o IPs (*Intellectual Properties*) en la FPGA o PL, todo ello en un entorno que integra ambas secciones (ver Figura 3.5).

El proceso de trabajo para la creación de un Proyecto a nivel RTL como el que se ha realizado se compone de los siguientes pasos:

1. Creación de Nuevo Proyecto.
2. Creación y diseño de fuentes mediante el lenguaje de descripción hardware VHDL y configuración de módulos utilizados.
3. Simulación: Se configuran valores de entrada para la creación de TestBench y estudio de cronogramas.
4. Síntesis: Se comprueba que nuestro diseño es sintetizable.
5. Implementación: Se comprueba que nuestro diseño se puede implementar.
6. Configuración de puertos de entrada/salida.
7. Generación de archivo de programación "Bitstream".
8. Conexión y programación de la placa de desarrollo.

Serán motivo de estudio y corrección los posibles errores derivados de los procesos de Síntesis, Implementación, Simulación y Generación de Bitstream. Además, serán motivo de estudio los posibles warnings que se generen en los mismo procesos.

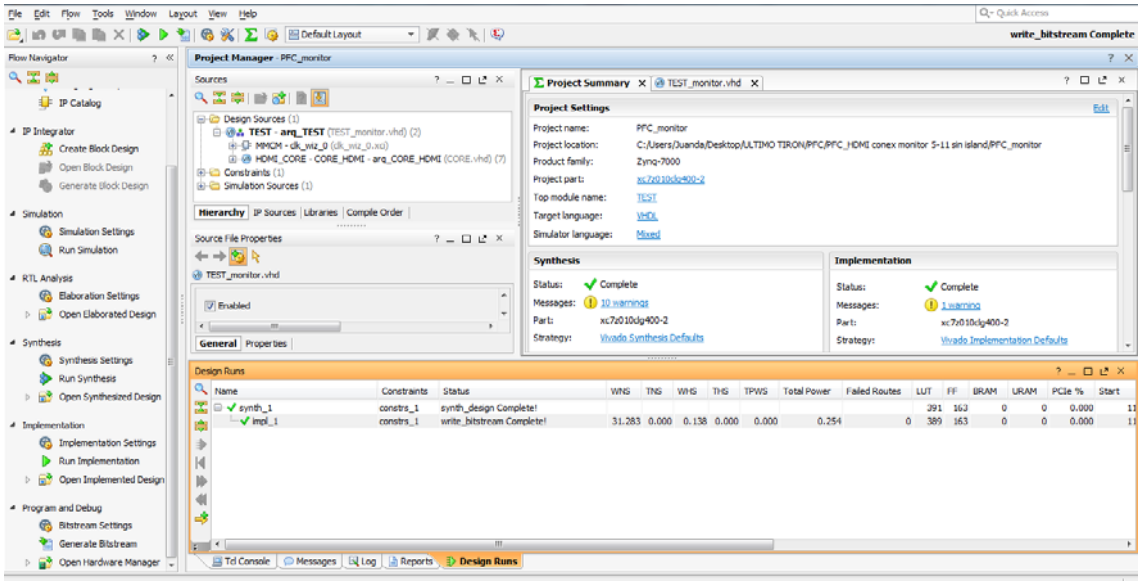


Figura 3.5. Entorno de desarrollo Vivado.

3.3. Recursos para generación de señales de reloj

Las FPGA de la Serie 7 de Xilinx ofrecen la posibilidad de utilizar el LogiCore IP Clocking Wizard que nos facilita el uso y gestión de los MMCM y PLL a fin de definir las señales de reloj necesarias en nuestro diseño.

Dispone de dos entradas de reloj simples o diferenciales y es capaz de generar hasta siete señales de reloj de salida diferentes. Entre sus muchas ventajas, las más importantes para el desarrollo de la interfaz HDMI son:

- Síntesis de frecuencia
- Alineamiento de fase
- Minimización de fluctuaciones a la salida
- Amplia tolerancia a jitters de entrada
- Buffer automático

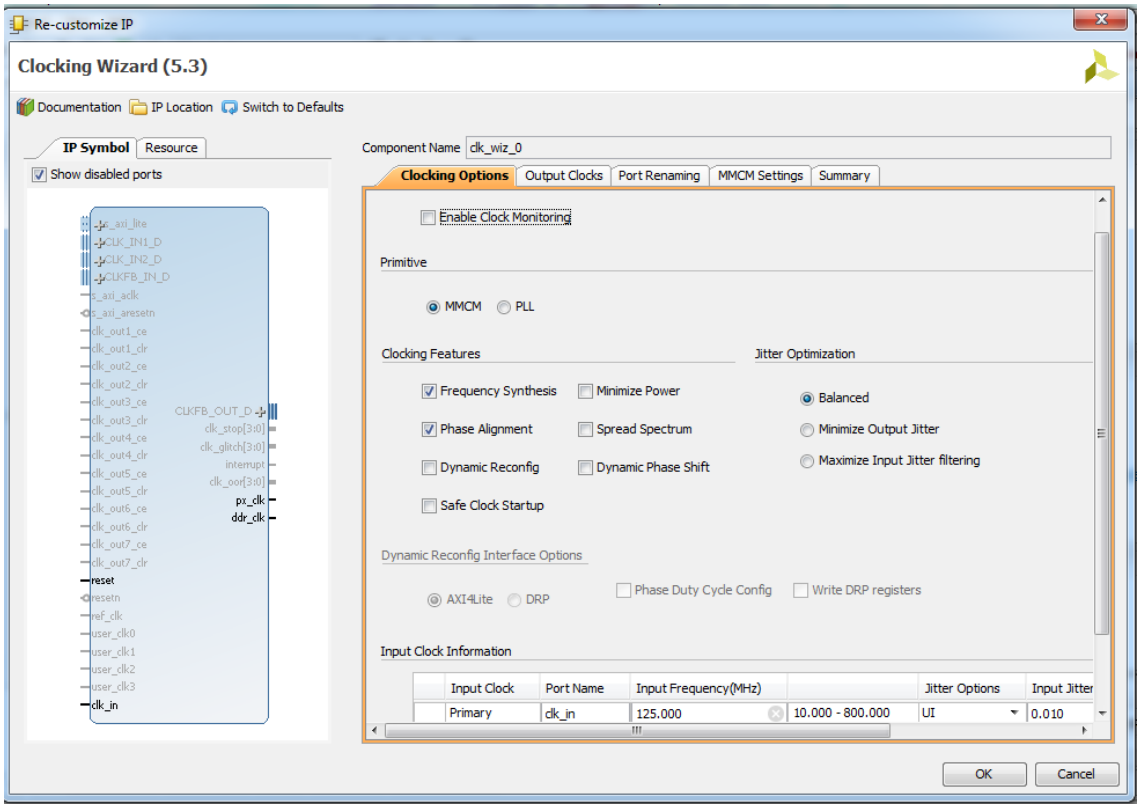


Figura 3.6.Ventana de configuración de Clocking Wizard en entorno Vivado.

3.4. Recursos avanzados de entrada y salida

Por otro lado, las FPGA de la Serie 7 de Xilinx ofrecen también la posibilidad de utilizar una serie de recursos de entrada y salida avanzados, se realiza la descripción de los utilizados en la realización de este Proyecto.

OBUFDS

Buffer de salida diferencial, permite la generación de una señal de salida diferencial a partir de una entrada en serie.

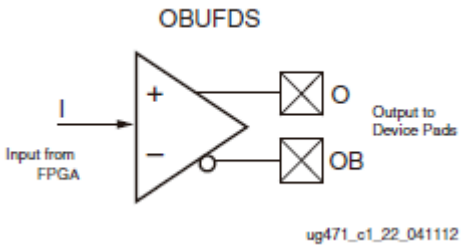


Figura 3.7. Recurso avanzado de salida OBUFDS. [6]

OUTPUT DDR (ODDR)

Registro de salida. Su primitiva nos ofrece dos entradas, un set/reset, un puerto de habilitación de reloj, una señal de reloj y la propia salida registrada.

Se pueden configurar parámetros de sincronismo y valor inicial de salida en la declaración de la primitiva.

Entre las ventajas de registrar una salida, la que se busca con el uso de este recurso es la sincronización con el resto de salidas del sistema.

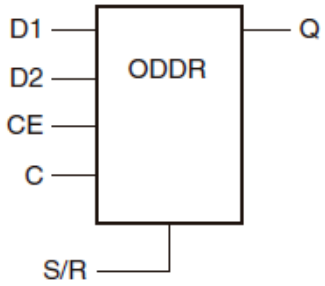


Figura 3.8. Primitiva del recurso avanzado de salida ODDR. [6]

OSERDESE2

Convertidor de dato paralelo a serie de entre 2 a 8 bits de entrada, se puede realizar una expansión utilizando dos módulos OSERDESE2 para serializar un dato de entrada de hasta 14 bits [6]. La configuración utilizada se detallará cuando se haga uso de este bloque en la descripción del sistema.

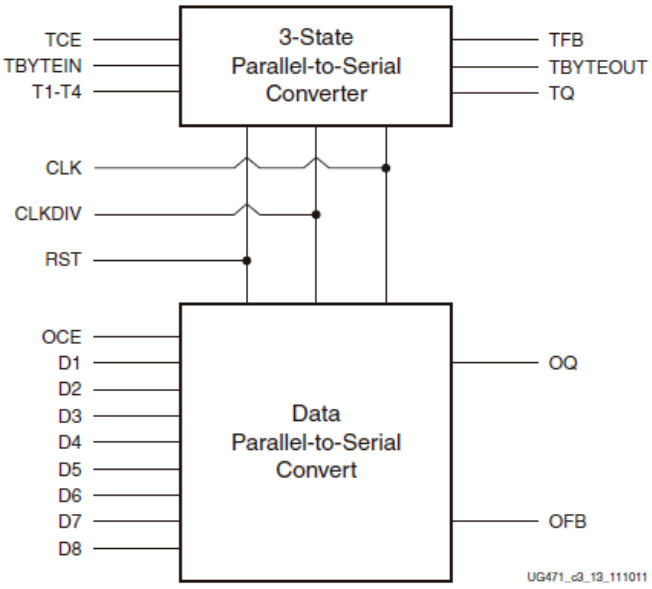


Figura 3.9. Diagrama de bloques del recurso avanzado de salida OSERDESE2. [6]

Capítulo 4. Desarrollo del sistema

4.1. Introducción

Para el desarrollo de la interfaz se ha empleado una metodología incremental, lo que nos ha permitido mantener el control sobre la complejidad y los riesgos del proyecto. Una vez realizado el estudio previo resumido en los capítulos anteriores, se han ido generando y verificando los diferentes módulos que componen el controlador así como la interconexión entre ellos.

Por último, se ha desarrollado un módulo de generación de señales y patrones de video que nos ha permitido verificar el funcionamiento del *core* mediante la conexión de la ZYBO a un monitor externo.

4.2. Especificaciones

Se resumen las especificaciones técnicas del controlador desarrollado:

- La entradas y salidas del *core* (ver figura 4.1).
- El interfaz se ha desarrollado para una resolución fija de 720x480 60Hz, siendo necesaria una señal de reloj de píxel de 27MHz y una señal de reloj de bit para salida diferencial de 135MHz (cinco veces mayor).
- El RESET se sincroniza de forma interna en el controlador.
- DEN indica que nos encontramos en un periodo de video activo.
- HSync y Vsync deben ser las adecuadas para la resolución de 720x480 60Hz.
- El dato de entrada PIXEL_DATA de 24 bits, estará compuesto por los valores de las componentes RGB de cada píxel.
- Se proporcionan como señales de salida los canales TMDS-0, TMDS-1, TMDS-2 y TMDS-CLK.

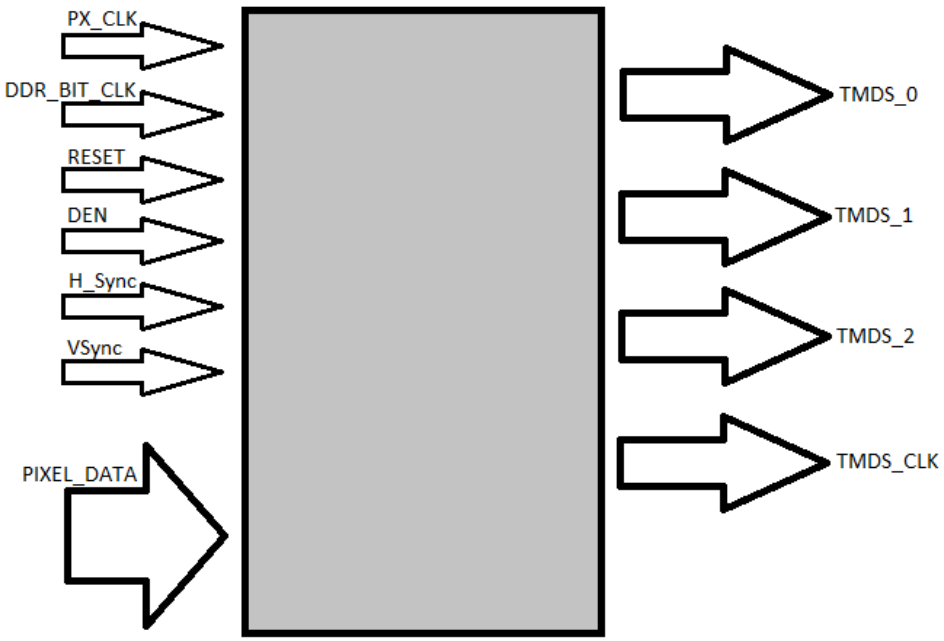


Figura 4.1. Esquema general de entradas y salidas del controlador desarrollado.

4.3. Desarrollo de la interfaz

Los principales pasos seguidos en el desarrollo de la interfaz han sido, por orden de realización:

1. Diseño y verificación de módulo para la codificación de video activo.
2. Inclusión y verificación de codificación de control y codificación de isla de datos con paquete NULL en el módulo de codificación.
3. Diseño y verificación de módulo para serialización del dato de salida del codificador.
4. Diseño y verificación de módulo para la generación de la señal de sincronización TMDS-CLK.
5. Diseño y verificación de módulo *core* fruto de las interconexiones de los módulos anteriores, se utilizara un módulo codificador y un serializador para cada canal TMDS RGB (ver Figura 4.2).
6. Diseño y verificación de banco de pruebas, generador de señales de video y patrones, así como de las señales de reloj necesarias para su funcionamiento.
7. Inclusión y verificación de llamada al *core* diseñado desde el banco de pruebas.
8. Conexión a monitor externo.

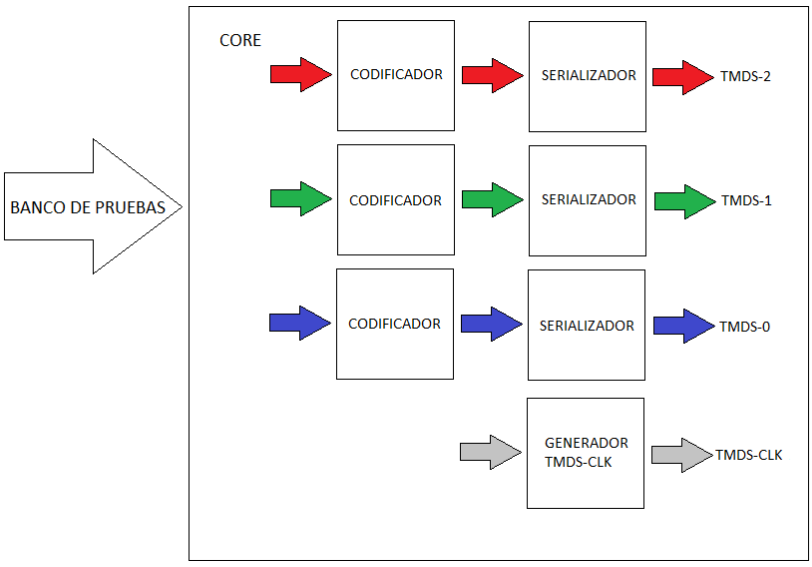


Figura 4.2. Esquema general del controlador desarrollado.

4.3.1. Módulo Codificador

En este módulo se realiza la codificación o asignación de valores de 8 bits obteniendo el dato de salida de 10 bits codificado. Para ello, nos serviremos como guía del análisis de la codificación en los diferentes periodos de funcionamiento del interfaz HDMI, visto en el Capítulo 2.

El tipo de información enviada según el periodo, se divide en los siguientes casos:

- Señal de video.
- Isla de datos.
- Señal de control, preámbulo de señal de video.
- Señal de control, preámbulo de isla de datos.
- Banda de guarda de la señal de video.
- Banda de guarda de isla de datos.

Como se ha visto en el apartado anterior, la resolución se ha fijado a 720x480, estos valores definen la zona de video activo, sin embargo, el tamaño real no visible es de 858x525, quedando unas zonas de 858x45 en la parte superior (blanking vertical) y de 138x480 en la parte izquierda (blanking horizontal) (ver Figura 4.3).

Por simplicidad, se ha optado por enviar islas de datos solo en la zona de blanking horizontal, utilizando todo el blanking vertical para el envío exclusivo de información de control puesto que la especificación HDMI lo permite.

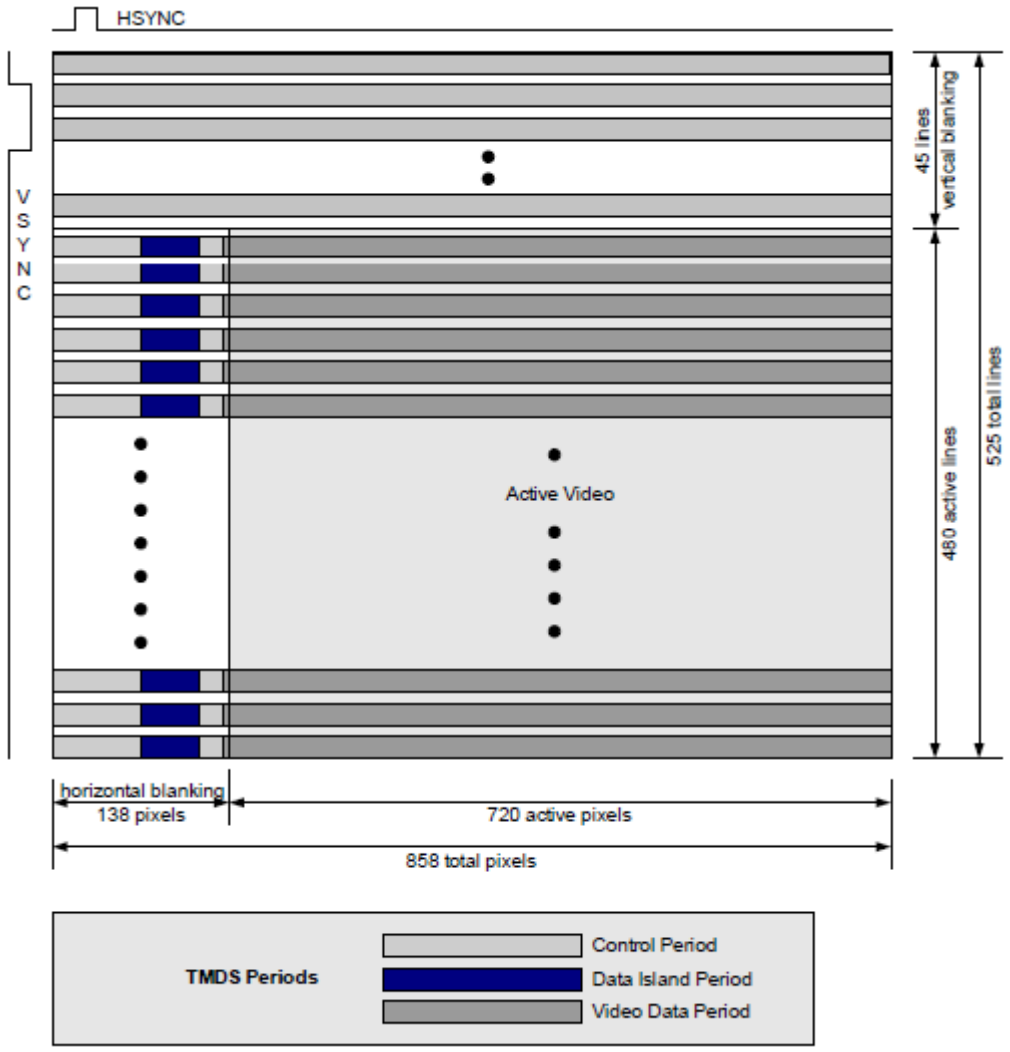


Figura 4.3. Ejemplo de envío de información en las diferentes zonas de la imagen real para una resolución de 720x480. [3]

Se ha definido, también por simplicidad, una trama fija para el envío de información fuera del blanking vertical (ver Figura 4.4).

Periodo de Control Preámbulo de Isla de Datos	Banda de Guarda Isla de Datos	Isla de Datos	Banda de Guarda Isla de Datos	Periodo de Control Preámbulo de Video Activo	Banda de Guarda Video Activo	Video Activo
56Píxels	2Píxels	32Píxels	2Píxels	44Píxels	2Píxels	720Píxels

Figura 4.4. Trama de envío de información elegida fuera de la zona de blanking vertical.

El procedimiento seguido para diferenciar los diferentes bloques se muestra en el diagrama de la Figura 4.5.

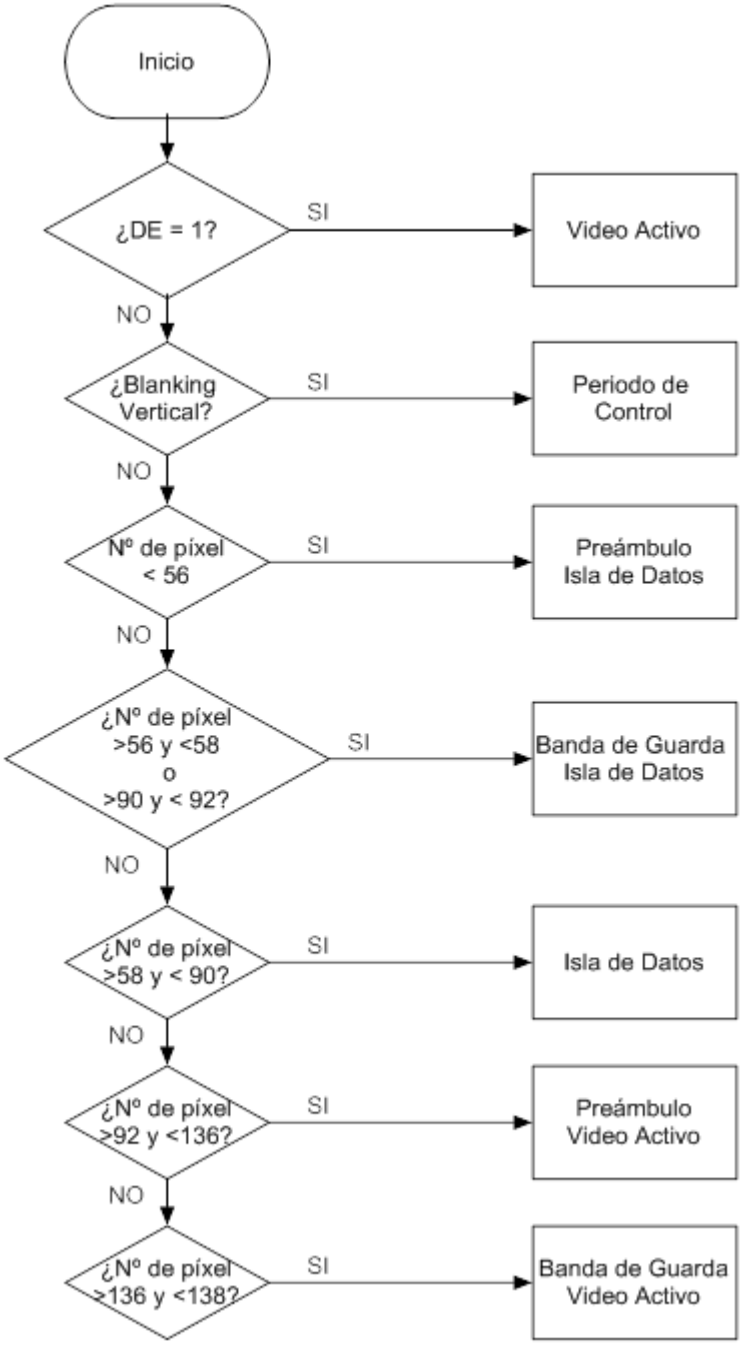


Figura 4.5. Diagrama de flujo para diferenciar bloques en módulo codificador.

Periodo de Video Activo

Se sigue el procedimiento de codificación descrito en el Capítulo 2:

- Cálculo del número de unos en el dato de entrada:

```
N1_dato := 0;
for i in 0 to 7 loop
      if (dato(i) = '1') then
            N1_dato := N1_dato + 1;
end if;
end loop;
```

- Reducción de transiciones en función del número de unos y de la disparidad del dato anterior:

```
if(N1_dato > 4 or (N1_dato = 4 and dato(0) = '0'))   then
      q_m(0) <= dato(0);
       q_m(1) <= q_m(0) xnor dato(1);
       q_m(2) <= q_m(1) xnor dato(2);
       q_m(3) <= q_m(2) xnor dato(3);
       q_m(4) <= q_m(3) xnor dato(4);
       q_m(5) <= q_m(4) xnor dato(5);
       q_m(6) <= q_m(5) xnor dato(6);
       q_m(7) <= q_m(6) xnor dato(7);
       q_m(8) <= '0';
else
       q_m(0) <= dato(0);
       q_m(1) <= q_m(0) xor dato(1);
       q_m(2) <= q_m(1) xor dato(2);
       q_m(3) <= q_m(2) xor dato(3);
       q_m(4) <= q_m(3) xor dato(4);
       q_m(5) <= q_m(4) xor dato(5);
       q_m(6) <= q_m(5) xor dato(6);
       q_m(7) <= q_m(6) xor dato(7);
      q_m(8) <= '1';
end if;
```

- Cálculo de unos y ceros en dato interno para ver si es conveniente realizar un balance DC:

```
N1_q_m := 0;
N0_q_m := 0;
for i in 0 to 7 loop
      if(q_m(i) = '1') then
            N1_q_m := N1_q_m + 1;
      end if;
end loop;
 N0_q_m := 8 - N1_q_m;
```

- Balanceo DC selectivo y cálculo de la nueva disparidad:

```
if(cnt = 0 or N1_q_m = N0_q_m) then
      q_out(9) <= not q_m(8);
      q_out(8) <= q_m(8);
       if(q_m(8) = '0') then
            q_out(7 downto 0)<=not q_m(7 downto 0);
            cnt <= cnt + (N0_q_m - N1_q_m);
      else
            q_out(7 downto 0) <= q_m(7 downto 0);
            cnt <= cnt + (N1_q_m - N0_q_m);
            end if;
      else
            if((cnt > 0 and N1_q_m > N0_q_m)or(cnt<0 and
                  N0_q_m > N1_q_m)) then
                  q_out(9) <= '1';
                  q_out(8) <= q_m(8);
                  q_out(7 downto 0)<=not q_m(7 downto 0);
                  if(q_m(8) = '0') then
                  cnt <= cnt + (N0_q_m - N1_q_m);
                  else
                  cnt<= cnt + 2 + (N0_q_m - N1_q_m);
                  end if;
            else
                  q_out(9) <= '0';
                  q_out(8 downto 0) <= q_m(8 downto 0);
            if(q_m(8) = '0') then
                  cnt<= cnt - 2 + (N1_q_m - N0_q_m);
                  else
                  cnt <= cnt + (N1_q_m - N0_q_m);
                  end if;
      end if;
```

Periodo de Control

Durante el periodo de control se realiza la asignación directa de los valores predefinidos.

```
control <= C1 & C0;
case control is
      when "00" => q_out <= "1101010100";
      when "01" => q_out <= "0010101011";
      when "10" => q_out <= "0101010100";
      when "11" => q_out <= "1010101011";
      when others => NULL;
end case;
```

Preámbulo Isla de Datos

Durante el preámbulo de la isla de datos se debe tener el cuenta el cambio de valor en la señal de control C0 del canal TMDS-2, la asignación posterior es la misma que en el periodo de control.

```
if TMDS_Channel = 2 then
      control <= "01";
else
      control <= C1 & C0;
end if;
```

Banda de Guarda de Isla de Datos

El valor de salida para el canal TMDS-0 es fruto de la codificación TERC4 de la concatenación "11VSyncHSync". Para los canales TMDS-1 y TMDS-2 se asigna el patrón fijo.

```
if TMDS_Channel = 0 then
      data_island <= "11" & C1 & C0;
            case (data_island) is   --CODIFICACION TERC4
            when "0000" => q_out <= "1010011100";
            when "0001" => q_out <= "1001100011";
            when "0010" => q_out <= "1011100100";
            when "0011" => q_out <= "1011100010";
            when "0100" => q_out <= "0101110001";
            when "0101" => q_out <= "0100011110";
            when "0110" => q_out <= "0110001110";
            when "0111" => q_out <= "0100111100";
            when "1000" => q_out <= "1011001100";
            when "1001" => q_out <= "0100111001";
            when "1010" => q_out <= "0110011100";
            when "1011" => q_out <= "1011000110";
            when "1100" => q_out <= "1010001110";
            when "1101" => q_out <= "1001110001";
            when "1110" => q_out <= "0101100011";
            when "1111" => q_out <= "1011000011";
            when others => NULL;
      end case;
else
      q_out <= "0100110011";
end if;
```

Isla de Datos

La configuración de los valores de entrada para la asignación de valores TERC4, será para el envío de la cabecera y el paquete NULL:

```
if TMDS_Channel = 0 then
      if (hcount = 58) then
            data_island <= "00" & C1 & C0;
      else
            data_island <= "10" & C1 & C0;
      end if;
else
      data_island <= "0000";
end if;
```

Preámbulo de Video Activo

Las señales de control C1 y C0 se definen por defecto para preámbulo de video activo, por lo que la asignación de valores a la salida es igual a la vista en el periodo de control.

Banda de Guarda de Video Activo

La banda de guarda de video activo se genera a partir de un patrón para canales TMDS-0 y TMDS-2 y otro para el canal TMDS-1:

```
if (TMDS_Channel = 1) then
      q_out <= "0100110011";
else
      q_out <= "1011001100";
end if;
```

4.3.2. Módulo Serializador

Realizada la codificación de los datos de entrada en el bloque anterior, disponemos de un dato de 10 bits que debemos serializar, es decir, el objetivo de este bloque es convertir un dato de entrada de 10bits paralelo en un dato de salida en serie diferencial (ver Figura 4.6).

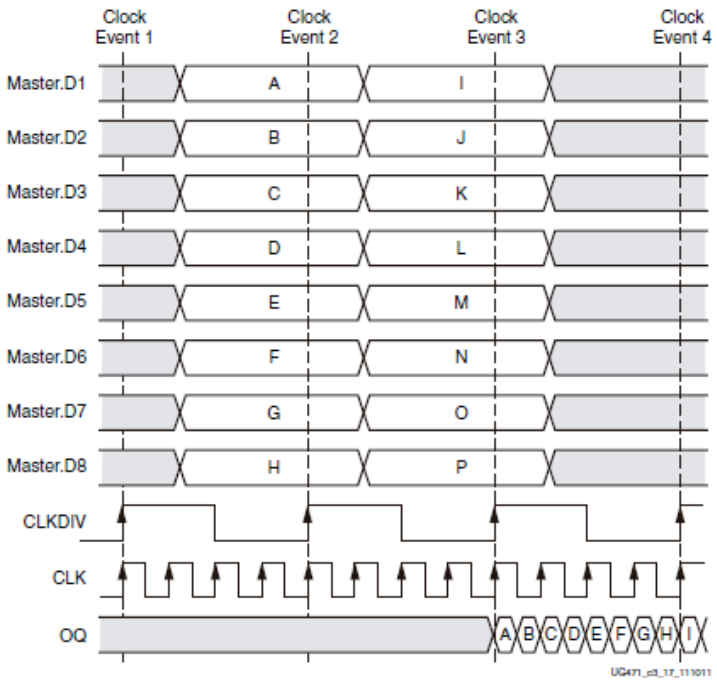


Figura 4.6. Ejemplo de flujo de datos del recurso avanzado de salida OSERDESE2 en configuración DDR 8:1. [6]

El funcionamiento de este bloque de serialización está basado en el uso del recurso avanzado de salida OSERDESE2, que haciendo uso de dos de estos bloques y mediante una configuración "Maestro/Esclavo" nos permite una salida en serie tomando como entrada un dato paralelo de hasta 14bits [6].

Para una configuración 10 a 1 como la que se desea realizar, es necesario (ver Figura 4.7):

1. Asignar el modo de funcionamiento "MASTER" o "SLAVE" según corresponda a dos bloques OSERDESE2 declarados.
2. Configurar la frecuencia de salida como "DDR" y proporcionar como entradas dos relojes sincronizados, uno para dato y otro para bit, siendo la frecuencia del reloj de bit cinco veces mayor (Pixel_CLK y DDR_CLK en nuestro caso).
3. Los puertos denominados "SHIFTOUT" del esclavo se conectan a los puertos "SHIFTIN" del maestro.
4. Los 8 bits menos significativos se conectan a las entradas D1-D8 del maestro, los 2 bits más significativos a las entradas D3 y D4 del esclavo.
5. Proporcionar una señal de reset síncrona y registrada.

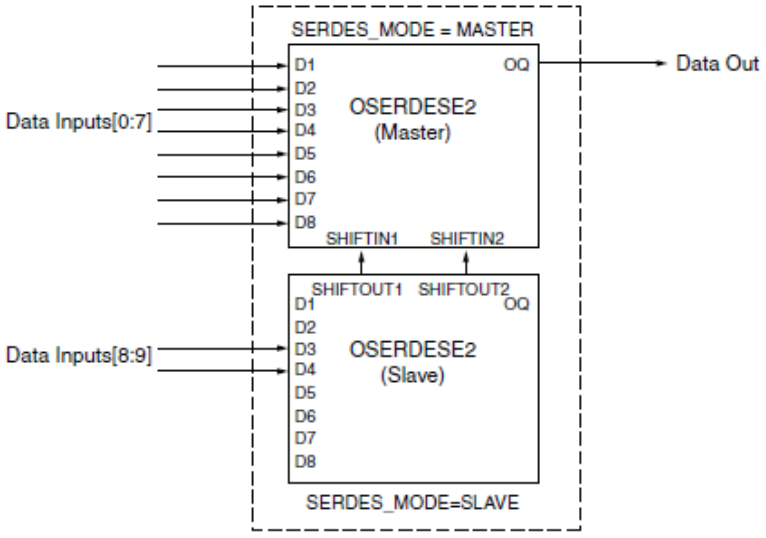


Figura 4.7. Configuración Maestro-Esclavo de recurso avanzado de salida OSERDESE2 para serialización 10 a 1. [6]

La configuración de los atributos y la asignación de entradas y salidas se realiza de la siguiente forma:

```
    Maestro : OSERDESE2
          generic map(
       DATA_RATE_OQ => "DDR",
       DATA_RATE_TQ => "SDR",
       DATA_WIDTH => 10,
       SERDES_MODE => "MASTER",
       TRISTATE_WIDTH => 1,
       TBYTE_CTL => "FALSE",
       TBYTE_SRC => "FALSE")
          port map(
       CLK => ddr_clk,
       CLKDIV => clk,
       D1 => dato(0),D2 => dato(1),D3 => dato(2),D4 => dato(3),
       D5 => dato(4),D6 => dato(5),D7 => dato(6),D8 => dato(7),
       T1 => '0',T2 => '0',T3 => '0',T4 => '0',
      TCE => '1',OCE => '1',
       TBYTEIN => '0',
       RST => reset_sincrono,
       SHIFTIN1 => shiftout1,
       SHIFTIN2 => shiftout2,
       OQ => dato_se,
       OFB => open,
       TQ => open,
       TFB => open,
       TBYTEOUT => open,
       SHIFTOUT1 => open,
       SHIFTOUT2 => open);
    Esclavo : OSERDESE2
          generic map(
       DATA_RATE_OQ => "DDR",
       DATA_RATE_TQ => "SDR",
       DATA_WIDTH => 10,
       SERDES_MODE => "SLAVE",
       TRISTATE_WIDTH => 1,
       TBYTE_CTL => "FALSE",
       TBYTE_SRC => "FALSE")
          port map(
       CLK => ddr_clk,
       CLKDIV => clk,
       D1 => '0',D2 => '0',D3 => dato(8),D4 => dato(9),
       D5 => '0',D6 => '0',D7 => '0',D8 => '0',
       T1 => '0',T2 => '0',T3 => '0',T4 => '0',
       TCE => '1',OCE => '1',
       TBYTEIN => '0',
       RST => reset_sincrono,
       SHIFTIN1 => '0',
       SHIFTIN2 => '0',
       OQ => open,
       OFB => open,
       TQ => open,
       TFB => open,
       TBYTEOUT => open,
       SHIFTOUT1 => shiftout1,
       SHIFTOUT2 => shiftout2);
```

Una vez disponemos del dato en serie, solo queda generarlo como salida diferencial, para ello se hace uso del recurso avanzado de salida OBUFDS, cuya configuración necesaria para disponer de una señal diferencial acorde al estándar TMDS se consigue mediante la configuración de salida por defecto y un configuración de ritmo rápido. La configuración de este atributo y la asignación de entradas y salidas se realiza de la siguiente forma:

```
    Salida_diferencial : OBUFDS
          generic map (
          IOSTANDARD => "DEFAULT",
                SLEW => "FAST")
          port map (
          I => clk_interno,
          O => tmds_clk(1),
          OB => tmds_clk(0));
```

Para un correcto funcionamiento del OSERDESE2, la señal de reset a la entrada se debe registrar y debe estar sincronizada con la señal de reloj. Para ello generamos un proceso dependiente de la señal de reloj de la siguiente forma:

```
process(clk)
    begin
    if (clk'event and clk = '1') then
          reset_sincrono <= reset;
    end if;
    end process;
```

A modo de resumen de este apartado, se muestra el diagrama de bloques del módulo (ver Figura 4.8).

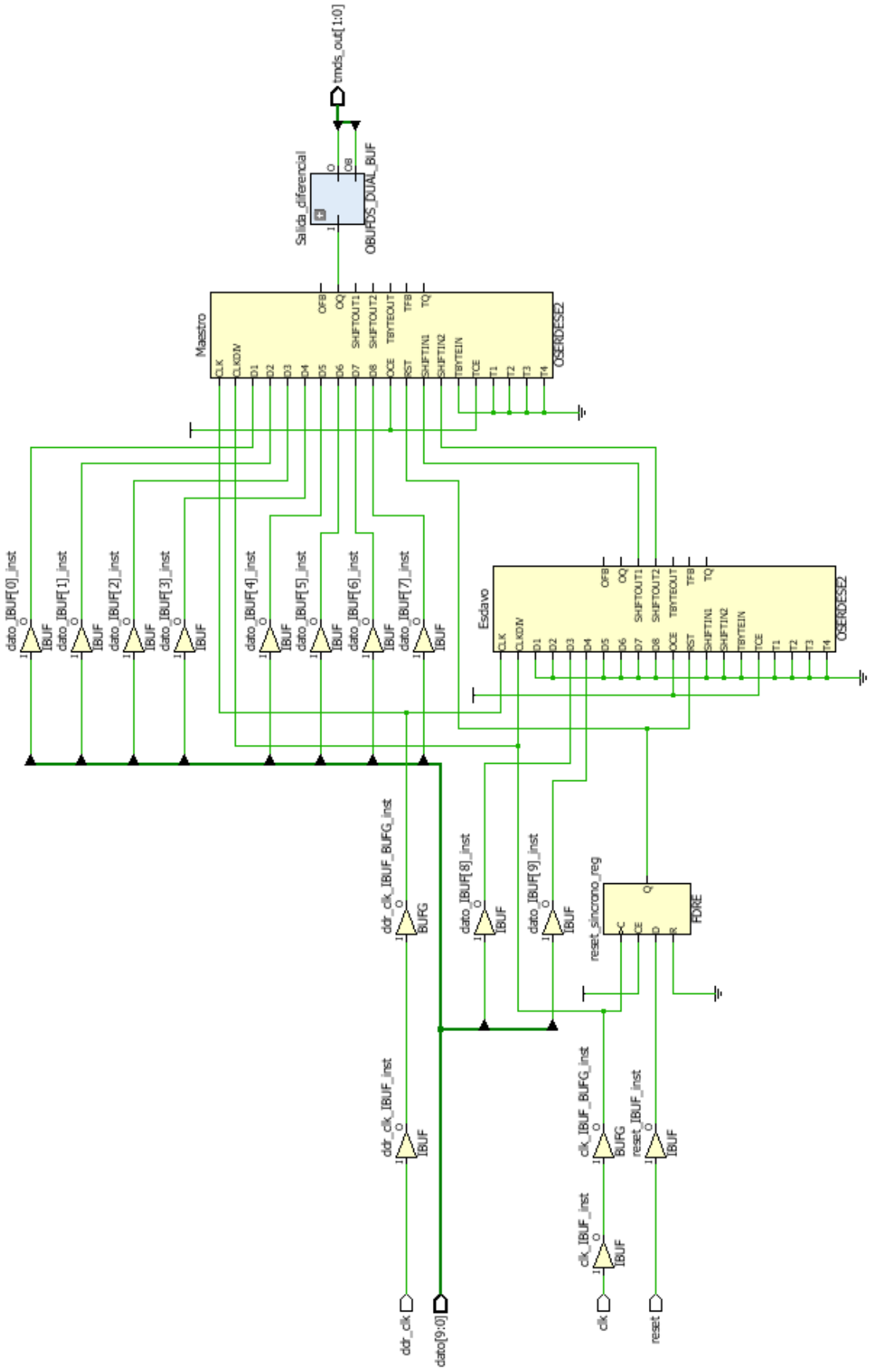


Figura 4.8. Diagrama de bloques del módulo serializador.

4.3.3. Módulo Generador de TMDS-CLK

La generación de la señal transmitida por el canal TMDS-CLK se realiza tomando como dato de entrada la señal de reloj de dato (no de bit). El objetivo de este bloque es proporcionar a la salida de nuestro *core* la señal de reloj de dato de forma diferencial, que debe estar perfectamente sincronizada tanto en flanco de subida como de bajada con los datos de salida serie de los otros tres canales TMDS. Para ello, se realiza un registro de salida y se convierte en señal diferencial haciendo uso de los recursos avanzados de salida ODDR y OBUDS (ver Figura 4.9).

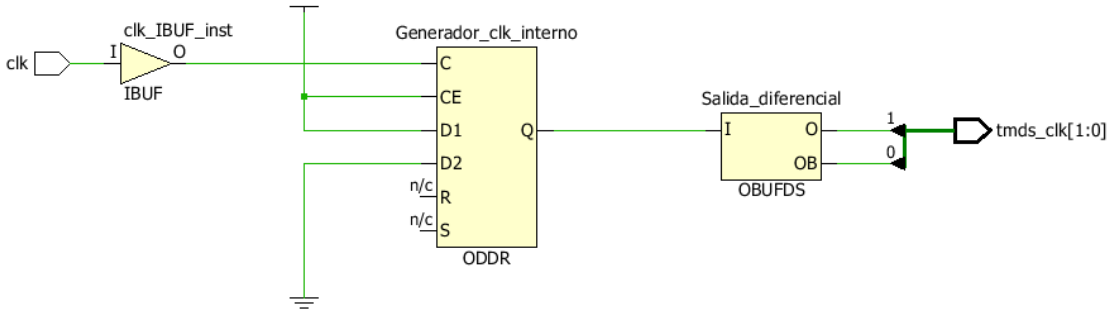


Figura 4.9. Diagrama de bloques del módulo generador de TMDS_CLK

En la configuración del ODDR, se asignará al atributo "DDR_CLK_EDGE", que define el funcionamiento según queramos realizar el registro de los datos en el mismo flanco de reloj o en el opuesto, el valor por defecto "OPPOSITE EDGE", valor inicial de la salida '0' y los set y reset síncronos (aunque en nuestro caso no se utilizan). La configuración de estos atributos y la asignación de entradas y salidas se realiza de la siguiente forma:

```
    Generador_clk_interno    : ODDR
          generic map (
       DDR_CLK_EDGE => "OPPOSITE_EDGE",
       INIT => '0',
       SRTYPE => "SYNC")
          port map (
       Q   => clk_interno,
       C   => clk,
       CE  => '1',
       D1  => '1',
       D2  => '0',
       R   => '0',
       S   => '0');
```

Al igual que en el bloque anterior, en la configuración del OBUFDS para obtener la salida diferencial deseada, solo es necesario la definición del atributo de entrada/salida por defecto y ritmo rápido.

4.3.4. Core

El módulo *core* se compone de las diferentes llamadas al resto de módulos anteriormente descritos. Se realizará una llamada al Codificador y otra al Serializador por cada canal RGB. A modo de ejemplo, se muestra la llamada para generar el canal TMDS-0:

```
Codificador_B : entity work.codificador_tmds
    port map(
          clk               => pixel_clock,
          reset             => reset,
          den               => den,
          dato              => pixel_data(7 downto 0),
          C0                => hsync,
          C1                => vsync,
          TMDS_Channel      => 0,
          q_out => tmds_cod0);
Serializador_B : entity work.serializador_tmds
    port map(
          clk         => pixel_clock,
          ddr_clk     => ddr_bit_clock,
          reset       => reset,
          dato        => tmds_cod0,
          tmds_out    => tmds_d0);
```

Por último se realiza la llamada al módulo Generador de TMDS-CLK:

```
TMDS_CLK_generador : entity work.generador_clk_tmds
    port map(
          clk      => pixel_clock,
          tmds_clk => tmds_clk);
```

4.3.5. Integración en banco de pruebas

Se realiza el diseño de un módulo que sirva para testear el funcionamiento del *core* desarrollado (ver Figura 4.10).

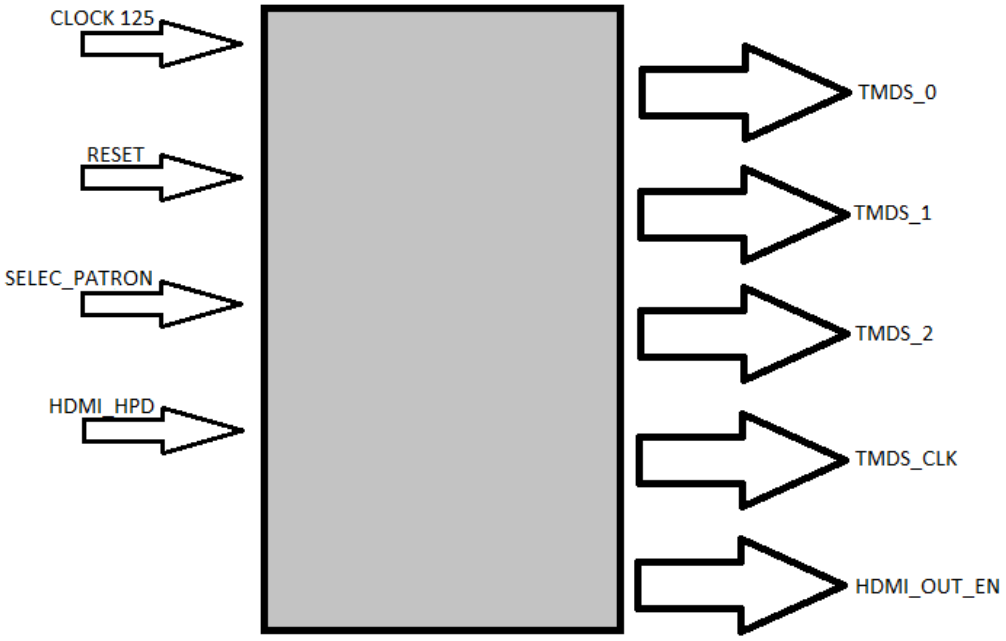


Figura 4.10. Esquema general de entradas y salidas en banco de pruebas.

Este módulo debe ser capaz de generar las señales necesarias para realizar la llamada al controlador, la cual se realizará de la siguiente forma:

```
HDMI_CORE : entity work.CORE_HDMI
   port map(
       pixel_clock      => hdmi_pixel_clock,
       ddr_bit_clock    => hdmi_ddr_clock,
       reset            => reset,
       den              => hdmi_den,
       hsync            => hdmi_hsync,
       vsync            => hdmi_vsync,
       pixel_data       => hdmi_data,
       tmds_clk         => hdmi_clk,
       tmds_d0          => hdmi_d0,
       tmds_d1          => hdmi_d1,
       tmds_d2          => hdmi_d2);
```

Generación de señales de reloj

Es requisito del controlador disponer a su entrada de dos señales de reloj de 27MHz y 135MHz para el pixel_clock y ddr_bit_clock respectivamente.

Estos valores son los necesarios para conseguir una frecuencia de barrido de 60Hz para la resolución de 720x480 elegida. El total de píxeles a recorrer será 858x525 (como se ha visto en capítulos anteriores), obteniendo un total de 450450 píxeles. Si multiplicamos por la frecuencia de 60Hz, obtenemos los 27MHz necesarios en pixel_clock. Debido a que serializamos un dato de salida

de 10 bits, el ddr_bit_clock debe ser 5 veces mayor, es decir, su valor será de 135MHz.

Para ello, haremos uso del LogiCore IP Clocking Wizard descrito en el Capítulo 3.

Tomaremos como reloj referencia el reloj de 125MHz interno de la ZYBO directamente conectado al PIN L16 y configuraremos el Clocking Wizard como MMCM, ajustando dos señales de reloj de salida a las frecuencias deseadas.

Se añade a nuestro proyecto localizando la IP en el catálogo, se configura y se declara. Por último, se asignan los puertos:

```
component clk_wiz_0 is
      port(
     clk_in   : in std_logic;
     rst      : in std_logic;
     px_clk   : out std_logic;
     ddr_clk  : out std_logic);
end component;
MMCM : clk_wiz_0
      port map(
     clk_in       => CLOCK_125,
     rst          => reset,
     px_clk       => hdmi_pixel_clock,
     ddr_clk      => hdmi_ddr_clock);
```

Reset

Se configura como reset del sistema el interruptor pulsador BTN0 del PIN R18, que se sincronizará con la señal pixel_clock.

Señales de sincronismo vertical y horizontal

Se generan las señales de sincronismo de video horizontal y vertical basándonos en los valores del estándar de HDTV para la resolución de 720x480 (ver Figura 4.11).

```
    total_px          <= 858;
    total_lin         <= 525;
    h_sync_tamaño     <= 62;
    v_sync_tamaño     <= 6;
    hbp               <= 60;
    hfp               <= 16;
    vbp               <= 30;
    vfp               <= 9;
    visible_px        <= 720;
    visible_lin       <= 480;
    if hcount >= hfp and hcount < h_sync_tamaño + hfp-1  then
          hdmi_hsync <= '1';
     else
          hdmi_hsync <= '0';
    end if;
    if vcount >= vfp and vcount < v_sync_tamaño + vfp-1  then
          hdmi_vsync <= '1';
    else
          hdmi_vsync <= '0';
    end if;
```

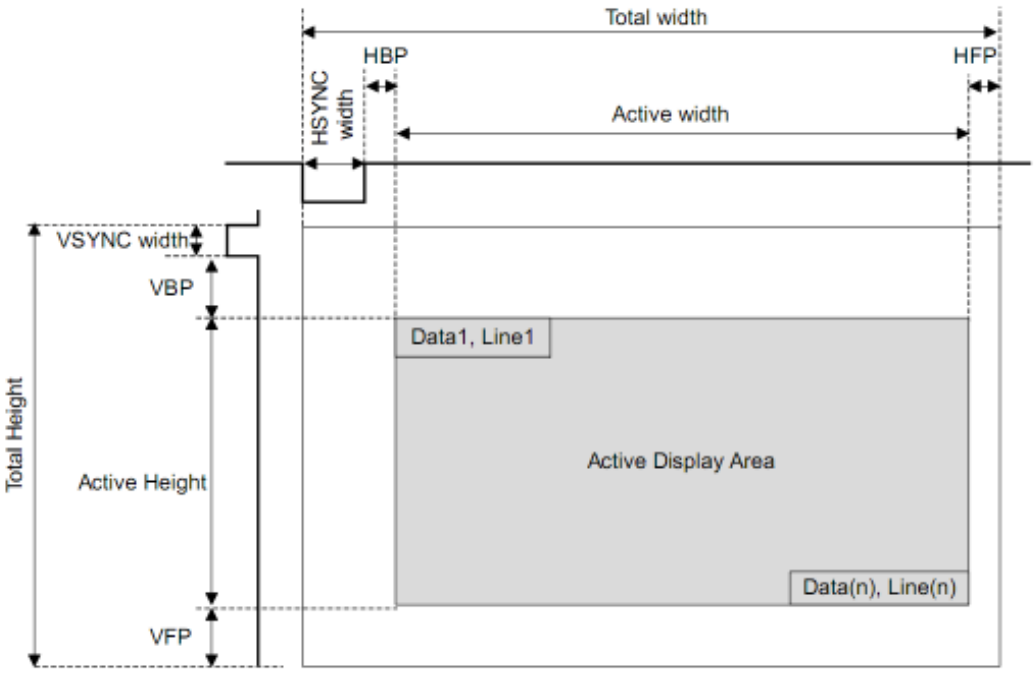


Figura 4.11. Ejemplo de señales de sincronismo vertical y horizontal. [7]

Den

El puerto DEN del *core* habilita el periodo de video activo. Se hace uso de dos señales que nos indicarán los márgenes horizontal y vertical entre los blankings y la zona de video activa.

```
-- Zona activa
if (hcount>h_sync_tamaño+hfp + hbp-1)and(hcount<total_px-1) then
    hdmi_hact <='1';
else
    hdmi_hact <='0';
end if;
if (vcount>v_sync_tamaño+vfp+vbp-1) and vcount<(total_lin-1) then
    hdmi_vact <='1';
else
    hdmi_vact <='0';
end if;
```

Generación de patrones para pixel data

Se generan dos patrones seleccionables mediante el interruptor selector SW0 (PIN G15). El patrón de barras para test de video más común,si SW0 igual a '1' (ver Figura 4.12) y un patrón con formas rectangulares para una mejor verificación del sincronismo tanto vertical como horizontal, si SW0 igual a '0' (ver Figura 4.13).

Se muestra el código del patrón de barras, como generar el patrón con formas rectangulares se puede ver en el Anexo 1 de este documento.

```
-- Bar pattern test
    if (hcount >= 138 and hcount < 228) then
          hdmi_data<="111111111111111111111111";    -- Blanco
    elsif (hcount >= 228 and hcount < 318) then
          hdmi_data<="111111111111111100000000";    -- Amarillo
    elsif (hcount >= 318 and hcount < 408) then
          hdmi_data<="000000001111111111111111";    -- Cian
    elsif (hcount >= 408 and hcount < 498) then
          hdmi_data<="000000001111111100000000";    -- Verde
    elsif (hcount >= 498 and hcount < 588) then
          hdmi_data<="111111110000000011111111";    -- Magenta
    elsif (hcount >= 588 and hcount < 678) then
          hdmi_data <= "111111110000000000000000"; -- Rojo
    elsif (hcount >= 678 and hcount < 768) then
          hdmi_data<="000000000000000011111111";    -- Azul
    else
          hdmi_data<="000000000000000000000000";    -- Negro
    end if;
```

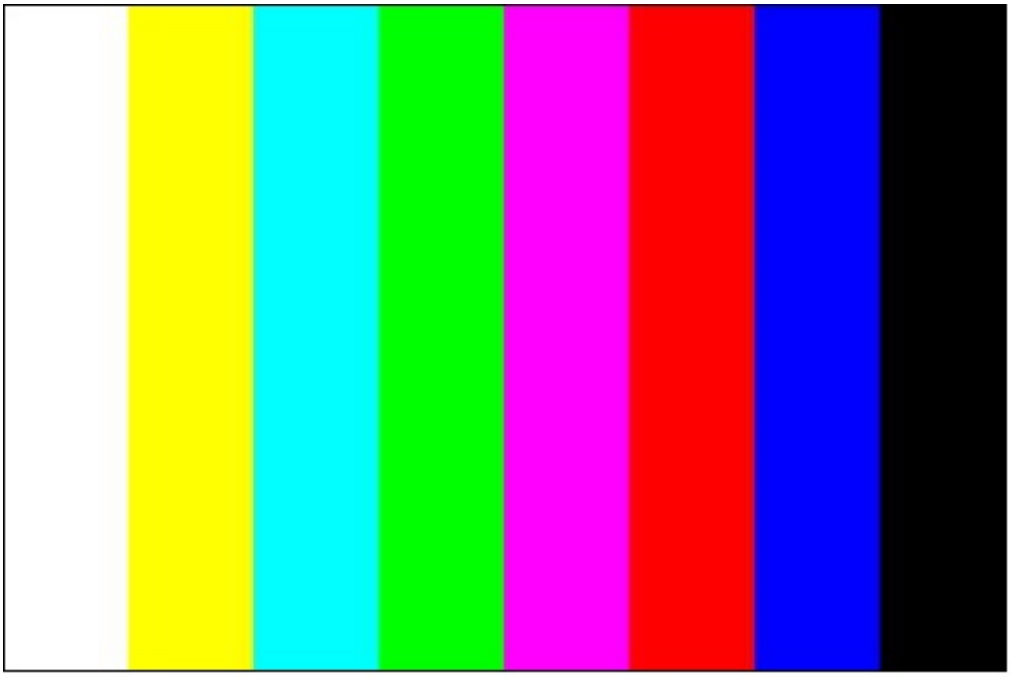


Figura 4.12. Patrón de video 1.

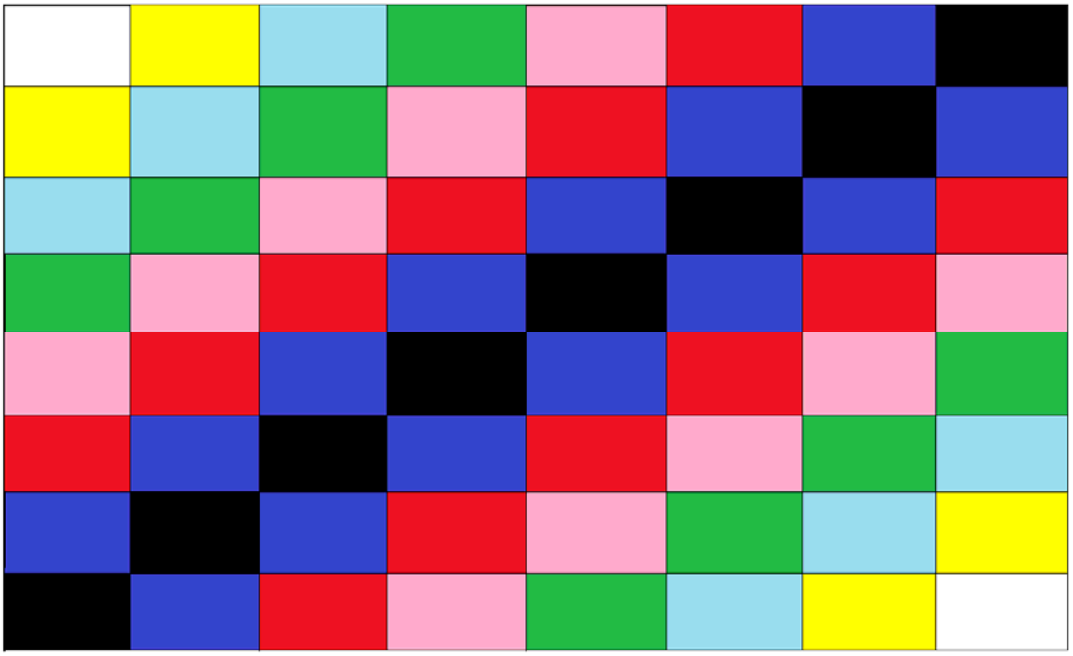


Figura 4.13. Patrón de video de cuadros de colores.

HDMI_OUT_EN y HDMI_HPD

Para la sincronización del transmisor y el receptor, el HDMI genera las señales iniciales descritas en el Capítulo 2 (señal de 5V y señal HPD). Interconectaremos una con otra para la confirmación directa de la recepción de la señal de 5V. La asignación de pines en la ZYBO será:

- HDMI_OUT_EN: F17
- HDMI_HPD: E18

Capítulo 5. Fase de pruebas

El principal procedimiento seguido para la verificación de los diferentes módulos que componen el Proyecto, así como la interconexión entre ellos, ha sido la generación de simulaciones y análisis de los cronogramas fruto de estas en la plataforma Vivado.

En la fase de pruebas final, se realiza la conexión a un monitor externo y se visualizan los patrones generados.

5.1. Descripción del entorno de pruebas

La plataforma de desarrollo Vivado ofrece la posibilidad de realizar simulaciones de funcionamiento de los diseños. Estas simulaciones se generan en el "Modo Simulación", asignando valores a las entradas de los módulos y ajustando la temporización. El resultado obtenido será un cronograma virtual sobre el que podremos navegar, localizar errores, comprobar la correcta temporización de las señales y verificar que los datos de salida son acordes a los valores de entrada pre-seleccionados.

5.2. Procesos de verificación

Siguiendo la metodología incremental descrita en el Capítulo 4, después del diseño de cada módulo se ha realizado una verificación de su funcionamiento mediante simulación Behavioral (RTL).

5.2.1. Verificación del Módulo Codificador

Se realizan diferentes simulaciones para comprobar el funcionamiento de todos los bloques que componen el módulo codificador.

Se van a exponer solo algunas de las pruebas realizadas, ya que el procedimiento a seguir es el mismo para todas.

Verificación de Periodo de Video Activo

La primera de las pruebas realizadas, se corresponde con la zona de video activo. La configuración de las entradas elegida en este caso será:

- CLK: Señal de reloj de 27MHz.
- DEN: '1' para video activo.
- Reset: Señal de reloj de menor frecuencia que nos permita visualizar los datos que deseamos, en este caso se asigna un periodo de 100000ps.
- C0 y C1: '0' por defecto.
- Dato: "10101010", nos va a permitir ver la reducción de las transiciones y el balanceo DC.

- Canal TMDS: 2 (no afecta en este caso).

Los comandos Vivado para esta configuración serán:

```
add_force {/codificador_tmds/clk} -radix bin {0 0ns} {1 18515ps} -
repeat_every 37030ps
add_force {/codificador_tmds/den} -radix bin {1 0ns}
add_force {/codificador_tmds/reset} -radix bin {1 0ns} {0 50000ps}
-repeat_every 10000000ps
add_force {/codificador_tmds/C0} -radix bin {0 0ns}
add_force {/codificador_tmds/C1} -radix bin {0 0ns}
add_force {/codificador_tmds/dato} -radix bin {10101010 0ns}
add_force {/codificador_tmds/TMDS_Channel} -radix unsigned {2 0ns}
run 100 ms
```

Siguiendo el flujo de codificación descrito en capítulos anteriores, cuando se analizaba la codificación de video, nos encontraríamos en el caso en el que el número de unos a la entrada es 4 y el LSB '0', con lo que el MSB del dato interno será '0', el LSB coincidirá con el del dato de entrada y el resto de valores serán fruto de operaciones XNOR, obteniendo un valor de dato intermedio: "011001100", donde se aprecia a simple vista la reducción de las transiciones.

Para calcular el valor de salida, al ser la disparidad anterior '0' (valor inicial), el MSB de salida será igual al MSB del dato interno negado, el siguiente bit será igual al MSB de dato interno, el resto, al ser el MBS '0', será la inversión del resto de bits, obteniendo el valor "1000110011".

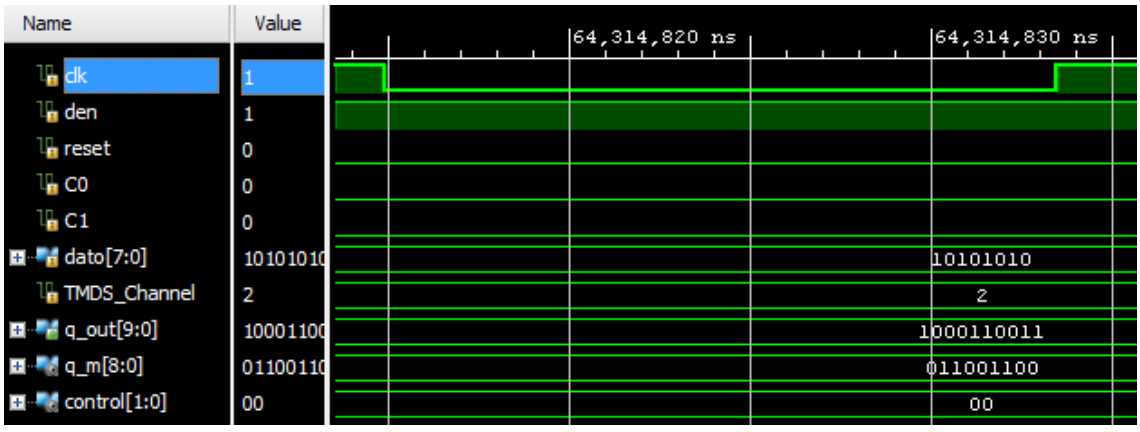


Figura 5.1. Cronograma para verificación de señal de dato intermedio y dato de salida en módulo codificador en periodo de video activo.

Verificación de Periodo Blanking

Con la misma configuración anterior a excepción de la entrada DEN (para situarnos fuera de la zona de video activo), podemos comprobar el comportamiento del sistema en la zona de blanking.

Los comandos son los mismos que los utilizados en la verificación de la zona de video activo, con la variación de DEN a nivel bajo:

```
add_force {/codificador_tmds/den} -radix bin {0 0ns}
```

Se puede observar en la figura 5.2 como la señal de salida toma los valores acordes a la zona de control, tomando como entradas las señales de control C0 y C1 a '0', es decir, se asigna el valor "1101010100" a la salida.

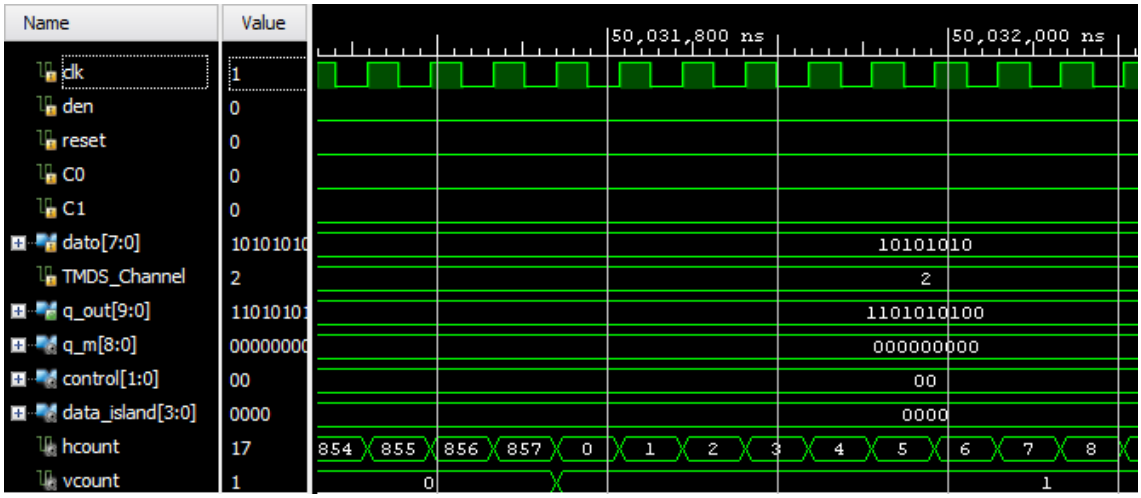


Figura 5.2. Cronograma para verificación de señal de salida en módulo codificador en periodo de control en blanking vertical.

Desplazándonos sobre mismo cronograma generado en la simulación anterior hasta que finalice el periodo de blanking vertical, podremos verificar, sin realizar modificaciones, la trama enviada durante el blanking horizontal.

Recordemos el orden de aparición de las diferentes salidas:

1. Periodo control preámbulo de isla de datos (56 píxeles)
2. Banda de guarda de isla de datos (2 píxeles)
3. Isla de datos (32 píxeles)
4. Banda de guarda de isla de datos (2 píxeles)
5. Periodo de control preámbulo de video activo (44 píxeles)

6. Banda de guarda de video activo (2 píxeles)

Se muestra a modo de ejemplo la transición entre los tres primeros periodos en la figura 5.3, observando los siguientes valores a la salida:

- Preámbulo de isla de datos para canal 2, implica asignación de valores de control con entrada "01", obteniendo como resultado "0010101011".
- Banda de guarda de isla de datos para canal 2, se realiza asignación directa de patrón "0100110011".
- Isla de datos con paquete NULL para canal 2, se asigna mediante codificación TERC4 la salida para entrada "0000", obteniendo el valor "1010011100".

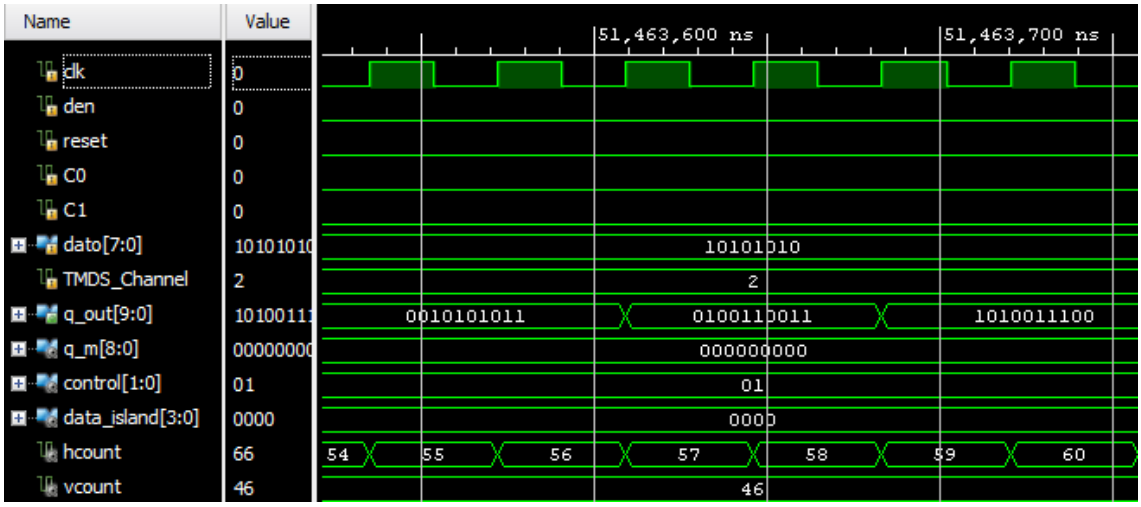


Figura 5.3. Cronograma para verificación de señal de salida en módulo codificador en periodo de blanking horizontal.

5.2.2. Verificación del Módulo Serializador

Siguiendo el mismo procedimiento, se procede a la verificación del módulo serializador.

Se asignan los siguientes valores de entrada:

- CLK: Señal de reloj de píxel de 27MHz.
- DDR_CLK: Señal de reloj de bit de 135MHz.
- Reset: Señal de reloj de menor frecuencia que nos permita visualizar los datos que deseamos, en este caso se asigna un periodo de 100000ps.

- Dato de entrada: "0010101011".

Se verifica la serialización a la salida y el resultado del buffer diferencial (ver figura 5.5).

Se debe tener en cuenta que al realizar la simulación y fijar un dato de entrada, no existe ninguna sincronización entre el dato de entrada y la señal de reloj de píxel, esto se traslada a la señal de salida ya el ritmo de salida lo marca la señal de reloj de bit.

Una vez que se integra este módulo en el *core*, el dato llegará sincronizado con el reloj de píxel y no se producirá es desfase que se observa en la figura 5.4.

Los comandos de asignación de entrada serán en este caso:

```
add_force {/serializador_tmds/clk} -radix bin {0 0ns} {1 18515ps}
-repeat_every 37030ps
add_force {/serializador_tmds/ddr_clk} -radix bin {0 0ns} {1
3703ps} -repeat_every 7406ps
add_force {/serializador_tmds/reset} -radix bin {1 0ns} {0
50000ps} -repeat_every 1000000ps
add_force {/serializador_tmds/dato} -radix bin {0010101011 0ns}
run 100 ms
```

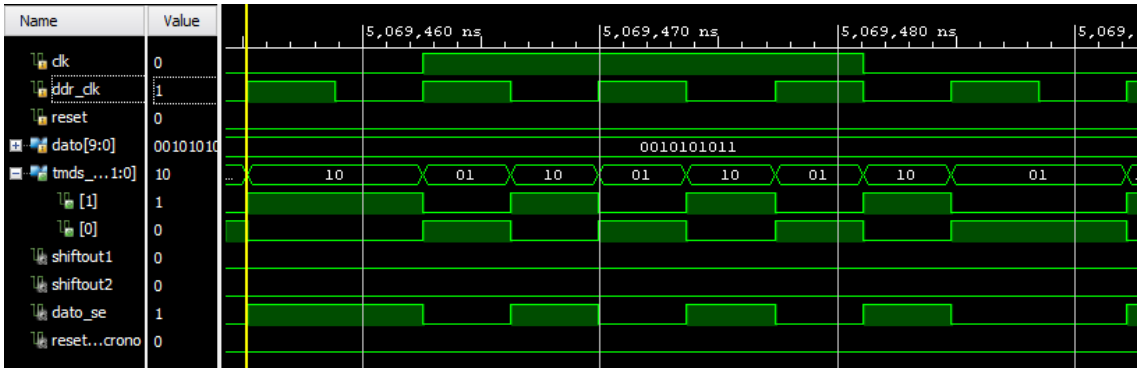


Figura 5.4. Ejemplo de simulación para verificación de módulo serializador.

5.2.3. Verificación del Módulo Generador de TMDS-CLK

En este caso, solo tendremos que indicar la entrada de la señal de reloj de píxel de 27MHz. Observamos en la figura 5.5. la correcta generación de la señal del canal TMDS-CLK.

El comando de asignación de la señal de entrada, como se ha visto anteriormente será:

```
add_force {/generador_clk_tmds/clk} -radix bin {0 0ns} {1 18515ps}
-repeat_every 37030ps
run 200 ms
```

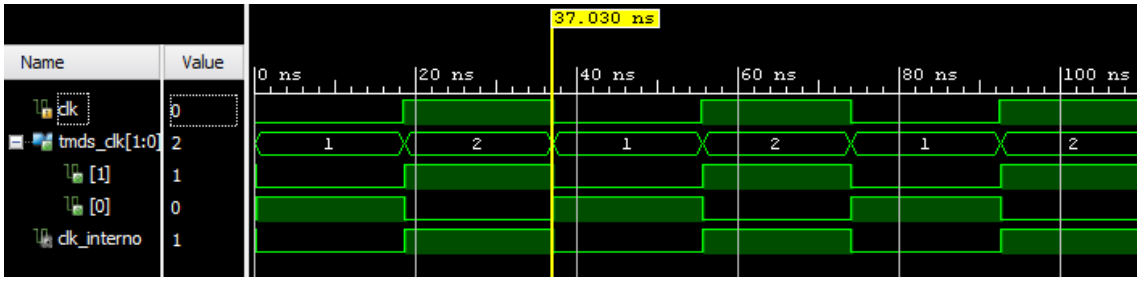


Figura 5.5. Simulación para verificación de módulo generador de TMDS-CLK

5.2.4. Verificación del sistema

En este apartado se muestra como realizar la verificación en conjunto de las señales de video generadas en el módulo de test y de las diferentes llamadas realizadas por el *core*.

Para ello, basta con una sola simulación que genere un cronograma con toda la información que necesitamos. Se deberán definir las siguientes entradas:

- Señal de reloj entrada al sistema de 125MHz.
- Reset: Señal de reloj suficientemente grande para ver la información deseada.
- Selección de Patrón: '1' para barras de colores o '0' para cuadros.

Los comandos necesarios serán:

```
add_force {/TEST/CLOCK_125} -radix bin {0 0ns} {1 4000ps} -
repeat_every 8000ps
add_force {/TEST/reset} -radix bin {1 0ns} {0 50000000000ps} -
repeat_every 100000000000ps
add_force {/TEST/selec_patron} -radix bin {1 0ns}
add_force {/TEST/hdmi_hpd} -radix bin {1 0ns}
```

A modo de muestra, podemos extraer la siguiente información de la simulación realizada:

- Visualización del blanking vertical, zona de video activo y señal de sincronismo vertical en un periodo completo del barrido del monitor.

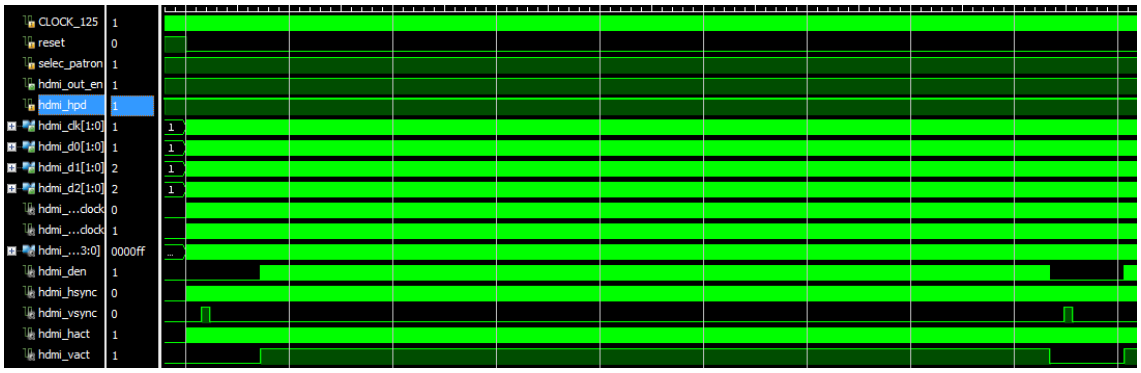


Figura 5.6. Simulación para verificación de banco de pruebas y *core*, zona blanking vertical, Periodo de Video Activo y señal de sincronismo vertical de video.

- Visualización de señal de sincronismo horizontal.

Figura 5.7. Simulación para verificación de banco de pruebas y *core*, señal de sincronismo horizontal.

- Visualización de canales TMDS para comprobación del sincronismo.

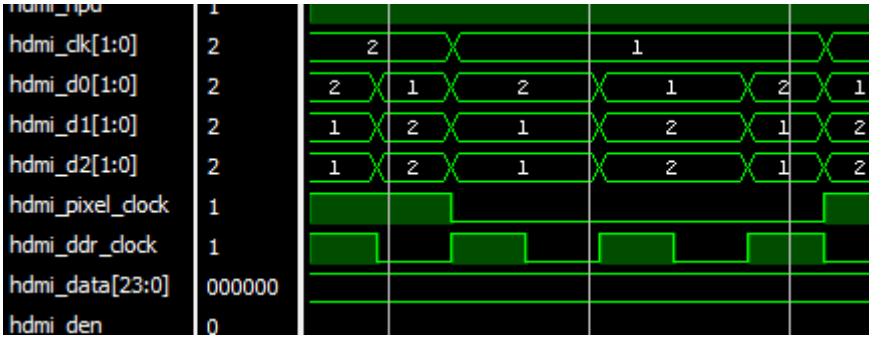


Figura 5.8. Simulación para verificación de banco de pruebas y *core*, sincronismo de canales TMDS.

Por último, se realiza la configuración del PINOUT de la ZYBO y se comprueba el funcionamiento completo del sistema conectando al monitor externo mediante el cable de interconexión HDMI.

Capítulo 6. Conclusiones y líneas futuras de trabajo

Para concluir, en este último capítulo se exponen las conclusiones obtenidas durante el desarrollo de este Proyecto. Se concluirá proponiendo posibles líneas de investigación y desarrollo a partir del controlador diseñado.

6.1. Conclusiones

Una vez diseñado y verificado el sistema, se consideran superados los objetivos marcados al inicio de este Proyecto:

- Se ha realizado con éxito la codificación de señal de video y generación y codificación de señales de control y auxiliares. Podemos decir que ha sido la parte que ha necesitado de un mayor periodo de desarrollo teórico debido a las múltiples posibilidades de codificación, se ha necesitado consulta y estudio constante de documentación. Además, durante el desarrollo de este módulo se ha adquirido soltura a la hora de trabajar con la plataforma de desarrollo Vivado, haciendo más ameno el resto del desarrollo.
- Se ha conseguido serializar un dato paralelo de 10 bits. Gracias a los recursos avanzados de salida, el desarrollo del Módulo Serializador ha sido más sencillo de lo esperado, el estudio y uso del módulo OSERDESE2 en su configuración maestro-esclavo ha simplificado considerablemente un módulo que, a priori, iba a necesitar de un período de desarrollo amplio.
- Se ha generado una señal de sincronismo diferencial acorde a los requisitos del estándar TMDS para canal de sincronización. Gracias al conocimiento adquirido durante el desarrollo de los módulos anteriores, generar la señal de sincronismo no supuso un trabajo excesivo.
- Se han integrado los módulos anteriores en un módulo *core* con sus respectivas llamadas a los módulos necesarios. Debido a que había que generar 3 canales para la información RGB, se realizó una llamada a los Módulos Codificador y Serializador por cada canal. Además, hubo que tener en cuenta los valores de las señales de control C0 y C1 para cada canal.
- Se ha diseñado un banco de pruebas capaz de generar las señales de video necesarias para la verificación del *core*. Ha sido necesario el estudio de la sincronización de video, siendo la parte más importante del

banco de pruebas, ya que el interfaz HDMI requiere del sincronismo perfecto de todas las señales de salida.

- Por último, se realizó la conexión a monitor externo, consiguiendo la visualización de los patrones en la pantalla del monitor.

6.2. Líneas futuras de trabajo

Debido al incremento de la comercialización del interfaz HDMI, habiéndose convertido en uno de los interfaces multimedia líderes en el mercado La realización de este Proyecto puede servir como preámbulo a un amplio abanico de posibilidades que podrían ser motivo de estudio.

Se proponen posibles mejoras y complementos al controlador desarrollado:

- Flexibilizar el controlador para funcionar a diferentes frecuencias y resoluciones. Todo el desarrollo se ha realizado para una resolución fija, se puede modificar el *core* para flexibilizarlo y mejorar así sus prestaciones.
- Desarrollar controlador de interfaz para recepción de video HDMI y su visualización en una pantalla (sin controlador propio).
- Incluir audio mediante el envío de paquetes en las islas de datos. Se ha desarrollado la parte de transmisión de video y generado islas de data modo de ejemplo con paquete NULL, pero existen numerosos paquetes motivo de estudio para envío de audio y señales auxiliares.
- Integrar nuestro controlador como bloque parte de un sistema más complejo.
- Desarrollar controlador para sensor/cámara que genere el dato de entrada al sistema. Se podría hacer uso de la parte PS de la ZYBO para el tratamiento de señales.

Apéndice A. Código del Sistema

Se incluye el código desarrollado en lenguaje de descripción hardware VHDL de cada uno de los módulos diseñados.

A.1. Código del Banco de Pruebas

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
library UNISIM;
use UNISIM.VComponents.all;

-- TEST para CORE H.D.M.I.

entity TEST is
    Port(
       CLOCK_125: in std_logic;
-- CLK referencia de 125MHz. Pin L16 IOSTANDARD LVCMOS33
       reset: in std_logic := '1';
-- Reset BTN0 Pin R18 IOSTANDARD LVCMOS33
       selec_patron: in std_logic := '0';
-- Selección de patrón SW0 Pin G15 IOSTANDARD LVCMOS33
       hdmi_out_en : out std_logic := '1';
-- Pin F17 IOSTANDARD LVCMOS33
       hdmi_hpd: in std_logic := '1';
-- Pin E18 IOSTANDARD LVCMOS33
       hdmi_clk: out std_logic_vector(1 downto 0) :="00";
-- Pin H16 (P) H17 (N) IOSTANDARD TMDS_33
       hdmi_d0: out std_logic_vector(1 downto 0) :="00";
-- Pin D19 (P) D20 (N) IOSTANDARD TMDS_33
       hdmi_d1: out std_logic_vector(1 downto 0) :="00";
-- Pin C20 (P) B20 (N) IOSTANDARD TMDS_33
       hdmi_d2: out std_logic_vector(1 downto 0) :="00");
-- Pin B19 (P) A20 (N) IOSTANDARD TMDS_33
end TEST;
```

```
architecture arq_TEST of TEST is
    signal hdmi_pixel_clock, hdmi_ddr_clock: std_logic :='0';
    signal hdmi_data : std_logic_vector(23downto0):=
"000000000000000000000000";
-- Dato de entrada RGB de 24 bits, se genera con patrones
    signal hdmi_den, hdmi_hsync, hdmi_vsync,hdmi_hact,hdmi_vact
: std_logic := '0';
    signal hcount,vcount, total_px, total_lin, h_sync_tamaño,
    v_sync_tamaño, hbp, hfp, vbp, vfp, visible_px, visible_lin
: natural range 0 to 860 :=0;

-----------------------------------------------------------------
component clk_wiz_0 is
-- MMCM para generación de señales CLK
    port(
        clk_in   : in std_logic;
        rst      : in std_logic;
        px_clk   : out std_logic;
        ddr_clk  : out std_logic);
end component;
-----------------------------------------------------------------

begin
    hdmi_out_en <= hdmi_hpd;
-----------------------------------------------------------------
    MMCM : clk_wiz_0
port map(
        clk_in       => CLOCK_125,
        rst          => reset,
        px_clk       => hdmi_pixel_clock,
        ddr_clk      => hdmi_ddr_clock);
-----------------------------------------------------------------
    process (hdmi_pixel_clock, hdmi_hsync, hdmi_vsync, hdmi_hact,
hdmi_vact)
      begin
-- Sincronismo y testeo reset
         if hdmi_pixel_clock='1' and hdmi_pixel_clock'event then

               if reset = '1' then
                     hcount      <= 0;
                     vcount      <= 0;
                     hdmi_hsync  <='0';
                     hdmi_vsync  <='0';
                     hdmi_hact   <='0';
                     hdmi_vact   <='0';
               else
```

```
-----------------------------------------------------------------
-- 720x480 (138 h_blanking y 45 v_blanking)
                total_px <= 858;
                 total_lin <= 525;
                h_sync_tamaño <= 62;
                v_sync_tamaño <= 6;
                hbp <= 60;
                hfp <= 16;
                vbp <= 30;
                vfp <= 9;
                visible_px <= 720;
                visible_lin <= 480;
-----------------------------------------------------------------
-- Recorrido de los pixels de la pantalla
               if hcount = (total_px - 1) then
                  hcount <= 0;
                  if vcount = (total_lin - 1) then
                      vcount <= 0;
                  else
                      vcount <= vcount + 1;
                  end if;
                else
                    hcount <= hcount + 1;
                end if;
-----------------------------------------------------------------
-- Zona activa
               if (hcount >  h_sync_tamaño + hfp + hbp -1) and
               (hcount < total_px) then -- Zona activa
                    hdmi_hact <='1';
                else
                    hdmi_hact <='0';
                end if;
                if (vcount > v_sync_tamaño + vfp + vbp - 1) and
               vcount < (total_lin) then
                    hdmi_vact <='1';
                else
                    hdmi_vact <='0';
                end if;
-----------------------------------------------------------------
-- Generación de señales de sincronismo horizontal y vertical
               if hcount >= hfp and hcount < h_sync_tamaño+hfp -1
               then
                    hdmi_hsync <= '1';
                else
                    hdmi_hsync <= '0';
                end if;
                if vcount >= vfp and vcount < v_sync_tamaño + vfp
- 1 then
                    hdmi_vsync <= '1';
                else
                    hdmi_vsync <= '0';
                end if;
-----------------------------------------------------------------
             end if;
         end if;
       hdmi_den <= hdmi_hact and hdmi_vact;
   end process;
```

```
-- Dibujo de patrones
    process(hcount,vcount, selec_patron)
       begin
-- Square pattern test
          if (selec_patron = '0') then
-- Blanco
if (((hcount >= 138 and hcount < 228) and (vcount >= 45 and vcount
< 105)) or
                ((hcount >= 768 and hcount < 857) and (vcount >=
465 and vcount < 524))) then
                hdmi_data <= "111111111111111111111111";
-- Amarillo
elsif (((hcount >= 138 and hcount < 228)and(vcount >= 105 and
vcount < 165)) or
                  ((hcount >= 228 and hcount < 318)and(vcount >=
45 and vcount < 105)) or
                  ((hcount >= 678 and hcount < 768)and(vcount >=
465 and vcount < 524)) or
                  ((hcount >= 768 and hcount < 857)and(vcount >=
405 and vcount < 465))) then
                hdmi_data <= "111111111111111100000000";
-- Cian
           elsif (((hcount >= 138 and hcount < 228)and(vcount >=
165 and vcount < 225)) or
                  ((hcount >= 228 and hcount < 318)and(vcount >=
105 and vcount < 165)) or
                  ((hcount >= 318 and hcount < 408)and(vcount >=
45 and vcount < 105)) or
                  ((hcount >= 588 and hcount < 678)and(vcount >=
465 and vcount < 524)) or
                  ((hcount >= 678 and hcount < 768)and(vcount >=
405 and vcount < 465)) or
                  ((hcount >= 768 and hcount < 857)and(vcount >=
345 and vcount < 405))) then
                hdmi_data <= "000000001111111111111111";
-- Verde
           elsif (((hcount >= 138 and hcount < 228)and(vcount >=
225 and vcount < 285)) or
                  ((hcount >= 228 and hcount < 318)and(vcount >=
165 and vcount < 225)) or
                  ((hcount >= 318 and hcount < 408)and(vcount >=
105 and vcount < 165)) or
                  ((hcount >= 408 and hcount < 498)and(vcount >=
45 and vcount < 105))or
                  ((hcount >= 498 and hcount < 588)and(vcount >=
465 and vcount < 524)) or
                  ((hcount >= 588 and hcount < 678)and(vcount >=
405 and vcount < 465)) or
                  ((hcount >= 678 and hcount < 768)and(vcount >=
345 and vcount < 405)) or
                  ((hcount >= 768 and hcount < 857)and(vcount >=
285 and vcount < 345))) then
                hdmi_data <= "000000001111111100000000";
-- Magenta
           elsif (((hcount >= 138 and hcount < 228)and(vcount >=
285 and vcount < 345)) or
                  ((hcount >= 228 and hcount < 318)and(vcount >=
225 and vcount < 285)) or
                  ((hcount >= 318 and hcount < 408)and(vcount >=
165 and vcount < 225)) or
```

```
((hcount >= 408 and hcount < 498)and((vcount >= 105 and vcount <
165)or
                  (vcount >= 465 and vcount < 524))) or
                  ((hcount >= 498 and hcount < 588)and((vcount >=
45 and vcount < 105)or
                  (vcount >= 405 and vcount < 465))) or
                  ((hcount >= 588 and hcount < 678)and(vcount >=
345 and vcount < 405))or
                  ((hcount >= 678 and hcount < 768)and(vcount >=
285 and vcount < 345)) or
                  ((hcount >= 768 and hcount < 857)and(vcount >=
225 and vcount < 285))) then
                hdmi_data <= "111111110000000011111111";
-- Rojo
            elsif (((hcount >= 138 and hcount < 228)and(vcount >=
345 and vcount < 405)) or
                  ((hcount >= 228 and hcount < 318)and(vcount >=
285 and vcount < 345)) or
                  ((hcount >= 318 and hcount < 408)and((vcount >=
225 and vcount < 285)or
                  (vcount >= 465 and vcount < 524))) or
                  ((hcount >= 408 and hcount < 498)and((vcount >=
165 and vcount < 225)or
                  (vcount >= 405 and vcount < 465))) or
                  ((hcount >= 498 and hcount < 588)and((vcount >=
105 and vcount < 165)or
                  (vcount >= 345 and vcount < 405))) or
                  ((hcount >= 588 and hcount < 678)and((vcount >=
45 and vcount < 105)or
                  (vcount >= 285 and vcount < 345))) or
                  ((hcount >= 678 and hcount < 768)and(vcount >=
225 and vcount < 285))or
                  ((hcount >= 768 and hcount < 857)and(vcount >=
165 and vcount < 225))) then
                hdmi_data <= "111111110000000000000000";
-- Azul
            elsif (((hcount >= 138 and hcount < 228)and(vcount >=
405 and vcount < 465)) or
                  ((hcount >= 228 and hcount < 318)and((vcount >=
345 and vcount < 405) or
                  (vcount >= 465 and vcount < 524))) or
                  ((hcount >= 318 and hcount < 408)and((vcount >=
285 and vcount < 345) or
                  (vcount >= 405 and vcount < 465))) or
                  ((hcount >= 408 and hcount < 498)and((vcount >=
225 and vcount < 285) or
                  (vcount >= 345 and vcount < 405))) or
                  ((hcount >= 498 and hcount < 588)and((vcount >=
165 and vcount < 225) or
                  (vcount >= 285 and vcount < 345))) or
                  ((hcount >= 588 and hcount < 678)and((vcount >=
105 and vcount < 165) or
                  (vcount >= 225 and vcount < 285))) or
                  ((hcount >= 678 and hcount < 768)and((vcount >=
45 and vcount < 105) or
```

```
(vcount >= 165 and vcount < 225))) or
                  ((hcount >= 768 and hcount < 857)and(vcount >=
105 and vcount < 165))) then
                hdmi_data <= "000000000000000011111111";
             else
-- Negro
                hdmi_data <= "000000000000000000000000";
            end if;
          else
-- Bar pattern test
-- Blanco
            if (hcount >= 138 and hcount < 228) then
                hdmi_data <= "111111111111111111111111";
-- Amarillo
            elsif (hcount >= 228 and hcount < 318) then
                hdmi_data <= "111111111111111100000000";
-- Cian
            elsif (hcount >= 318 and hcount < 408) then
                hdmi_data <= "000000001111111111111111";
-- Verde
            elsif (hcount >= 408 and hcount < 498) then
                hdmi_data <= "000000001111111100000000";
-- Magenta
            elsif (hcount >= 498 and hcount < 588) then
                hdmi_data <= "111111110000000011111111";
-- Rojo
            elsif (hcount >= 588 and hcount < 678) then
                hdmi_data <= "111111110000000000000000";
-- Azul
            elsif (hcount >= 678 and hcount < 768) then
                hdmi_data <= "000000000000000011111111";
-- Negro
            else
                hdmi_data <= "000000000000000000000000";
            end if;
           end if;
       end process;
-----------------------------------------------------------------
HDMI_CORE : entity work.CORE_HDMI
-- Llamada al CORE
      port map(
          pixel_clock => hdmi_pixel_clock,
          ddr_bit_clock => hdmi_ddr_clock,
          reset => reset,
          den => hdmi_den,
          hsync => hdmi_hsync,
          vsync => hdmi_vsync,
          pixel_data => hdmi_data,
          tmds_clk => hdmi_clk,
          tmds_d0 => hdmi_d0,
          tmds_d1 => hdmi_d1,
          tmds_d2 => hdmi_d2);
-----------------------------------------------------------------
end arq_TEST;
```

A.2. Código del *CORE*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

-- CORE para transmisión de video H.D.M.I.

entity CORE_HDMI is
  port(
    pixel_clock     : in std_logic;
-- Pixel clock
    ddr_bit_clock   : in std_logic;
-- DDR bit clock (pixel_clock*5)
    reset           : in std_logic;
-- Reset síncrono con pixel clock, activo nivel alto
    den             : in std_logic;
-- Display enable, activo nivel alto
    hsync           : in std_logic;
-- Sincronización horizontal de video
    vsync           : in std_logic;
-- Sincronización vertical de video
    pixel_data      : in std_logic_vector(23 downto 0);
-- Dato de entrada RGB de 24 bits
    tmds_clk        : out std_logic_vector(1 downto 0);
-- CLK diferencial de salida
    tmds_d0         : out std_logic_vector(1 downto 0);
-- Salidas diferenciales codificadas y serializadas
    tmds_d1         : out std_logic_vector(1 downto 0);
    tmds_d2         : out std_logic_vector(1 downto 0));
end CORE_HDMI;

architecture arq_CORE_HDMI of CORE_HDMI is
signal tmds_cod0: std_logic_vector(9 downto 0) :="0000000000";
signal tmds_cod1: std_logic_vector(9 downto 0) :="0000000000";
signal tmds_cod2: std_logic_vector(9 downto 0) :="0000000000";

begin
Codificador_R: entitywork.codificador_tmds
Codificación CANAL TMDS-2 RED
            port map(
                clk => pixel_clock,
                reset => reset,
                den => den,
                dato => pixel_data(23 downto 16),
                C0 => '0',
                C1 => '0',
                TMDS_Channel => 2,
                q_out => tmds_cod2);

        Serializador_R : entity work.serializador_tmds
-- Serialización CANAL TMDS-2
            port map(
                clk => pixel_clock,
                ddr_clk => ddr_bit_clock,
                reset => reset,
                dato => tmds_cod2,
                tmds_out => tmds_d2);
```

```
Codificador_G : entity work.codificador_tmds
-- Codificación CANAL TMDS-1 GREEN
            port map(
                clk => pixel_clock,
                reset => reset,
                den => den,
                dato => pixel_data(15 downto 8),
                C0 => '1',
                C1 => '0',
                TMDS_Channel => 1,
                q_out => tmds_cod1);

        Serializador_G : entity work.serializador_tmds
-- Serialización CANAL TMDS-1 GREEN
            port map(
                clk => pixel_clock,
                ddr_clk => ddr_bit_clock,
                reset => reset,
                dato => tmds_cod1,
                tmds_out => tmds_d1);
-------------------------------------------------------------------
        Codificador_B : entity work.codificador_tmds
-- Codificación CANAL TMDS-0 BLUE
            port map(
                clk => pixel_clock,
                reset => reset,
                den => den,
                dato => pixel_data(7 downto 0),
                C0 => hsync,
                C1 => vsync,
                TMDS_Channel => 0,
                q_out => tmds_cod0);

        Serializador_B : entity work.serializador_tmds
-- Serialización CANAL TMDS-0 BLUE
            port map(
                clk => pixel_clock,
                ddr_clk => ddr_bit_clock,
                reset => reset,
                dato => tmds_cod0,
                tmds_out => tmds_d0);
-------------------------------------------------------------------
        TMDS_CLK_generador : entity work.generador_clk_tmds
-- Generación de CANAL TMDS-CLK
            port map(
                clk => pixel_clock,
                tmds_clk => tmds_clk);
-------------------------------------------------------------------
end arq_CORE_HDMI;
```

A.3. Código del Módulo Codificador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

-- Se realiza la codificación según los periodos de Video Activo,
Control o Isla de Datos
-- Se obtiene un dato de salida de 10 bits

entity codificador_tmds is
  port(
    clk             : in  std_logic;
-- Pixel clock
    den             : in  std_logic;
-- Display enable, activo nivel alto
    reset           : in  std_logic;
-- Reset síncrono con pixel clock, activo nivel alto
    C0              : in  std_logic;
-- Señal de control C0
    C1              : in  std_logic;
-- Señal de control C1
    dato            : in  std_logic_vector(7 downto 0);
-- Dato de entrada
    TMDS_Channel    : in  natural range 0 to 2;
-- Canal en codificación
    q_out           : out std_logic_vector(9 downto 0));
-- Dato de salida codificado

end codificador_tmds;

architecture arq_codificador_tmds of codificador_tmds is
  signal q_m: std_logic_vector(8 downto 0) :="000000000";
-- Dato interno
  signal control: std_logic_vector(1 downto 0) :="00";
-- Concatenación C1 C0
  signal data_island: std_logic_vector(3 downto 0) := "0000";
-- Dato de entrada TERC4
  signal hcount,vcount: natural range 0 to 860 :=0;
-- Recorrido de píxeles de pantalla
  signal cnt: integer range -16 to 15 :=0;
-- Registro para seguimiento de la disparidad en el flujo de datos
begin
------------------------------------------------------------------
  process(clk)
-- Diseño en un sólo proceso
    variable N1_dato        : integer range 0 to 8 :=0;
-- Número de '1's en dato
    variable N1_q_m         : integer range 0 to 8 :=0;
-- Número de '1's en dato interno
    variable N0_q_m         : integer range 0 to 8 :=0;
-- Número de '0's en dato interno
```

```
  begin
    if(clk'event and clk = '1') then
-- Sincronismo y testeo reset
        if reset = '1' then
            q_m      <= "000000000";
            N1_dato  := 0;
            N1_q_m   := 0;
            N0_q_m   := 0;
            cnt      <= 0;
            hcount   <= 0;
            vcount   <= 0;
            q_out    <= "0000000000";
        else
-----------------------------------------------------------------
            if hcount = (858 - 1) then
-- Recorrido de los pixels de la imagen
                hcount <= 0;
                if vcount = (525 - 1) then
                    vcount <= 0;
                else
                vcount <= vcount + 1;
                end if;
            else
                hcount <= hcount + 1;
            end if;
-----------------------------------------------------------------
            if(den = '1') then
-- PERIODO DE VIDEO ACTIVO
-----------------------------------------------------------------
                N1_dato := 0;
-- Cálculo del número de '1's de dato
                for i in 0 to 7 loop
                    if (dato(i) = '1') then
                        N1_dato := N1_dato + 1;
                    end if;
                end loop;
-----------------------------------------------------------------
-- Asignación de valores a dato interno
if(N1_dato > 4 or (N1_dato = 4 and dato(0) = '0')) then
                    q_m(0) <= dato(0);
                    q_m(1) <= q_m(0) xnor dato(1);
                    q_m(2) <= q_m(1) xnor dato(2);
                    q_m(3) <= q_m(2) xnor dato(3);
                    q_m(4) <= q_m(3) xnor dato(4);
                    q_m(5) <= q_m(4) xnor dato(5);
                    q_m(6) <= q_m(5) xnor dato(6);
                    q_m(7) <= q_m(6) xnor dato(7);
                    q_m(8) <= '0';
                else
                    q_m(0) <= dato(0);
                    q_m(1) <= q_m(0) xor dato(1);
                    q_m(2) <= q_m(1) xor dato(2);
                    q_m(3) <= q_m(2) xor dato(3);
                    q_m(4) <= q_m(3) xor dato(4);
                    q_m(5) <= q_m(4) xor dato(5);
                    q_m(6) <= q_m(5) xor dato(6);
                    q_m(7) <= q_m(6) xor dato(7);
                    q_m(8) <= '1';
                end if;
```

```
-----------------------------------------------------------------
-- Cálculo del número de '1's y '0's de dato interno
                N1_q_m := 0;
                N0_q_m := 0;
                for i in 0 to 7 loop
                    if(q_m(i) = '1') then
                        N1_q_m := N1_q_m + 1;
                    end if;
                end loop;
                N0_q_m := 8 - N1_q_m;
-----------------------------------------------------------------
-- Balanceo DC selectivo, cálculo de nueva disparidad
               if(cnt = 0 or N1_q_m = N0_q_m) then
                    q_out(9) <= not q_m(8);
                    q_out(8) <= q_m(8);
                    if(q_m(8) = '0') then
                        q_out(7 downto 0) <= not q_m(7 downto 0);
                        cnt <= cnt + (N0_q_m - N1_q_m);
                    else
                        q_out(7 downto 0) <= q_m(7 downto 0);
                        cnt <= cnt + (N1_q_m - N0_q_m);
                    end if;
                else
                    if((cnt > 0 and N1_q_m > N0_q_m) or (cnt < 0
                     and N0_q_m > N1_q_m)) then
                        q_out(9) <= '1';
                        q_out(8) <= q_m(8);
                        q_out(7 downto 0) <= not q_m(7 downto 0);
                        if(q_m(8) = '0') then
                            cnt <= cnt + (N0_q_m - N1_q_m);
                        else
                            cnt <= cnt + 2 + (N0_q_m - N1_q_m);
                        end if;
                    else
                        q_out(9) <= '0';
                        q_out(8 downto 0) <= q_m(8 downto 0);
                        if(q_m(8) = '0') then
                            cnt <= cnt - 2 + (N1_q_m - N0_q_m);
                        else
                            cnt <= cnt + (N1_q_m - N0_q_m);
                        end if;
                    end if;
                end if;
-----------------------------------------------------------------
            else
-- PERIODO DE BLANKING
                cnt <= 0;
-----------------------------------------------------------------
                if (vcount < 45) then
-- BLANKING VERTICAL
```

```
-- PERIODO DE CONTROL
                  control <= C1 & C0;
                    case control is
                        when "00" => q_out <= "1101010100";
                        when "01" => q_out <= "0010101011";
                        when "10" => q_out <= "0101010100";
                        when "11" => q_out <= "1010101011";
                        when others => NULL;
                    end case;
-----------------------------------------------------------------
                 else
-- BLANKING HORIZONTAL
-----------------------------------------------------------------
-- PERIODO DE CONTROL Pre-Isla de Datos
                     if (hcount < 56) then
                        if TMDS_Channel = 2 then
                            control <= "01";
                        else
                            control <= C1 & C0;
                        end if;
                        case control is
                            when "00" => q_out <= "1101010100";
                            when "01" => q_out <= "0010101011";
                            when "10" => q_out <= "0101010100";
                            when "11" => q_out <= "1010101011";
                            when others => NULL;
                        end case;
-- BANDAS DE GUARDA - Isla de Datos
                    elsif ((hcount >= 56 and hcount < 58) or
                           (hcount >= 90 and hcount < 92)) then
                        if TMDS_Channel = 0 then
                            data_island <= "11" & C1 & C0;
-- CODIFICACIÓN TERC4
                            case (data_island) is
                           when "0000" => q_out <= "1010011100";
                           when "0001" => q_out <= "1001100011";
                           when "0010" => q_out <= "1011100100";
                           when "0011" => q_out <= "1011100010";
                           when "0100" => q_out <= "0101110001";
                           when "0101" => q_out <= "0100011110";
                           when "0110" => q_out <= "0110001110";
                           when "0111" => q_out <= "0100111100";
                           when "1000" => q_out <= "1011001100";
                           when "1001" => q_out <= "0100111001";
                           when "1010" => q_out <= "0110011100";
                           when "1011" => q_out <= "1011000110";
                           when "1100" => q_out <= "1010001110";
                           when "1101" => q_out <= "1001110001";
                           when "1110" => q_out <= "0101100011";
                           when "1111" => q_out <= "1011000011";
                                when others => NULL;
                            end case;
                        else
                            q_out <= "0100110011";
                        end if;
```

```
-- PERIODO DE ISLA DE DATOS

                    elsif (hcount >= 58 and hcount < 90) then
                     if TMDS_Channel = 0 then
                            if (hcount = 58) then
                                data_island <= "00" & C1 & C0;
                            else
                                data_island <= "10" & C1 & C0;
                            end if;
                        else
                            data_island <= "0000";
                        end if;
-- CODIFICACIÓN TERC4
                        case (data_island) is
                            when "0000" => q_out <= "1010011100";
                            when "0001" => q_out <= "1001100011";
                            when "0010" => q_out <= "1011100100";
                            when "0011" => q_out <= "1011100010";
                            when "0100" => q_out <= "0101110001";
                            when "0101" => q_out <= "0100011110";
                            when "0110" => q_out <= "0110001110";
                            when "0111" => q_out <= "0100111100";
                            when "1000" => q_out <= "1011001100";
                            when "1001" => q_out <= "0100111001";
                            when "1010" => q_out <= "0110011100";
                            when "1011" => q_out <= "1011000110";
                            when "1100" => q_out <= "1010001110";
                            when "1101" => q_out <= "1001110001";
                            when "1110" => q_out <= "0101100011";
                            when "1111" => q_out <= "1011000011";
                            when others => NULL;
                        end case;
-- PERIODO DE CONTROL Pre-Video Activo
                     elsif (hcount >= 92 and hcount < 136) then
                        control <= C1 & C0;
                        case control is
                            when "00" => q_out <= "1101010100";
                            when "01" => q_out <= "0010101011";
                            when "10" => q_out <= "0101010100";
                            when "11" => q_out <= "1010101011";
                            when others => NULL;
                        end case;
-- BANDA DE GUARDA Periodo de Video Activo

                    elsif (hcount >= 136 and hcount < 138) then
                        if (TMDS_Channel = 1) then
                            q_out <= "0100110011";
                        else
                            q_out <= "1011001100";
                        end if;
                    end if;
                end if;
            end if;
        end if;
    end if;
  end process;
-----------------------------------------------------------------
end arq_codificador_tmds;
```

A.4. Código del Módulo Serializador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
library UNISIM;
use UNISIM.VComponents.all;

-- Se realiza la serialización del dato codificado T.M.D.S. de 10
bits para transmisión diferencial.

entity serializador_tmds is
  port(
    clk             : in std_logic;
-- Pixel_clock
    ddr_clk         : in std_logic;
-- DDR bit clock (pixel_clock*5)
    reset           : in std_logic;
-- Reset síncrono con pixel clock, activo nivel alto
    dato            : in std_logic_vector(9 downto 0);
-- Dato de salida del codificador T.M.D.S.
    tmds_out        : out std_logic_vector(1 downto 0));
-- Salida diferencial del serializador
end serializador_tmds;

architecture arq_serializador_tmds of serializador_tmds is
  signal shiftout1, shiftout2   : std_logic;
-- Expansión maestro-esclavo OSERDESE2
  signal dato_se                : std_logic;
-- Dato serializado
  signal reset_sincrono         : std_logic;

begin
-----------------------------------------------------------------
-- Sincronización y registro de reset
    process(clk)
        begin
        if (clk'event and clk = '1') then
            reset_sincrono <= reset;
        end if;
    end process;
-----------------------------------------------------------------
  Maestro : OSERDESE2
    generic map(
      DATA_RATE_OQ => "DDR",
      DATA_RATE_TQ => "SDR",
      DATA_WIDTH => 10,
      SERDES_MODE => "MASTER",
      TRISTATE_WIDTH => 1,
      TBYTE_CTL => "FALSE",
      TBYTE_SRC => "FALSE")
```

```
port map(
      CLK => ddr_clk,
      CLKDIV => clk,
      D1 => dato(0),
      D2 => dato(1),
      D3 => dato(2),
      D4 => dato(3),
      D5 => dato(4),
      D6 => dato(5),
      D7 => dato(6),
      D8 => dato(7),
      T1 => '0',T2 => '0',T3 => '0',T4 => '0',
      TCE => '1',
      OCE => '1',
      TBYTEIN => '0',
      RST => reset_sincrono,
      SHIFTIN1 => shiftout1,
      SHIFTIN2 => shiftout2,
      OQ => dato_se,
      OFB => open,
      TQ => open,
      TFB => open,
      TBYTEOUT => open,
      SHIFTOUT1 => open,
      SHIFTOUT2 => open);
  Esclavo : OSERDESE2
    generic map(
      DATA_RATE_OQ => "DDR",
      DATA_RATE_TQ => "SDR",
      DATA_WIDTH => 10,
      SERDES_MODE => "SLAVE",
      TRISTATE_WIDTH => 1,
      TBYTE_CTL => "FALSE",
      TBYTE_SRC => "FALSE")
    port map(
      CLK => ddr_clk,
      CLKDIV => clk,
      D1 => '0',
      D2 => '0',
      D3 => dato(8),
      D4 => dato(9),
      D5 => '0',
      D6 => '0',
      D7 => '0',
      D8 => '0',
      T1 => '0',T2 => '0',T3 => '0',T4 => '0',
      TCE => '1',
      OCE => '1',
      TBYTEIN => '0',
      RST => reset_sincrono,
      SHIFTIN1 => '0',
      SHIFTIN2 => '0',
      OQ => open,
      OFB => open,
      TQ => open,
```

```
    TFB => open,
      TBYTEOUT => open,
      SHIFTOUT1 => shiftout1,
      SHIFTOUT2 => shiftout2);
-------------------------------------------------------------------
-- Buffer de salida diferencial
   Salida_diferencial : OBUFDS
    generic map (
      IOSTANDARD => "DEFAULT",
          SLEW => "FAST")
    port map (
      I => dato_se,
      O => tmds_out(1),
      OB => tmds_out(0));
end arq_serializador_tmds;
```

A.5. Código del Módulo Generador TMDS-CLK

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
library UNISIM;
use UNISIM.VComponents.all;

-- Se realiza la generación del CANAL TMDS-CLK
entity generador_clk_tmds is
  port(
    clk   : in std_logic; -- Pixel clock
    tmds_clk :out std_logic_vector(1 downto 0));-- TMDS-CLK
end generador_clk_tmds;
architecture arq_generador_clk_tmds of generador_clk_tmds is
    signal clk_interno  :   std_logic;
begin
  Generador_clk_interno    : ODDR
    generic map (
      DDR_CLK_EDGE => "OPPOSITE_EDGE",
      INIT => '0',
      SRTYPE => "SYNC")
    port map (
      Q => clk_interno,
      C => clk,
      CE => '1',
      D1 => '1',
      D2 => '0',
      R => '0',
      S => '0');
-------------------------------------------------------------------
-- Buffer de salida diferencial
   Salida_diferencial : OBUFDS
    generic map (
      IOSTANDARD => "DEFAULT",
      SLEW => "FAST")
    port map (
      I => clk_interno,
      O => tmds_clk(1),
      OB => tmds_clk(0));
-------------------------------------------------------------------
end arq_generador_clk_tmds;
```

A.6. Código de asignación de pines E/S

```
#CLK 125Mhz
set_property PACKAGE_PIN L16 [get_ports CLOCK_125]
set_property IOSTANDARD LVCMOS33 [get_ports CLOCK_125]

#Configuración de usuario
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN R18 [get_ports reset]
set_property PACKAGE_PIN G15 [get_ports selec_patron]
set_property IOSTANDARD LVCMOS33 [get_ports selec_patron]

#HDMI PINOUT
set_property PACKAGE_PIN H16 [get_ports {hdmi_clk[1]}]
set_property IOSTANDARD TMDS_33 [get_ports {hdmi_clk[1]}]
set_property PACKAGE_PIN D19 [get_ports {hdmi_d0[1]}]
set_property IOSTANDARD TMDS_33 [get_ports {hdmi_d0[1]}]
set_property PACKAGE_PIN C20 [get_ports {hdmi_d1[1]}]
set_property IOSTANDARD TMDS_33 [get_ports {hdmi_d1[1]}]
set_property PACKAGE_PIN B19 [get_ports {hdmi_d2[1]}]
set_property IOSTANDARD TMDS_33 [get_ports {hdmi_d2[1]}]
set_property PACKAGE_PIN F17 [get_ports hdmi_out_en]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_out_en]
set_property PACKAGE_PIN E18 [get_ports hdmi_hpd]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_hpd]
```

Referencias

[1] Página web *https://www.hdmi.org/index.aspx*

[2] Jeffrey A. Boccaccio, Derek R. Flickinger; "*HDMI® Uncensored – Inside HDMI*", 2012.

[3] High-Definition Multimedia Interface Specification Version 1.3a, Hitachi, Ltd., Matsushita Electric Industrial Co., Ltd., Philips Consumer Electronics International B.V., Silicon Image, Inc., Sony Corporation, Thomson Inc., Toshiba Corporation y HDMI Licensing. Disponible en: *http://www.fpga4fun.com/files/HDMISpecification13a.pdf*

[4] Xilinx. ZYBO Reference Manual. Disponible en: *https://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPZYBO/documentation/ZYBO_RM_B_V6.pdf*

[5] Página web oficial de Xilinx. *https://www.xilinx.com/*

[6] Xilinx. 7 Series FPGAs SelectIO Resources User Guide. Disponible en: *https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf*

[7] Página web *http://www.fpga4fun.com*

[8] Xilinx. Zynq-7000 All Programmable SoC: Embedded Design Tutorial. Disponible en: *https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug1165-zynq-embedded-design-tutorial.pdf*

[9] Xilinx. Vivado Design Suite Tutorial: Embedded Processor Hardware Design. Disponible en:

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug940-vivado-tutorial-embedded-design.pdf

[10] Xilinx. Vivado Design Suite. Using Constraints. Disponible en: *https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug903-vivado-using-constraints.pdf*

[11] Xilinx. Clocking Wizard v5.3. LogiCORE IP Product Guide.Disponible en: *https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v5_3/pg065-clk-wiz.pdf*

[12] Xilinx. Implementing a TMDS Video Interface in the Spartan-6 FPGA. Disponible en: *https://www.xilinx.com/support/documentation/application_notes/xapp495_S6TMDS_Video_Interface.pdf*

[13] Página web *http://hamsterworks.co.nz/mediawiki/index.php/Main_Page*