Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO EN FPGA DE UN ENLACE DE COMUNICACIONES SERIE PARA UN CONVERTIDOR DE POTENCIA DISTRIBUIDO

Autor:

MARTIN CABALLERO, PABLO

Director:

SANTIAGO DE PABLO GÓMEZ
Dpto. Tecnología Electrónica

Valladolid, Julio 2019

RESUMEN

El presente proyecto se centra en el desarrollo de un enlace de comunicaciones para un convertidor de potencia distribuido, siendo este enlace un elemento de vital importancia en dicho tipo de convertidor, puesto que se necesita una comunicación constante entre todos los módulos de potencia para realizar las correspondientes conmutaciones en el momento preciso. El desarrollo del enlace se ha realizado con un conjunto de circuitos digitales, programados en lenguaje de máquinas de estados (ASM++), simulados con una serie programas antes de su implementación física, y finalmente se ha probado su comportamiento en un dispositivo reconfigurable tipo FPGA con el que se ha podido verificar el correcto funcionamiento de todos los módulos del enlace de comunicaciones transmitiendo datos a 50 Mbps.

PALABRAS CLAVE

- Transmisor
- Receptor
- Convertidor distribuido
- ASM++
- Sincronización

ABSTRACT

The present project focuses on the development of a communication link for a distributed power converter, this link being an element of vital importance in this type of converter, since a constant communication between all the power modules is needed to realize the corresponding commutations at the precise moment. The development of the link has been made with a set of digital circuits, programmed in the language of state machines (ASM ++), simulated with a series of programs before their physical implementation, and finally their behavior has been tested in a reconfigurable device type FPGA with which has been able to verify the correct operation of all the modules of the communication link transmitting data at 50 Mbps.

KEYWORDS

- Transmitter
- Receiver
- Ristributed converter
- ASM++
- Synchronization

Índice de contenidos

Índice de Figuras

1 INTRODUCCIÓN Y OBJETIVOS

Este proyecto de fin de grado ha sido realizado gracias a la colaboración del Departamento de Tecnología Electrónica de la Universidad de Valladolid.

1.1. INTRODUCCIÓN

El presente proyecto, titulado **"DESARROLLO EN FPGA DE UN ENLACE DE COMUNICACIONES SERIE PARA UN CONVERTIDOR DE POTENCIA DISTRIBUIDO"**, pretende realizar un sistema de comunicaciones que permita enviar de forma segura paquetes de bits para el control de las conmutaciones de los distintos módulos de un convertidor distribuido.

Se entiende por comunicación toda transferencia de información con sentido desde un lugar (remitente, origen, fuente, transmisor) a otro lugar (destino, receptor) utilizando un patrón físico para codificar la información que deberá ser único, capaz de ser enviado por el transmisor y detectado y entendido por el receptor.

Para el procesamiento de la información se diseñará un sistema electrónico, es decir un conjunto de circuitos digitales que interactuarán entre sí para dar lugar a dicha comunicación. La realización de los diseños se llevará a cabo a través de una serie de lenguajes de descripción de hardware (HDL) empleados en el modelado de sistemas electrónicos, los más comunes utilizados internacionalmente son VHDL y Verilog.

Estos lenguajes se encargan de definir la estructura, diseño y operación de los circuitos digitales haciendo posible obtener una descripción formal de los mismos, así como un análisis automático y una simulación previa de los circuitos sin necesidad de esperar a su implementación física. El aprendizaje de estos no es excesivamente difícil, pero diseñar circuitos digitales complejos escritos directamente en lenguaje VHDL o Verilog puede volverse en ocasiones un trabajo tedioso y visualmente poco descriptivo. Para simplificar el diseño de estos circuitos, a largo de este proyecto se va a utilizar una potente herramienta grafica de diseño denominada diagramas ASM ++, basada en las máquinas de estado algorítmicas tradicionales que permite realizar los diseños de forma más intuitiva y visual.

Una vez se tengan definidos los diagramas ASM++, mediante un compilador de diagramas ASM++ (desarrollado por el Departamento de Tecnología Electrónica de la Universidad de Valladolid) se pueden obtener las descripciones de los circuitos en los diferentes lenguajes HDL y a partir de estos realizar las verificaciones y simulaciones necesarias para comprobar el correcto funcionamiento de lo que sería el circuito real.

1.2. OBJETIVOS DEL PROYECTO

La implementación de un convertidor de potencia distribuido en una planta solar fotovoltaica para la producción de energía eléctrica necesita imprescindiblemente de un enlace de comunicaciones que permita transmitir información de forma fiable y a la suficiente velocidad para que se puedan tomar a tiempo las decisiones correspondientes en los *switches* de conmutación en función de los requerimientos del convertidor. Con dicho convertidor se estima obtener un rendimiento de transformación mayor que con otros dispositivos utilizados hasta el momento.

Dada la finalidad del enlace de comunicaciones los objetivos de este proyecto han sido:

- El diseño de un circuito transmisor que envié datos codificados de forma secuencial para su posterior recuperación de forma sincronizada.
- El diseño de un circuito resicronizador que se encargue se sincronizar la señal transmitida por el transmisor con el receptor debido a la naturaleza plesiocrona del sistema.
- El diseño de un circuito receptor que se encargue de decodificar y testear que los paquetes de datos recibidos son correctos.
- La implementación de todos circuitos en un dispositivo reprogramable tipo FPGA para la realización de pruebas físicas y verificar su funcionamiento.

2. ESTADO DEL ARTE

2.1 MARCO ACTUAL

La deficiente previsión de las fuentes de energía basadas en los combustibles fósiles y el inminente aumento de consumo de energía que requiere la sociedad moderna y la industrialización está impulsando buscar nuevas alternativas limpias para la generación de energía. Además, vivimos unos tiempos en los que la contaminación es uno de los factores que hay que tener muy presente en cualquier tipo acción que se realice, partiendo desde el nivel más bajo como puede ser dejar el grifo dado mientras te lavas los dientes hasta los niveles más altos como pueden ser cualquier tipo de proceso destinado a la fabricación en masa de productos o la generación de energía a escala global. La contaminación está causando daños muy severos en nuestro planeta que afectan directamente a todos los seres vivos que habitamos en él, como se puede observar cada día se multiplican las noticias relacionadas con los efectos de la contaminación, como son la desertificación, escasez de agua dulce, cambios climáticos (calentamiento global), contaminación de los océanos, lluvia acida entre otras.

La generación de energía eléctrica ocupa una de las posiciones de mayor peso de cara a la contaminación, es por ello por lo que cada día se apuesta más por métodos de generación renovables como son las centrales hidráulicas, eólicas y fotovoltaicas.

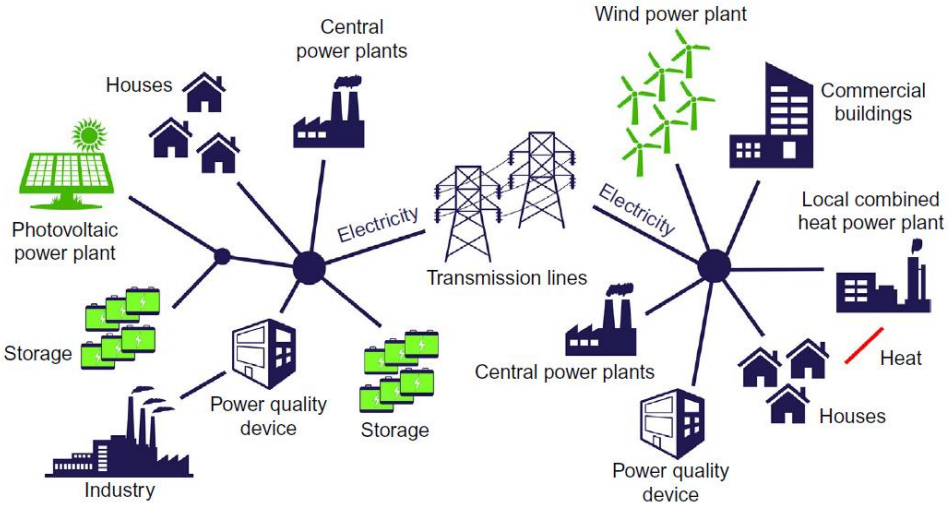


Figura 1: Generación y usos de la energía eléctrica.

Aquí es donde se enmarca este proyecto, en la generación de energía eléctrica mediante paneles fotovoltaicos mediante métodos modernos que requieren una comunicación constante entre los diferentes dispositivos para ser viable.

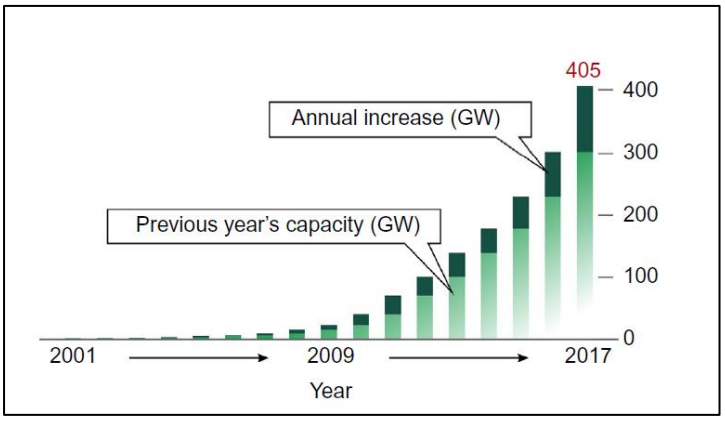


Figura 2: Incremento global de generación de energía eléctrica mediante paneles fotovoltaicos.

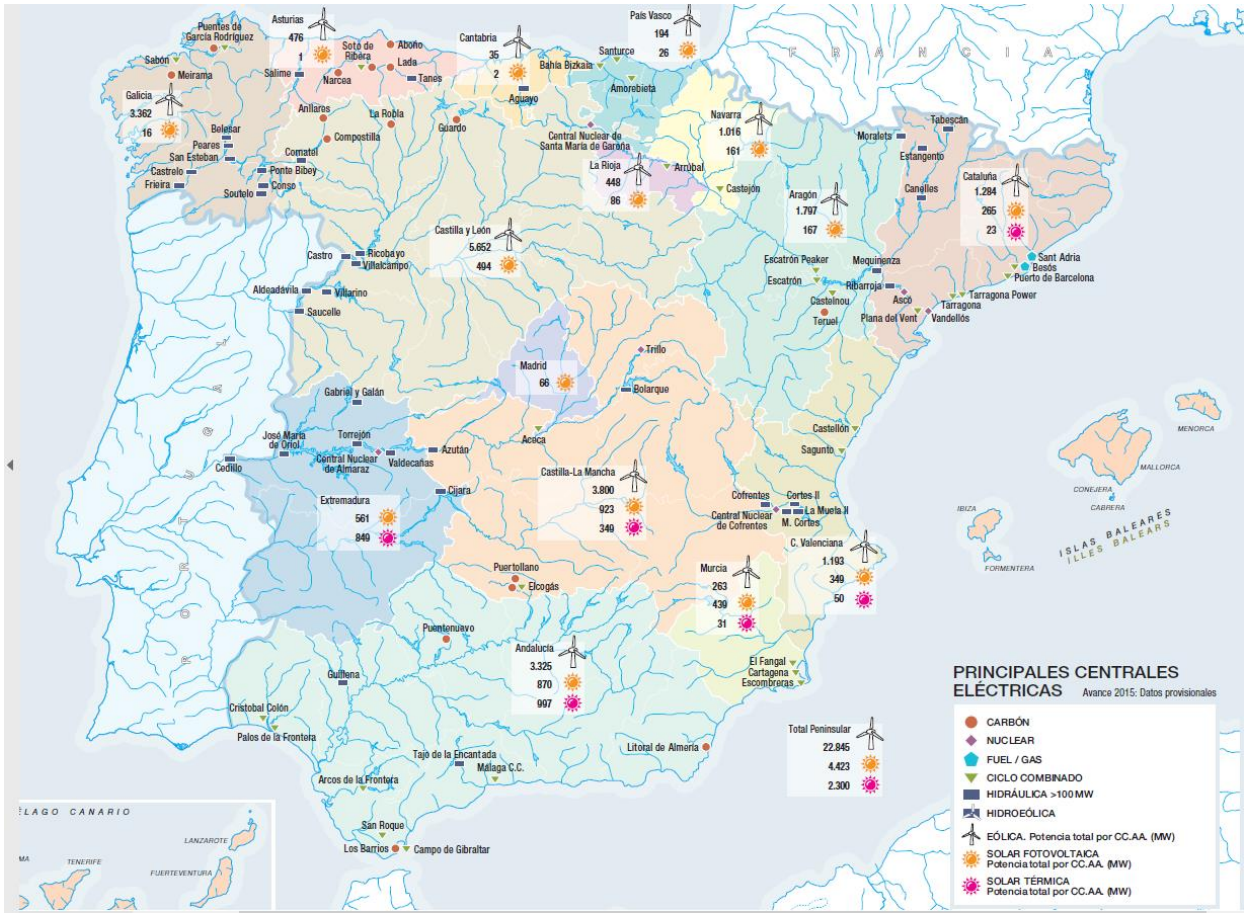


Figura 3: Principales centrales eléctricas año 2015.

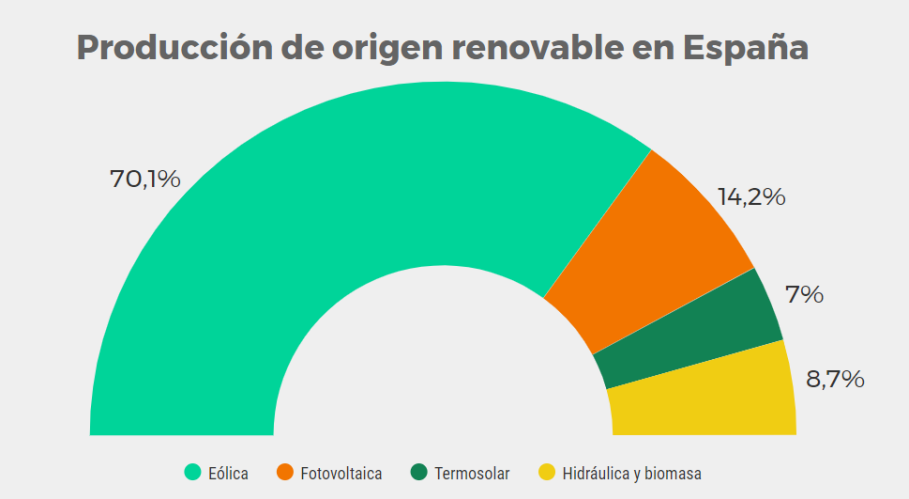


Figura 4: Producción energía de origen renovable en España año 2017.

2.2 GENERACIÓN ENERGÍA ELÉCTRICA MEDIANTE PANELES FOTOVOLTAICOS

Dado que el coste de obtener energía con paneles fotovoltaicos es relativamente bajo en comparación con el precio de la energía eléctrica, cada día se está investigando más como mejorar la eficiencia de transformación. En la actualidad existen tres tipos de instalaciones muy diferenciados en función de la potencia que producen y el tipo de convertidores que usan:

- ***Central inverter:*** se usa para instalaciones de alto voltaje (más de 1500V en continua) que producen más de 100 kWp y se consiguen rendimientos de transformación del orden del 98%. A continuación, se muestra una imagen del esquema una instalación con convertidor central.
- ***String inverter:*** se usa en instalaciones de menor voltaje que una estructura *"central inverter"*, normalmente se trabaja con tensiones del orden de 400V pero la transformación de la energía está repartida por agrupaciones de placas fotovoltaicas, es decir se tienen distintos hilos de trabajo, que terminan produciendo una potencia global elevada, de orden de 100kWp. Utilizando este método de transformación se consiguen rendimientos del orden del 98,5%. A continuación, se muestra una imagen de como seria la estructura de un *"string inverter"*. Dentro de las estructuras *"string inverter"* se pueden realizar conexiones de varias ramas de trabajo en serie o paralelo según las especificaciones de potencia requeridas. Estas estructuras se denominan *"multistring"* y se usan para alimentar a las tres fases de la red. A continuación, se muestra dos imágenes de como seria las conexiones en serie y en paralelo.

- ***Module converter:*** se usan en instalaciones pequeñas de bajo voltaje que producen hasta un 1kWp. Dentro de este grupo podemos encontrar 2 tipos:
  - ***Power optimizer:*** maximizando y optimizando cada panel fotovoltaico de forma individual consiguiéndose rendimientos del orden del 98,8%. A continuación, se muestra la estructura general de un *"power optimizer"*.
  - ***Micro inverter:*** la forma de trabajo es la misma que en la estructura *"power optimizer"* pero los convertidos son muchos más compactos y se disminuye el rendimiento de transformación que oscila entre el 90% y el-95%. A continuación, se adjunta una imagen del montaje general de un *"micro inverter"*.

En la actualidad la mayoría de los campos fotovoltaicos de alta producción de energía adoptan un esquema de producción *"central inverter"* consiguiéndose altos rendimientos de transformación. Sin embargo, para instalaciones de muy alta potencia, del orden de MW (megavatios) se están comenzando a usar convertidores modulares multinivel, básicamente se divide la instalación en submódulos (SMs) que se van conectando en serie hasta conseguir el nivel de tensión que se requiera. A continuación, se muestra una imagen la estructura de un convertidor MMC.

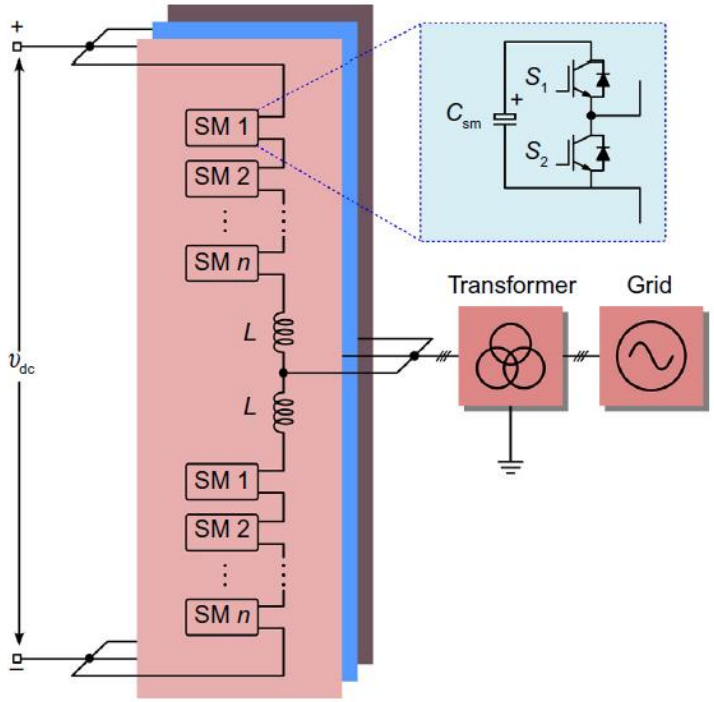


Figura 5: Estructura de un convertidor MMC.

2.3 SISTEMAS DE TELECOMUNICACIONES ACTUALES

Vivimos en una sociedad en la que el uso de los sistemas de telecomunicaciones es cada día más imprescindible, como por ejemplo internet, la telefonía móvil, la televisión, el GPS entre muchas otras aplicaciones. Cuando hablamos de telecomunicaciones nos estamos refiriendo a toda transmisión de señales, ya sean de carácter electromagnético u óptico. Para que dicha transmisión sea posible se necesita de un medio físico en el que se propaguen las señales.

2.3.1 COMUNICACIÓN SERIE Y PARALELO

A la hora de transmitir datos hay que tener presente que volumen se necesita transmitir para la aplicación que se esté desarrollando. Dependiendo de esto se pueden diferenciar dos formas generales de comunicación:

- **Comunicación serie o secuencial:** consiste en él envió de datos bit a bit de forma secuencial sobre un canal de comunicación o un bus de datos. Es ampliamente utilizada en telecomunicaciones e informática ya se necesita un número más reducido de líneas de transmisión.
- **Comunicación en paralelo:** se utilizan varias líneas de transmisión y por cada una de ellas se envían bits de forma independiente y todos al mismo tiempo. Presenta el inconveniente de reordenar los bits que conforman los mensajes en el receptor. En comparación con la comunicación serie, a la misma frecuencia de trabajo presenta mayor rendimiento de transmisión.

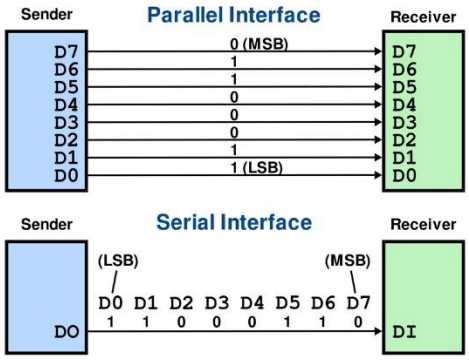


Figura 6: Esquema comunicación paralelo y comunicación serie.

2.3.2 MEDIOS FÍSICOS DE COMUNICACIÓN

Dependiendo del carácter de las señales que se quieren transmitir, los medios físicos de telecomunicaciones se pueden dividir en dos grandes grupos:

- **Medios de transmisión guiados:** Aquellos que precisan de un canal sólido, como pueden ser cables para transmitir señales eléctricas o mediante fibra óptica para transmitir señales de luz.
- **Medios de transmisión no guiados:** por ejemplo, las señales que se envían a través del aire mediante ondas electromagnéticas.

Dentro de los medios de transmisión guiados nos encontramos con los siguientes tipos:

- **Cable de pares:** formado por grupo de dos hilos (denominados pares), aislados entre sí y recubiertos de un material plástico. Se usan para transmitir información en distancias cortas ya que en distancias largas se pierde información. Se ve afectado posibles interferencias externas como pueden ser las interferencias electromagnéticas que podrían alterar la información transmitida.
- **Cable coaxia**l: Conductor central rodeado por una capa conductora cilíndrica. Se emplea en sistemas troncales o de largo alcance que portan señales múltiples con gran número de canales.
- **Fibra Óptica:** Este medio de comunicación se basa en el transporte de señales mediante impulsos de luz generados por emisores de luz Leds. El cable consta de uno o más filamentos el centro de cada filamento se denomina núcleo, que es por donde se propaga la luz. EL núcleo esta recubierto de una capa de vidrio que refleja la luz hacia el interior consiguiéndose largas distancias de propagación. Ofrece mayor fiabilidad que los sistemas de comunicación inalámbricos, se consiguen altas velocidades de transmisión y además la propagación de la información por la fibra no se ve afectada por las interferencias electromagnéticas.

Dentro de los medios de transmisión no guiados los más principales son los siguientes:

- **Radiofrecuencia u ondas de radio:** empleadas en la transmisión de televisión, radio, radar, telefonía móvil entre otras.
- **Red por infrarrojos:** Las redes por infrarrojos permiten la comunicación entre dos nodos utilizando un par de leds que emiten destellos de luz en el espectro infrarrojo.
- **Microondas:** las microondas permiten transmisiones tanto con antenas terrestres como con satélites. Dada sus frecuencias, del orden de 1 a 10 Ghz, las microondas son muy direccionales y solo se pueden emplear en situaciones en que existe una línea visual entre emisor y receptor.

2.3.3 CODIFICACIONES PARA LA TRANSMISIÓN DE DATOS

Cuando se quieren transmitir datos de forma sincronizada y poder identificarlos fácilmente siguiendo patrones, es necesario usar codificaciones que permitan enviar mensajes y tramas de sincronización y control por una misma línea de transmisión. Se podría no hacer uso de estas, pero se tendrían que enviar los mensajes de sincronización y control por una línea y los mensajes por otra, lo que conllevaría sincronizar dos líneas de transmisión físicamente separas y resolver otros problemas adicionales, es por ello que usar codificaciones resulta más eficiente.

Se entiende por codificación dentro del entorno de las comunicaciones, al proceso de conversión de un paquete de datos en otro paquete de datos de mayor o menor tamaño cuya información sea equivalente al paquete original. Existen multitud de codificaciones en función del tipo de señal (óptica, sonora, electromagnética) con la que se esté trabajando, a continuación, se explican algunos tipos que podrían ser de interés para este proyecto.

- **Codificación 4b5b:** se trata de una codificación altamente utilizada en las comunicaciones de datos, se basa en el mapeo de grupos de 4 bits en grupos de 5 bits para su transmisión. Las palabras de 5 bits están predeterminadas en un diccionario, por lo que podrán ser interpretadas por los circuitos receptores. Este tipo permite una codificación unipolar.

Data		4B5B code	Data		4B5B code
(Hex)	(Binary)		(Hex)	(Binary)	
0	0000	11110	8	1000	10010
1	0001	01001	9	1001	10011
2	0010	10100	A	1010	10110
3	0011	10101	B	1011	10111
4	0100	01010	C	1100	11010
5	0101	01011	D	1101	11011
6	0110	01110	E	1110	11100
7	0111	01111	F	1111	11101

Figura 7: Codificación 4b5b normalizada.

- **Codificación 8b10b:** se basa en el mapeo de grupos de 8 bits en grupos de 10 bits para su transmisión. Esta codificación necesita cumplir un equilibrio entre cero y unos, pero permite un mayor rango de transmisión, ya que se dispondrán 512 combinaciones para mensajes y 512 combinaciones para palabras de control y palabras de equilibrado de la línea. Este tipo de codificación presenta una serie de dificultades en comparación con la 4b5b ya que las combinaciones son mucho más numerosas y tener control sobre ellas exige una mayor complejidad a la hora de programar y hay que tener presente el equilibrado en continua (*DC Balance*) de ceros y unos. Además, para este tipo de codificación se aconseja que sea bipolar.
- **Codificación 64b66b:** se basa en el mapeo de grupos de 64 bits en grupos de 66 bits, lo cual proporciona suficientes cambios de estado para recuperar la señal de reloj y se consigue una mejor alineación de las secuencias de datos en el receptor. Permite velocidades de transmisión de datos muy elevadas y se consigue una mejora de la eficiencia en comparación con la codificación 8b10b descrita anteriormente.
- **Codificación Manchester (1b2b):** esta codificación se basa en transformar cada bit en una transición ascendente o descendente

dependiendo de si se tiene un 1 o 0, por lo que siempre vamos a tener una transición en el medio de cada bit transmitido. La principal desventaja de esta codificación frente a las otras descritas es que se reduce la velocidad de transmisión en un 50% por lo que se tiene que trabajar a frecuencias más altas.

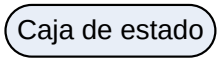
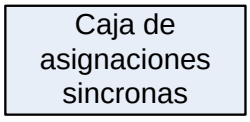
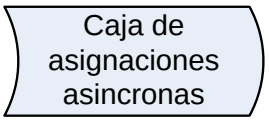
2.3.4 COMUNICACIÓN ENTRE DOS SISTEMAS

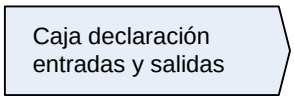
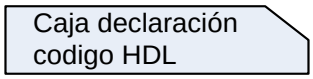
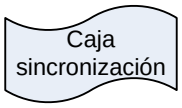
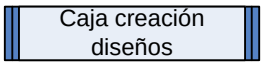
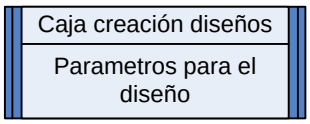
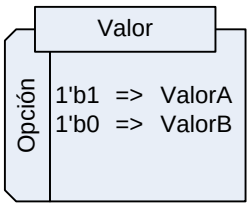
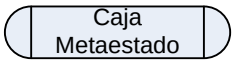
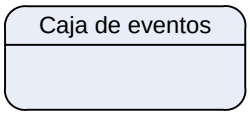
Cuando se quiere comunicar dos sistemas que se encuentran físicamente separados nos vamos a encontrar con algunas limitaciones temporales debido a lo que tardan las señales en enviarse por los medios físicos de transmisión según la separación de los sistemas. Es por ello por lo que existen distintos tipos sistemas en función de la transmisión y recepción de datos:

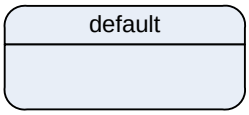
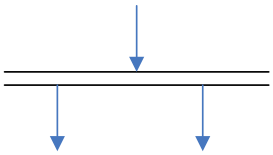
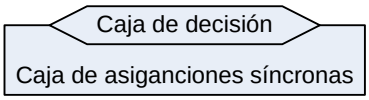
- **Sistema síncrono:** un sistema síncrono es aquel en el que los relojes se controlan para funcionar, idealmente, a velocidades idénticas, o a la misma tasa media con un desplazamiento de fase relativo fijo, dentro de un rango limitado.
- **Sistemas asíncronos:** son aquellos que no tienen señales de reloj centralizadas, por lo que no dependen de tiempos estrictos de llegada de señales para realizar una operación determinada. La coordinación se logra a través de eventos como la llegada de paquetes, transiciones en las señales, protocolos de intercambio y otros métodos.
- **Sistemas plesiócronos:** En el entorno de las telecomunicaciones, se define un sistema plesiócrono cuando diferentes partes del sistema están casi, pero no del todo, perfectamente sincronizadas. Un emisor y un receptor operan de forma plesiosincrónica si operan a la misma frecuencia nominal de reloj, pero pueden tener una ligera falta de coincidencia en la frecuencia de reloj, lo que conduce a un pequeño desfase. El desajuste entre los relojes de los dos sistemas se conoce como la diferencia plesiócrona.
  En general, los sistemas plesiócronos se comportan de manera similar a los sistemas síncronos, excepto que se debe emplear algún medio para hacer frente a los "desfases de sincronización", que irán cambiando a lo largo del tiempo debido a la naturaleza plesiócrona del sistema.
- **Sistema Mesócrono:** es aquel en el que los relojes de los sistemas funcionan a la misma frecuencia, pero con fases desconocidas.

2.4 ENTORNO DE PROGRAMACIÓN ASM++

La Universidad de Valladolid en colaboración con el Departamento de Tecnología Electrónica y teniendo como referente en la materia al director de este proyecto, han desarrollado un lenguaje grafico denominado ASM++ que junto con la herramienta *MSVisio* permite generar circuitos digitales de una forma más sencilla. Una vez se tiene el código en el entorno gráfico, otra herramienta llamada *AsmCompiler* se encarga reconocer los elementos gráficos que contenga el archivo de *MsVisio* y transfórmalos en diferentes lenguajes de programación de máquinas de estados, entre los que se encuentran VHDL y Verilog. A continuación, se muestran los elementos gráficos del leguaje ASM++ más utilizados.

CAJA	DESCRIPCIÓN
	Las cajas de estado identifican el comienzo y el final de cada ciclo de reloj. Se pueden rellenar con un nombre significativo según las acciones que se realicen en dicho estado o se pueden dejar en blanco y el compilador automáticamente genera un nombre aleatorio. También se pueden utilizar como enlace entre dos partes de un diagrama. En todo grafico de ASM++ debe existir al menos una de estas cajas.
	En estas cajas se pueden escribir múltiples operaciones, pero se ejecutarán simultáneamente al finalizar el ciclo de reloj en el que se describen y serán visibles para el resto del circuito un ciclo de reloj más tarde del que se han descrito. Cualquier señal definida en una caja de este tipo tendrá un comportamiento síncrono.
	En estas cajas se pueden escribir múltiples operaciones que se ejecutaran en paralelo durante el ciclo de reloj en el que se han descrito. Cualquier señal definida en una caja de este tipo tendrá un comportamiento asíncrono. Al usar este tipo de caja hay que tener presente que los resultados de las operaciones son visibles únicamente en ciclo de reloj en el que se están ejecutando, un ciclo después la información se pierde.

	Las cajas de decisión permiten elegir entre dos caminos alternativos basándose en la condición descrita en su interior,
	Estas cajas se utilizan para definir las entradas y salidas de un diseño.
	Estas "cajas de códigos" se utilizan para describir directivas de compilación (precedidas por un símbolo #) y código fuente directo (VHDL oVerilog), por ejemplo, para declarar señales internas se utilizan las palabras 'signal','wire' y 'reg'.
	Esta caja establece el nombre y la polaridad de la señal de sincronización utilizada por un circuito o por un hilo. Todas las cajas situadas después de esta están sincronizadas con la señal de reloj descrita.
 	Esta caja especifica el nombre de cada módulo/entidad en el que se está trabajando y opcionalmente permite definir parámetros genéricos para todo el circuito (segunda caja).
Sync Table Box 	Esta caja funciona como un switch. En la imagen se muestra un pequeño ejemplo: dependiendo del valor la señal "opción" en la señal "Valor" se almacenará síncronamente el ValorA (si "opción == 1") o el ValorB (si "opción==0").
	Esta caja se utiliza para indicar el final de un diagrama AMS++.
 	Esta caja se utiliza para describir el comportamiento del circuito cuando ocurre el evento definido en la parte superior de esta (las imágenes pequeñas son un ejemplo) Su principal aplicación es describir las secuencias de inicialización (reinicio) y el primer estado de

	las máquinas de estado. Estas cajas deben estar ubicadas antes de cualquier caja de estado.
	Esta caja se utiliza para especificar valores por defecto tanto de señales síncronas como de señales asíncronas. Debe ser utilizada antes de cualquier caja de estado.
	Una "línea doble" establece subprocesos paralelos que se ejecutaran de manera concurrente. Esta forma del lenguaje ASM++ permite la descripción de múltiples máquinas de estado, con la misma sincronización o distinta. Una vez que un hilo se bifurca en otros, no puede volver a unirse.
Referencia en la misma página 	Esta caja es un "conector" que enlaza sin efectos adicionales dos partes de un diagrama. Esta caja de forma circular solo se puede usar para conectar dos puntos de un circuito que se encuentren en la misma hoja.
	Esta caja es una forma compacta de programar que si se cumple la condición realiza la operación síncrona que encuentra debajo, si no se cumple la condición pasa al siguiente estado sin realizar la operación. También se puede encontrar este tipo de conjunto con una caja de operaciones asíncronas.

Existe la siguiente página web: **www.deeper.uva.es** donde se puede consultar más documentación acerca de los diagramas ASM++.

2.5 DISPOSITIVOS RECONFIGURABLES TIPO FPGA

2.5.1 DESCRIPCIÓN

Una matriz de puertas programables o FPGA (*Field-Programmable Gate Array*), es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada en el momento, mediante un lenguaje de descripción especializado. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip. Su interior es una jerarquía de interconexiones programables que permite a los bloques lógicos ser interconectados según la necesidad del diseñador del sistema, algo parecido a una placa de inserción programable. Cada bloque lógico de pequeño tamaño se denomina celda básica o LUT y aceptara un número máximo de entradas para realizar operaciones lógicas.

Una LUT funciona como una tabla de verdad que tiene un número fijo de entradas y saca como resultado una salida de un solo bit, que puede ser registrado por un DFF (*D Flip Flop*) o puede ser utilizado para otra acción

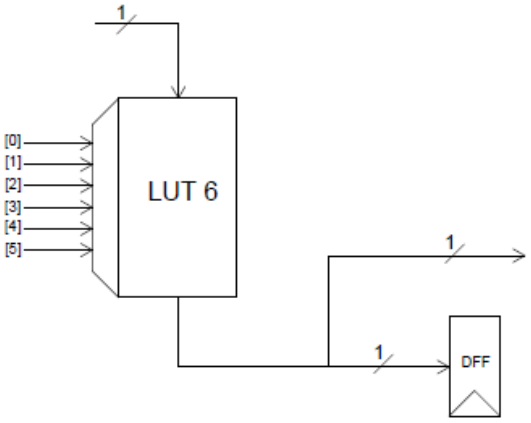


Figura 8: Esquema de una LUT-6

2.5.2 MERCADO ACTUAL

En la actualidad el mercado de los dispositivos reprogramables tipo FPGA está dominado principalmente por dos empresas que son:

- **Intel:** esta empresa ofrece actualmente 5 tipos de FPGAs:
  - **La familia Intel® Agilex ™ FPGA**, construida con tecnología de 10 nm, permite la aceleración y conectividad personalizadas para una amplia gama de aplicaciones que requieren un uso intensivo de ancho de banda y de cómputo a la vez que proporciona una mejora en el rendimiento y una reducción de la potencia.

- **La familia Intel® MAX® 10 FPGA** revolucionan la integración no volátil al ofrecer capacidades de procesamiento avanzadas en un pequeño chip de bajo coste.
- **La familia Intel® Cyclone® FPGA** está diseñada para satisfacer las necesidades de diseño de bajo consumo de energía.
- **La familia Intel® Arria®** ofrece rendimiento y eficiencia energética situándose en la gama media de todas las familias.
- **La familia Intel® Stratix® FPGA y SoC** productos de alto rendimiento y vanguardia que ofrecen una mayor productividad.

- **Xilinx:** esta empresa ofrece una alta variedad dispositivos, dentro de las FPGAs podemos encontrar los siguientes:
  - **Familia SPARTAN-6 (45nm):** ofrecen altas relaciones de lógica a pin, empaque de factor de forma pequeño, procesador MicroBlaze ™ y una gran variedad de protocolos de E/S compatibles. Ideal para aplicaciones avanzadas y la automatización industrial.
  - **Familia VIRTEX-7 (28nm):** ofrecen el mejor rendimiento de su clase, rendimiento DSP y ancho de banda de E/S para sus diseños. La familia se utiliza en una gran variedad de aplicaciones, como redes 10G a 100G, radar portátil y prototipos ASIC.
  - **Familia KINTEX-7 (28nm):** ofrecen la mejor relación rendimiento/precio en un chip de 28 y soporta los estándares principales como PCIe® Gen3 y 10 Gigabit Ethernet. Ideal para aplicaciones que incluyen soluciones inalámbricas 3G y 4G, pantallas planas y soluciones de video sobre IP.
  - **Familia ARTIX-7 (28nm):** proporcionan los circuitos de mayor rendimiento por vatio, llevan un procesador MicroBlaze ™ y el soporte DDR3 de 1,066Mb/s. Se pueden utilizar para múltiples aplicaciones como cámaras de visión artificial y backhaul inalámbrico de gama baja.
  - **Familia SPARTAN-7 (28nm):** ofrecen el mejor rendimiento en su clase por vatio,para cumplir con los requisitos más estrictos. Estos dispositivos cuentan con un procesador de software MicroBlaze ™ que ejecuta más de 200 DMIP con soporte de 800Mb/s DDR3 integrado en tecnología de 28nm. Además, los dispositivos Spartan-7 ofrecen un ADC integrado, funciones de seguridad dedicadas y calidad Q (-40 a + 125 °C). Estos dispositivos son ideales para aplicaciones industriales, de consumo y automovilísticas, ya que disponen de una conectividad muy amplia y fusión de sensores.

También ofrecen productos de alto rendimiento en tamaños 20nm y 16nm:

- Familia VIRTEX UltraSCALE (20nm):
- Familia KINTEX UltraSCALE (20nm):
- Familia VIRTEX UltraSCALE + (16nm):
- Familia KINTEX UltraSCALE + (16nm):

2.5.3 PLATAFORMA ATLYS CON FPGA SPARTAN-6

Para este proyecto se recomienda utilizar la plataforma *Atlys* de la marca *Digilent* basada en una FPGA de la marca *Xilinx* modelo *Spartan-6 LX45.*

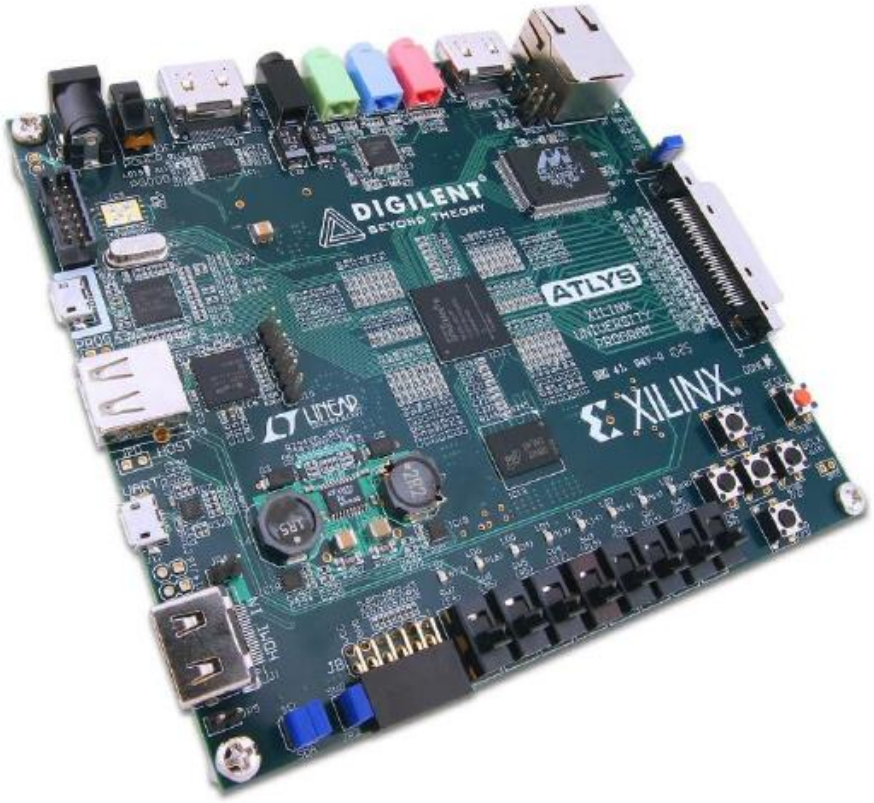


Figura 9: Plataforma Atlys.

De entre todas estas prestaciones que para futuros avances serán muy útiles, para este proyecto las más significativas son los 8 *switches* con lo que se podrán generar entradas de datos de 8 bits (1 bit por switch) y el módulo que lleva integrado para la generación señales de reloj desfasadas unas de otras 90º con total exactitud, este módulo no está incluido en otras plataformas, por lo que ha sido uno de los factores decisivos para apostar por la plataforma *Atlys*. También hay que tener en cuenta que dispone de un puerto de conexión micro USB para cargar los programas directamente desde el ordenador.

3 DESARROLLO DEL TFG

3.1 MARCO DEL PROYECTO

El Departamento de Tecnología Electrónica de la Universidad de Valladolid investiga como mejorar el rendimiento de transformación de energía eléctrica proveniente de paneles fotovoltaicos mediante un convertidor de potencia distribuido, lo que se pretende es transformar la corriente eléctrica de continua a alterna a la salida de cada panel fotovoltaico, con el objetivo de maximizar el rendimiento de cada uno, ya que debido a los procesos de fabricación, de ensamblado y a la no uniformidad en la composición de los materiales fotovoltaicos, no todos los paneles tienen las mismas curvas de tensión y corriente, por lo unos serán capaces de entregar más potencia que otros.

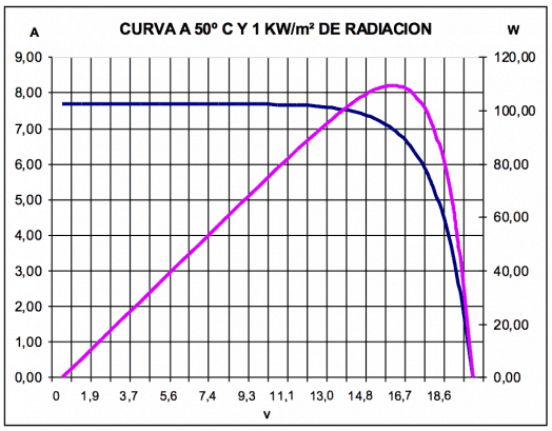


Figura 10: Curva de potencia de un panel fotovoltaico.

Para cumplir con el objetivo propuesto, se apuesta por utilizar la estructura de un convertidor modular multinivel (véase Figura 5) pero con otro funcionamiento interno, se controlará la conmutación de los *switches* de cada módulo a la frecuencia correspondiente para obtener a la salida un frente de onda estable. La función de los *switches* será la de conectar o desconectar el suministro de tensión de un módulo paneles fotovoltaicos de manera que la tensión media evolucione según lo requiera el convertidor para crear el frente de onda senoidal. Para que este sistema funcione se necesita imprescindiblemente de un enlace de comunicaciones que conecte a todos los módulos y transmita la información lo suficientemente rápido y con una tasa de errores muy pequeña, de manera que se controlen con total precisión las conmutaciones de los *switches* y así obtener una transformación con la menor distorsión posible.

3.2 DESARROLLO DEL CANAL DE COMUNICACIONES

Antes de comenzar con la descripción de todos los aspectos, conviene mencionar que la metodología de trabajo ha sido *"bottom-up"*, es decir se ha diseñado cada parte del proyecto desarrollando primero los aspectos más básicos para finalmente conseguir el objetivo final propuesto.

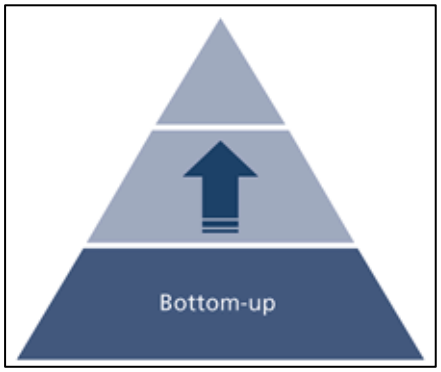


Figura 11: Método de trabajo "Bottom-up".

Un canal de comunicaciones está formado por un emisor un receptor y un medio físico en el que se propaga la información. Para este proyecto se apuesta por la fibra óptica como medio físico de transmisión de datos, ya que poner en marcha el esquema del convertidor de potencia distribuido generará mucho ruido eléctrico a la entrada y salida de cada panel o grupo de paneles fotovoltaicos, el cual podría provocar distorsiones en la comunicación si se utilizase un medio de transmisión que utilice ondas electromagnéticas.

Para el enlace de comunicaciones se ha optado por una comunicación serie unipolar usando una codificación 4b5b con el objetivo de simplificar el diseño y programación de los circuitos y evitar problemas de equilibrado de tensión que requieren otros tipos de codificaciones de mayor complejidad para trabajar a la misma velocidad de transferencia de datos. El hecho de elegir una comunicación serie es porque la aplicación a la que va destinado dicho enlace de comunicaciones requiere transmitir datos a largas distancias, del orden de cientos de metros por lo que una comunicación en paralelo supondría una cantidad de cableado mucho mayor con el correspondiente aumento del coste del proyecto.

Para elegir la velocidad de transmisión, se ha tomado como referencia orientativa las siguientes cuentas:

- Transmitiendo a una velocidad de 50 Mbps, se trabaja con una señal de reloj de 20 ns por lo que suponiendo que se transmitiesen mensajes de 32 bits codificados en 4b5b se necesitarían 40 clk para la transmisión de cada mensaje más los correspondientes ciclos de reloj para transmitir las palabras de sincronización, inicio y fin de mensaje, las cuales se estima que ocupen unos 35 clk. Sumando el tiempo de todos

componentes necesarios para enviar un mensaje de 32 bits se tardaría unos 75 clk y como cada ciclo de reloj equivale a 20 ns a la velocidad propuesta, estaríamos hablando de 75 clk * 20 ns (50Mbps) = 1,5 microsegundos.

Si la instalación estuviese formada por 10 módulos interconectados entre sí, es decir cada módulo se encarga de enviar el mensaje que le llega al módulo siguiente, estaríamos hablando de aproximadamente 10*1,5 µs = 15 µs en recorrer todos los módulos y volver al ordenador central, siendo este el menor tiempo en el que se podrían realizar las conmutaciones de cada módulo. Luego finalmente se llega a que en una instalación de 10 paneles trasmitiendo mensajes a 50 Mbps el convertidor de potencia distribuido podría trabajar a una frecuencia de $1/(15*e^{-6})$ = 66 KHz

Realizando unas cuentas similares, pero en vez de 10 módulos con 100, se necesitarían 100*1,5 µs = 150 µs para transmitir un mensaje por todos los módulos y volver al transmisor central. Luego el convertidor podría funcionar a una frecuencia máxima 6,6 KHz, pero al tener un número elevado de módulos, la frecuencia de conmutación de cada uno de ellos puede ser mucho más lenta, ya que cada uno aporta un pequeño incremento de tensión al convertidor. Es por ello que transmitir a 50Mps seria también viable para instalaciones de mayor tamaño.

La codificación 4b5b (4bits→5bits) que cumple para transmitir a 50 Mbps y permite enviar 16 combinaciones de paquetes de datos (2^n donde n es el número de bits, para n=4 →16 combinaciones) diferentes codificados como paquetes de 5 bits. De las 32 combinaciones posibles que ofrecen los 5 bits (2^5=32) se han elegido para la transmisión de datos aquellas que como máximo tienen dos ceros o dos unos al comienzo del paquete, como máximo tiene dos ceros o dos unos al final del paquete y que tengan un equilibrio 3-2, es decir como máximo tres 1's o tres 0's en cada palabra de 5 bits, de esta forma se limitaran en gran medida las posibilidades de tener más de 4 bits iguales seguidos, se podría pensar que es imposible tener más de 4 bits seguidos iguales utilizando las condiciones descritas, pero existe una tasa de error en los bits que se trasmiten, esto se denomina *Bit Error Rate (BER)* y hay que tenerlo en cuenta a la ha hora diseñar el código del receptor chequeando todos los paquetes de datos que se transmiten. Las 16 combinaciones restantes que han sido descartadas para la transmisión de datos se utilizaran como palabras de control para la sincronización, inicio de paquete de datos y fin de paquete de datos. A continuación, se muestra una tabla con las combinaciones elegidas para cada cosa:

Utilidad	Codificación 5 bits
Trama sincronización A	11111
Trama sincronización B	00000
Trama inicio A	11000
Trama inicio B	11100
Trama finalización A	10111
Trama finalización B	10000
Mensaje 0	11010
Mensaje 1	00101
Mensaje 2	00110
Mensaje 3	01001
Mensaje 4	01010
Mensaje 5	01011
Mensaje 6	01100
Mensaje 7	01101
Mensaje 8	01110
Mensaje 9	10001
Mensaje 10	10010
Mensaje 11	10011
Mensaje 12	10100
Mensaje 13	10101
Mensaje 14	10110
Mensaje 15	11001

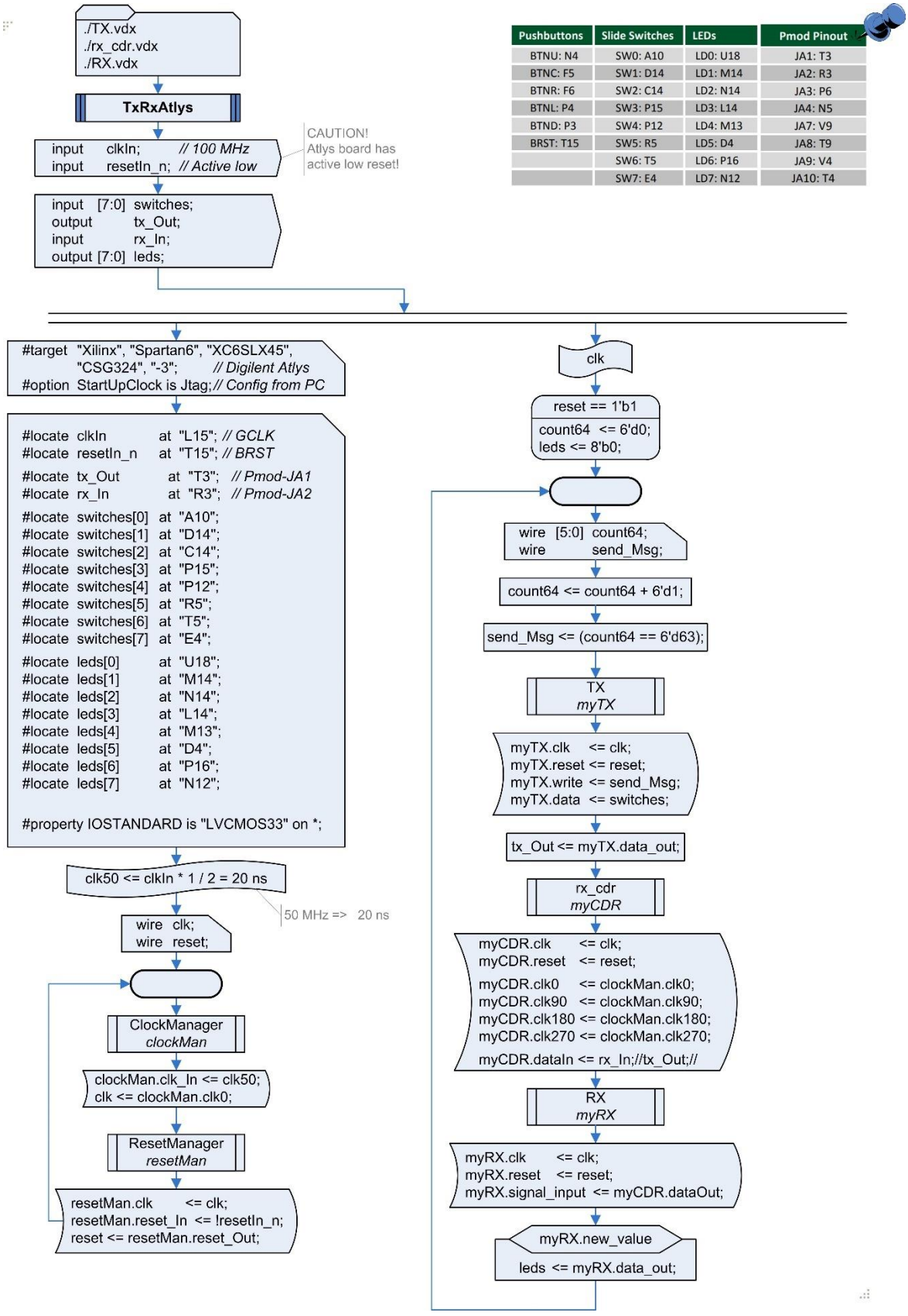
Tabla 1: Codificación 4b5b.

Las palabras de inicio y finalización están elegidas a conciencia, las primeras rompen con la trama sincronización, pasamos de estar trasmitiendo cinco unos y cinco ceros a transmitir dos unos y tres ceros y seguidamente tres unos y dos ceros, esto provoca un cambio brusco en la señal a nivel eléctrico, por lo que es más fácil de reconocer por el receptor. Las palabras de finalización son un paso fluido hacia la trama de sincronización, se comportan así cuando se envían como bloque 10 bits donde se puede observar la presencia de cuatro unos seguidos de cuatro ceros (10 bits →1011110000) que dan paso a la trama sincronización (10 bits →1111100000).

3.3 DESARROLLO DEL CIRCUITO GENERAL

Uno de los objetivos del proyecto es introducir el código que controlará el enlace de comunicaciones, en un dispositivo reconfigurable tipo FPGA, para ello hay que diseñar un circuito digital siguiendo los criterios de las máquinas de estados finitos. Una máquina de estados es un modelo computacional capaz de procesar automáticamente información recibida por una entrada para producir una salida, el diseño de estas máquinas se realiza por métodos como el de la máquina de estados algorítmica (*ASM = Algorith State Machine*).

A continuación, se muestra el diseño que engloba todos los componentes del enlace de comunicaciones más adelante se explicara detalladamente cada módulo por separado. Este primer diseño cuenta con nuevos componentes a nivel de programación que no están definidos en el apartado 2.4 de este mismo documento, pero serán descritos a medida que se explique cada parte del mismo.



Pushbuttons	Slide Switches	LEDs	Pmod Pinout
BTNU: N4	SW0: A10	LD0: U18	JA1: T3
BTNC: F5	SW1: D14	LD1: M14	JA2: R3
BTNR: F6	SW2: C14	LD2: N14	JA3: P6
BTNL: P4	SW3: P15	LD3: L14	JA4: N5
BTND: P3	SW4: P12	LD4: M13	JA7: V9
BRST: T15	SW5: R5	LD5: D4	JA8: T9
	SW6: T5	LD6: P16	JA9: V4
	SW7: E4	LD7: N12	JA10: T4

Figura 12: Diseño para implementación en FPGA.

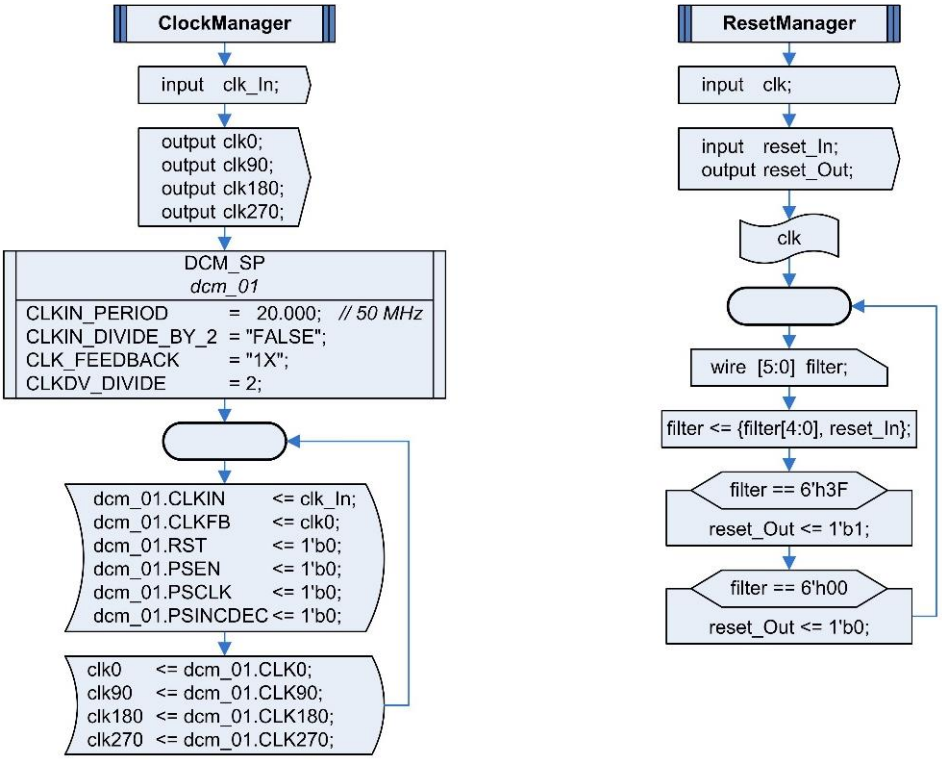


Figura 13: Diseños ClockManager y ResetManager.

El diseño de **ClockManager** se encarga de gestionar las señales de reloj generadas por el módulo que integra la FPGA, el cual permite generar las 4 señales de reloj desfasadas 90º entre ellas.

El diseño **ResetManager** se encarga de gestionar la señal de *"reset"*. Lo que hace básicamente es verificar que la señal de *"reset"* se ha mantenido activa al menos 6 ciclos de reloj, si esto se cumple se activa la señal de *"reset"*, si no se cumple se mantiene desactivada. Es una forma de evitar malinterpretaciones ya que bajo determinadas circunstancias como por ejemplo la presencia de ruido eléctrico, existe la posibilidad de que alguna señal sufra una alteración, en el caso de la señal *"reset"* si se hiciera caso a su variación en un único ciclo de reloj, ante cualquier alteración se podría reiniciar el circuito sin necesidad.

Nada más empezar a leer el diseño **TxRxAtlys** nos encontramos con el primer elemento distinto que se ha introducido en estos diseños, esta caja que se encarga de buscar los ficheros que se nombran en su interior en el mismo directorio en que se ha creado el diseño en el que se está trabajando, en este caso el diseño TxRxAtlys.

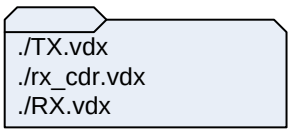


Figura 14: Caja de búsqueda de ficheros.

La siguiente novedad se encuentra en las cajas de código, donde se introduces tres formas nuevas de realizar declaraciones, cada una ella tiene una funcionalidad específica y además repercuten a la hora de compilar el diseño, debido a que su implementación genera nuevos ficheros que explicare más adelante.

```
#target  "Xilinx", "Spartan6", "XC6SLX45",
         "CSG324", "-3";        // Digilent Atlys
#option  StartUpClock is Jtag;// Config from PC
```

Figura 15: Declaración de la tarjeta FPGA y configuración de la misma.

#target: este comando se utilizar para declarar las especificaciones del tipo de tarjeta con la que se va a trabajar. Cuando el compilador detecta un comando de este tipo automáticamente genera unos ficheros tipo **[.ucf] [.bat] y [.tcl]** que son imprescindibles para el proceso de síntesis de la FPGA. Mas adelante se contará su utilidad.

#option: este comando se utilizar para declarar como se cargarán los diseños en la FPGA. Esta acción puede hacerse de dos formas:

- De forma directa a través de *Jtag*. Trabajando de esta forma se cargan los códigos en la memoria RAM de la FPGA y son útiles mientas el dispositivo este bajo tensión, una vez se desconecta se pierde la información de estos.
- De forma indirecta a través de la *EEPROM*. Trabajando de esta forma se cargan los códigos en la memoria *EEPROM* de la FPGA, la cual se comporta como una memoria no volátil por lo que cuando se desconecta el dispositivo mantiene la información de los códigos.

Lo siguiente con lo que nos encontramos en el diseño es la declaración de los aspectos físicos de la FPGA, es decir por donde se van a transportar las señales, y también se declara bajo que tensión va a trabajar el circuito.

```
#locate  clkIn          at  "L15"; // GCLK
#locate  resetIn_n      at  "T15"; // BRST
#locate  tx_Out           at  "T3";   // Pmod-JA1
#locate  rx_In            at  "R3";   // Pmod-JA2
#locate  switches[0]  at  "A10";
#locate  switches[1]  at  "D14";
#locate  switches[2]  at  "C14";
#locate  switches[3]  at  "P15";
#locate  switches[4]  at  "P12";
#locate  switches[5]  at  "R5";
#locate  switches[6]  at  "T5";
#locate  switches[7]  at  "E4";
#locate  leds[0]        at  "U18";
#locate  leds[1]        at  "M14";
#locate  leds[2]        at  "N14";
#locate  leds[3]        at  "L14";
#locate  leds[4]        at  "M13";
#locate  leds[5]        at  "D4";
#locate  leds[6]        at  "P16";
#locate  leds[7]        at  "N12";

#property IOSTANDARD is "LVCMOS33" on *;
```

Figura 16: Declaración de pines físicos de la FPGA y tensión de trabajo.

Para declarar los aspectos físicos de la FPGA se utiliza el comando **#locate** seguido de la señal con la que se quiere trabajar y después por donde se va a transmitir, por ejemplo *(#locate tx_Out* at "T3") quiere decir que la señal *"tx_Out"* se transmitirá por el pin "T3" de la FPGA.

El comando ***#property*** se utilizar para declarar bajo que tensión trabajara la FPGA.

Una vez completada la asignación de los componentes físicos de la FPGA, se le asigna a la señal de reloj un valor, en nuestro caso como se va a transmitir a 50Mbps se necesita una señal de reloj de 20ns.

```
clk50 <= clkIn * 1 / 2 = 20 ns
```

Figura 17: Asignación de la señal de reloj.

El diseño TxRxAtlys como se había mencionado anteriormente reúne todos módulos que hacen viable la transmisión de datos. Estos módulos tienen sus propias entradas y salidas al igual que el diseño en él que nos encontramos ahora, por lo que para que funcionen mutuamente hay que asignar individualmente el valor que corresponda a las señales de entrada de cada diseño. Por ejemplo, se muestra a continuación la asignación de las entradas del circuito transmisor.

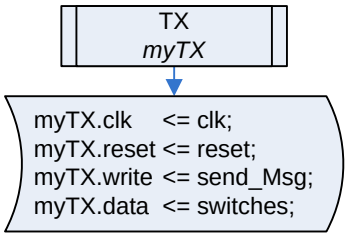


Figura 18: Implementación del módulo TX en el diseño TxRXAtlys.

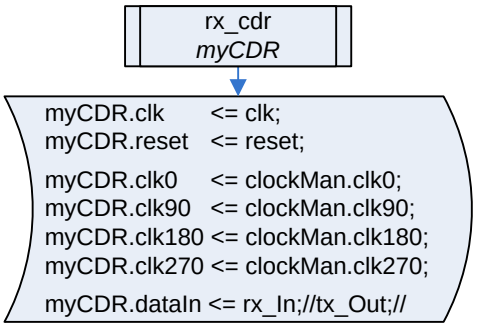


Figura 19: Implementación del módulo rx_cdr en el diseño TxRxAtlys.

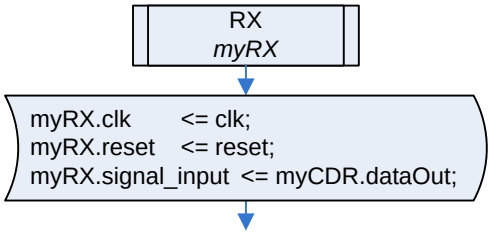


Figura 20: Implementación del módulo RX en el diseño TxRxAtlys

3.4 DESARROLLO DEL CIRCUITO TRANSMISOR.

El transmisor es uno de los elementos fundamentales del canal de comunicaciones. Su función es transmitir los paquetes de datos que entran al sistema desde el exterior. A continuación, se muestra dos imágenes globales del diseño y después se explicarán detalladamente todos los aspectos de este.

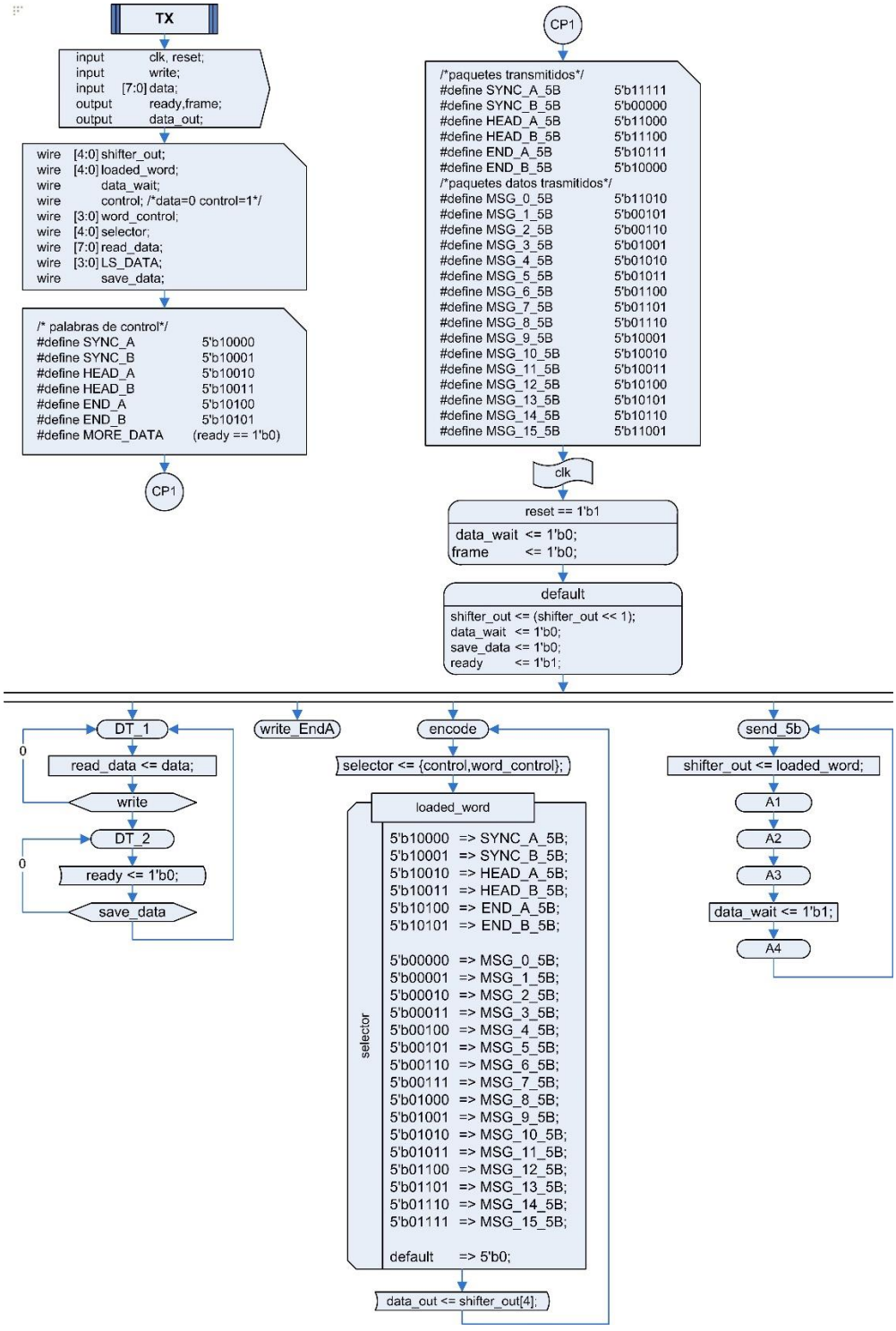


Figura 21: Pagina 1 del diseño del transmisor.

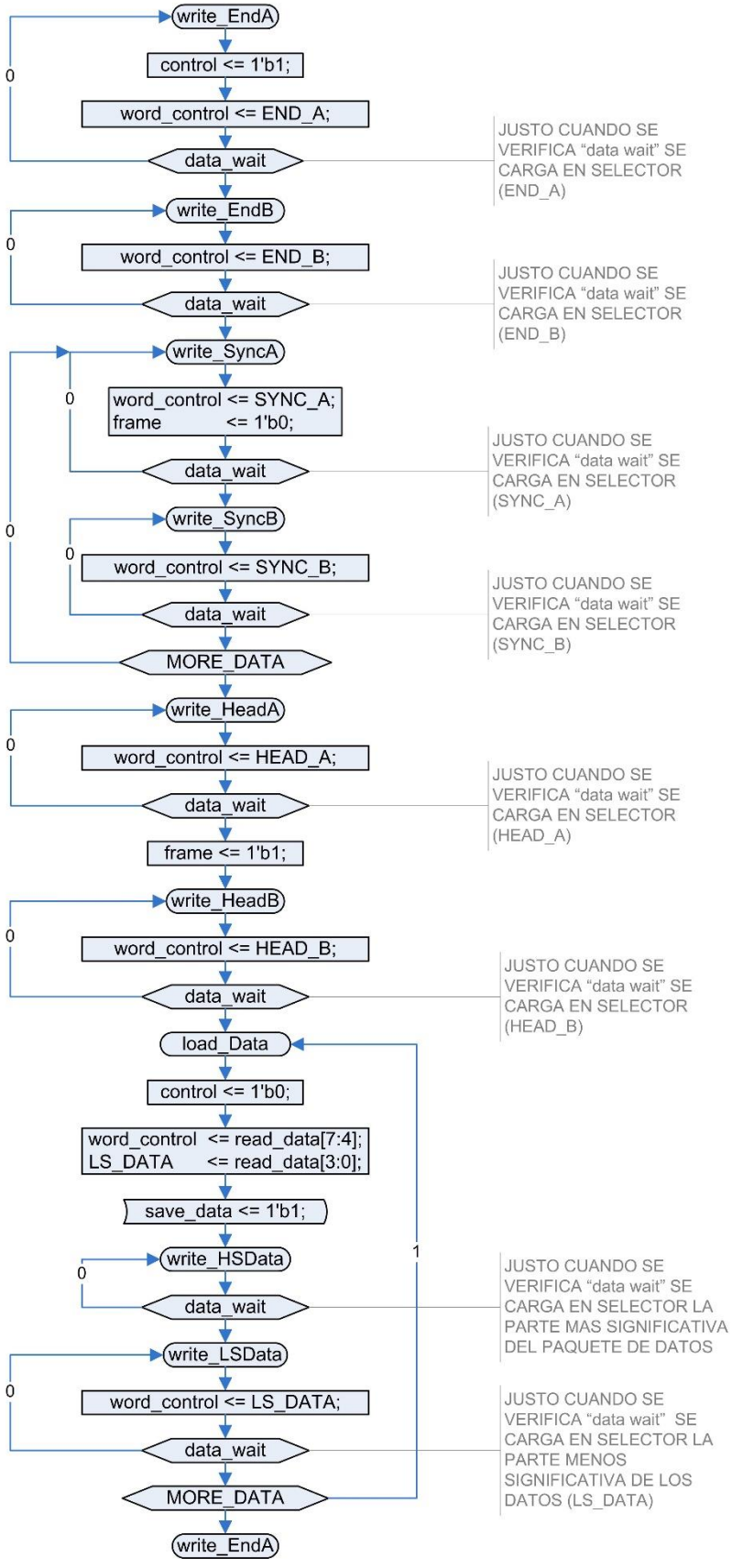


Figura 22: Pagina 2 del diseño del transmisor.

Como se puede observar en las imágenes adjuntadas anteriormente el diseño del transmisor de paquetes de datos tendrá tres entradas:

- ***clk:*** la señal de reloj para sincronizar el sistema
- ***write:*** es una señal de un bit que nos dirá cuando se quiere escribir un dato.
- ***data:*** es una señal de ocho bits que nos dirá que paquete de datos se quiere enviar.

Y tendrá tres salidas:

- ***ready:*** es la señal que nos dirá cuando el circuito está listo para recibir más paquetes de datos que posteriormente serán transmitidos
- ***data_out:*** es la señal de salida de bits de forma secuencial.
- ***frame:*** es la señal que indicara a los circuitos exteriores durante que ciclos de reloj se está produciendo la transmisión de datos.

En el entorno grafico de *MsVisio* esto se implementa de la siguiente forma:

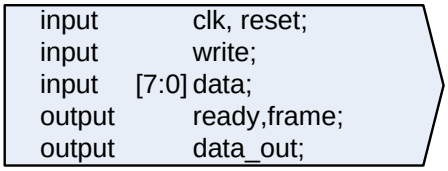


Figura 23: Bloque de definición de entradas y salidas ASM++.

Internamente el circuito cuenta con nueve señales, que son utilizadas para procesar los datos. A continuación, se muestra el bloque de declaración de señales internas del entorno de diseño ASM++:

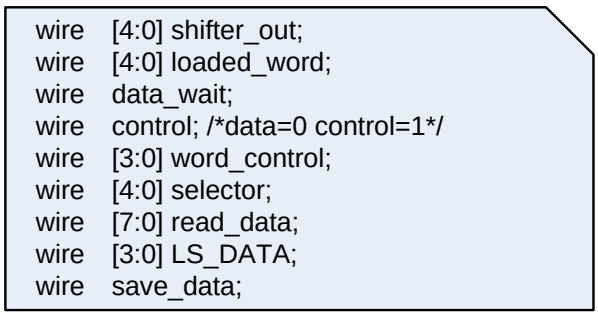


Figura 24: Bloque declaración señales internas.

Descripción de cada una de las señales:

- ***shifter_out:*** señal utilizada como vector de desplazamiento para mostrar de forma secuencial los paquetes de datos.

- ***loaded_word:*** señal utilizada como variable para guardar internamente cual es el paquete de datos que se quiere transmitir.
- ***data_wait*:** es la señal que marca ritmo de funcionamiento del circuito, su función es ser activada internamente cuando se haya procesado un paquete de datos y se pueda cargar uno nuevo.
- ***control:*** esta señal se utiliza para dar carácter a la codificación 4b5b, cuando tiene el valor 1 todos los paquetes que se cargan son adquieren la función de control, ya sea sincronización, inicio o finalización del paquete de datos. Cuando adquiere el valor 0 quiere decir que los paquetes de bits que se están manejando son mensajes con una función útil.
- ***word control:*** señal utilizada para indicar cual es el paquete de datos que se quiere transmitir.
- ***read_data:*** señal interna que almacena los paquetes de datos útiles que entran desde el exterior a través de "data".
- ***LS_DATA:*** esta señal de cuatro bits se encarga de almacenar la parte menos significativa (*Low Significant*) del paquete de datos que entra por *"data".*
- ***save_data*:** esta señal de un bit nos indica cuando se procesado un paquete de datos proveniente de "data" y se utiliza para saber cuándo se puede almacenar internamente un nuevo valor de la señal *"data".*

Para utilizar la codificación 4b5b se han definido todas las combinaciones de la siguiente forma:

```
/* palabras de control*/
#define SYNC_A          5'b10000
#define SYNC_B          5'b10001
#define HEAD_A          5'b10010
#define HEAD_B          5'b10011
#define END_A           5'b10100
#define END_B           5'b10101
#define MORE_DATA       (ready == 1'b0)
```

Figura 25: Bloque de definición de palabras de control.

Las palabras SYNC_A, HEAD_A, END_B y demás de la tabla son utilizadas en el código para referirse a los 5bits que aparecen asignados a la izquierda de cada palabra, más adelante se observará la función de este renombramiento.

```
/*paquetes transmitidos*/
#define SYNC_A_5B              5'b11111
#define SYNC_B_5B              5'b00000
#define HEAD_A_5B              5'b11000
#define HEAD_B_5B              5'b11100
#define END_A_5B               5'b10111
#define END_B_5B               5'b10000
/*paquetes datos trasmitidos*/
#define MSG_0_5B               5'b11010
#define MSG_1_5B               5'b00101
#define MSG_2_5B               5'b00110
#define MSG_3_5B               5'b01001
#define MSG_4_5B               5'b01010
#define MSG_5_5B               5'b01011
#define MSG_6_5B               5'b01100
#define MSG_7_5B               5'b01101
#define MSG_8_5B               5'b01110
#define MSG_9_5B               5'b10001
#define MSG_10_5B              5'b10010
#define MSG_11_5B              5'b10011
#define MSG_12_5B              5'b10100
#define MSG_13_5B              5'b10101
#define MSG_14_5B              5'b10110
#define MSG_15_5B              5'b11001
```

Figura 26: Bloque de definición de los paquetes de datos transmitidos.

En esta figura se definen los 5 bits que se transmitirán según la palabra que se elija con la codificación 4b y la palabra de control.

Figura 27: Caja de sincronización.

Esta caja se encarga de sincronizar según la señal de reloj asignada todas las cajas que vayan después de ella.

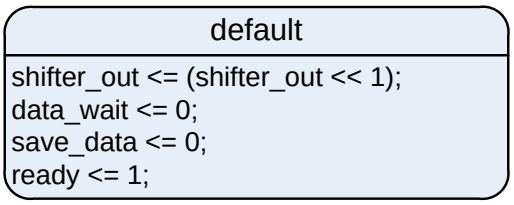


Figura 27: Caja para especificar valores por defecto.

La caja de *"default"* se encarga de asignar valores por defecto a todas las señales síncronas y asíncronas siempre que no se le asigne ningún otro valor mientras se está ejecutando el código. Tiene aplicaciones interesantes cuando se quiere dar un carácter concreto a una señal durante estados diferentes.

Las primeras consideraciones que hay que tener claras para entender los diagramas ASM++ son los dos tipos de operaciones básicas que conforman los mismos. Las primeras son las operaciones síncronas, las cuales almacenan

en un registro el resultado de la operación realizada y comienza a ser visible para el resto del circuito, un ciclo de reloj más tarde que cuando ha sido descrita la operación. El resultado permanece guardado hasta que es cambiado en otro estado distinto En los diagramas ASM++ estas operaciones vienen dadas por el siguiente tipo de caja:

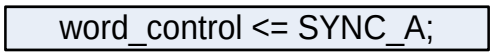


Figura 28: Ejemplo de operación síncrona.

Las segundas son las operaciones asíncronas, en las cuales se realiza la operación y se muestra su resultado únicamente durante el propio ciclo de ejecución, en cuanto se salta a otro estado el valor asíncrono del estado anterior desaparece y en el nuevo estado adquirirá un nuevo valor. En los diagramas ASM++ estas operaciones vienen dadas por el siguiente tipo de caja:

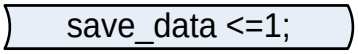


Figura 29: Ejemplo de operación asíncrona.

En la primera página es donde se engloba todo el diseño del transmisor, el circuito está formado por 4 hilos que trabajan de forma síncrona bajo la misma señal de reloj. El primer hilo que comienza con "DT_1" se encarga de leer los datos del exterior y almacenarlos en la señal interna *"read_data".* Cuando se activa la señal *"write"*, este hilo del circuito se queda esperando a la señal *"save_data",* que se activara cuando se haya guardado el dato anterior pudiendo así almacenar un nuevo valor del exterior en la señal *"read_data".*

El segundo hilo del diseño, el que comienza con el estado *"write_EndA"* se encarga de gestionar el valor adquiere la señal *"word_control"* y la señal "control" dependiendo de si se quiere transmitir un paquete de datos o si se está enviando únicamente la señal de sincronización. Para tomar estas decisiones este hilo "escucha" y "habla" con el primero, cuando la señal "ready" se pone a cero, está definido en el circuito cómo:

#define MORE_DATA (ready == 1'b0)

Es decir, la palabra *MORE_DATA* significa lo mismo que *ready* == 1'b0. Cuando *MORE_DATA* está activo indica al segundo hilo que se quiere transmitir un paquete de datos. Cuando el segundo hilo guarda el paquete de datos indica al primer hilo mediante la señal *"save_data"* que puede almacenar un nuevo dato del exterior. Esta comunicación entre hilos permite una transmisión ininterrumpida de datos.

El tercer hilo del diseño funciona como un "switch" que se ejecuta durante todos los ciclos de reloj. La señal "selector" adquiere un valor asíncrono de 5 bits en cada ciclo de reloj, siendo este la fusión de los bits de la señal *"control"* y la señal *"loaded_word".* El valor de los bits de estas dos

señales será utilizado para indicar que paquete de datos de la tabla se le asigna a la señal síncrona *"loaded_word"*. Como este hilo solo tiene un estado que se ejecuta todos los ciclos de reloj, hemos incluido la siguiente operación asíncrona:

```
data_out <= shifter_out[4];
```

Figura 30: Señal de salida del transmisor.

Esta operación se encarga de asignar a la señal de salida *"data_out"* el valor del quinto bit de la señal *"shifter_out"*. Dicha señal va desplazando sus bits una posición hacia la izquierda en todos los ciclos de reloj mediante la siguiente operación definida por defecto:

```
default
shifter_out <= (shifter_out << 1);
data_wait <= 0;
save_data <= 0;
ready <= 1;
```

Figura 31: Operación de desplazamiento de la señal "shifter_out".

A continuación, se muestra una imagen del funcionamiento del desplazamiento de los bits hacia la izquierda:

Señal "shifter_out"

	bit[4]	bit[3]	bit[2]	bit[1]	bit[0]
clk 1	0	1	0	1	1
clk 2	1	0	1	1	0
clk 3	0	1	1	0	0
clk 4	1	1	0	0	0
clk 5	1	0	0	0	0

Figura 32: Simulación de desplazamiento de bits hacia la izquierda.

El cuarto hilo del diseño se encarga de cargar en la señal *"shifter_out"* el valor de la señal *"loaded_word"* que se decidió en el tercer hilo y cargar en la señal *"data_wait"* el valor 1 en un determinado ciclo de reloj. Como todos los hilos se "comunican" entre sí, tiene que estar perfectamente calculado el momento en el que se cambian los valores de las señales para que no se

detenga el flujo de transmisión de datos. La señal *"data_wait"* condiciona el funcionamiento del segundo hilo y está expresamente situada dos ciclos de reloj antes de realizar la operación síncrona de cargar en *"shifter_out"* la señal *"loaded_word"*, de esta forma se consigue que mientras se está ejecutando el estado A4 la señal *"data_wait"* vale 1 y al mismo tiempo que se está ejecutando este, estado estamos cargando de forma síncrona en la señal *"loaded_word"* los bits correspondientes y justo un ciclo de reloj más tarde del estado A4, es decir en el estado send_5b, serán visibles los bits de *"loaded_word"* y estos serán cargados en la señal *"shifter_out"* por lo que se puede observar que son cargados justo en el estado en el comienzan a ser visibles, es por esto por lo que hay que diseñar los circuitos calculando con detalle cuando y donde se modifican los valores de cada señal. En las simulaciones realizadas del circuito receptor se puede observar todo lo descrito.

3.6 SIMULACIÓN DEL CIRCUITO TRANSMISOR

Finalmente, para probar el correcto funcionamiento de todo el circuito descrito y antes de implementarlo físicamente, se han realizado una serie de pruebas con visualizador de formas de onda. Existen varios en el mercado, nosotros hemos utilizado la herramienta *"GTKWave"* con la que hemos podido visualizar el comportamiento del circuito ante distintas situaciones. Antes de utilizar la herramienta *"GTKWave"* hay que diseñar un fichero de test bench utilizando el lenguaje de ASM++, en dicho fichero se fijan todos los estímulos de las señales según el tipo de prueba que se quiera realizar. El diseño de un test bench tiene sus propias reglas que han de cumplirse para que se generen correctamente los ficheros. A continuación, adjunto una imagen de un diseño en el que se han realizado dos pruebas de transmisión de datos.

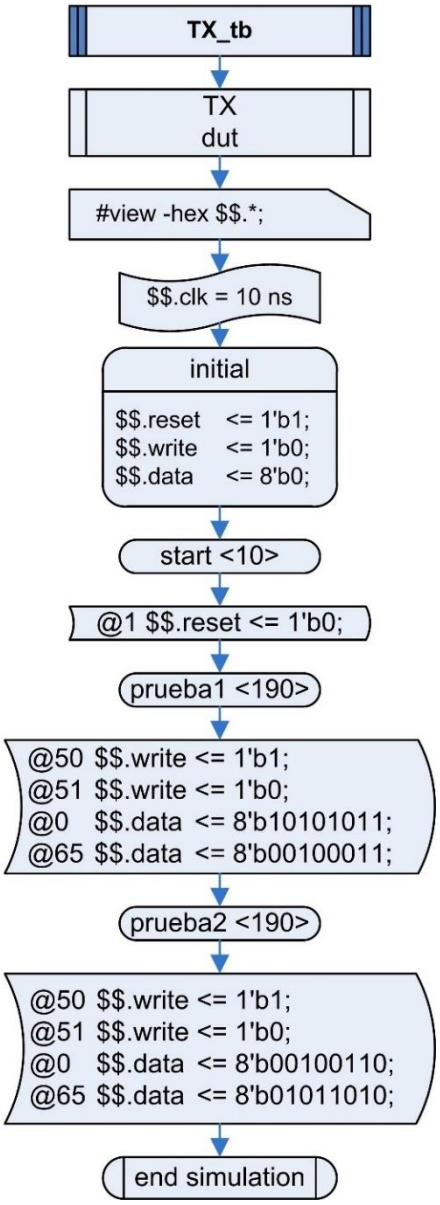


Figura 33: Diseño de test bench del transmisor.

Cuando se crea un diseño de test bench hay que implementar el módulo en el que se encuentra el diseño, esta acción se realiza con la siguiente caja.

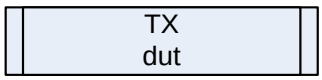


Figura 34: Implementación del código que se quiere testear.

Esta caja lo que hace es crear un código tipo TX que se llamará *"dut"* dentro del diseño del test bench.

La siguiente caja se encargará de que se visualicen todas las señales que están cargadas en el diseño del test bench.

Figura 35: Caja en la que se declara que se visualicen todas las señales.

Seguidamente se define el valor de la señal de reloj y el valor las señales de las entradas cuando arranca el circuito. Esto se implementa en las siguientes cajas.

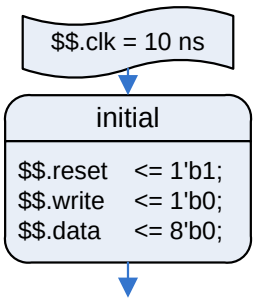


Figura 36: Declaración de la señal de reloj y las señales de entrada.

Antes de comenzar con las pruebas, lo primero que se hace es reiniciar el circuito, es decir se activa la señal de *"reset"* (caja initial) durante un ciclo de reloj. Hay que tener en cuenta que, si no se desactivan las señales, mantiene el estímulo a lo largo del test bench.

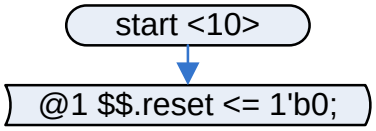


Figura 37: Desactivación de la señal "reset".

Una vez reseteado el circuito, ya se pueden realizar pruebas. La "prueba1" que se ha realizado en este test bench tiene una duración de 190 ciclos de reloj. En el ciclo 30 se activa la señal *"write"* y se carga el valor de *"data"* con los mensajes número 0 y número 1, en el ciclo 31 se desactiva *"write"*. Se realiza la misma prueba en el ciclo 90 pero enviando los mensajes número 2 y número 3.

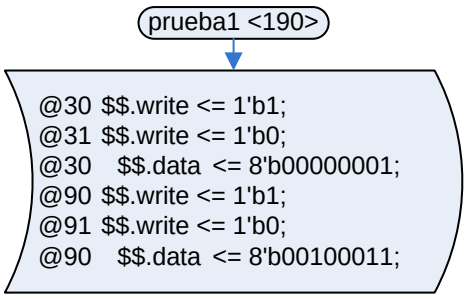


Figura 38: Prueba número 1 circuito TX.

La segunda prueba que se realiza es idéntica a la primera, pero enviando los mensajes número 9,10,11 y 12.

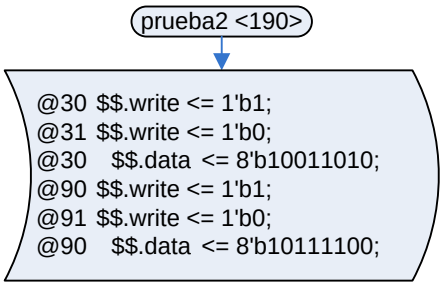


Figura 39: Prueba numero 2 circuito TX.

Una vez se tienen programadas las pruebas, se compila el diseño con la herramienta ***ASM++ Compiler*** que ya tiene integrado en su entorno el visualizador de formas de onda *GTKWave*. Si los diseños tuvieran algún fallo el compilados nos avisaría de los mismos y el test bonche no se generaría. A continuación, se adjunta una imagen del entorno de compilación en el que se ha generado correctamente la compilación del circuito TX.

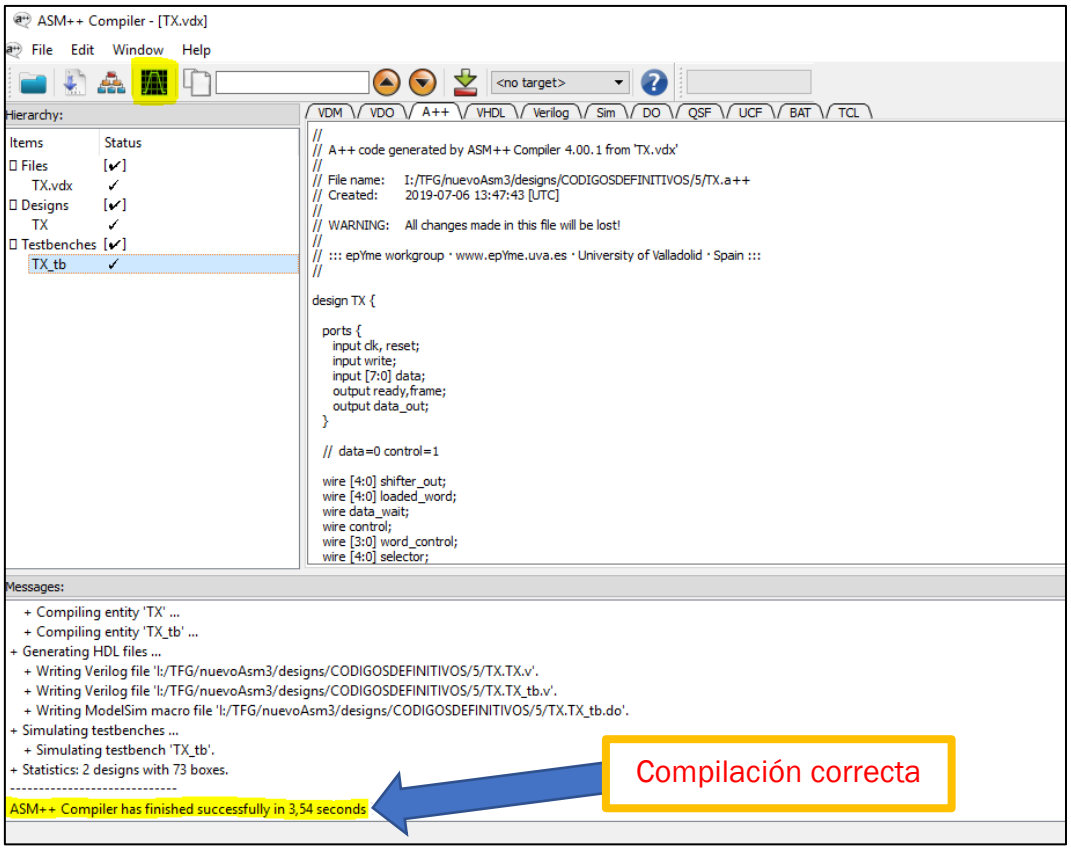


Figura 40: Compilación de los diseños TX y test bench.

Una vez se han compilado correctamente los diseños podemos comenzar a usar la herramienta *GTKWave*. A continuación, se muestran unas imágenes en las que se puede observar que las pruebas realizadas con correctas. La primera es una vista global de la transmisión de los cuatro primeros mensajes. Después adjuntare imágenes más detalladas de las distintas señales.

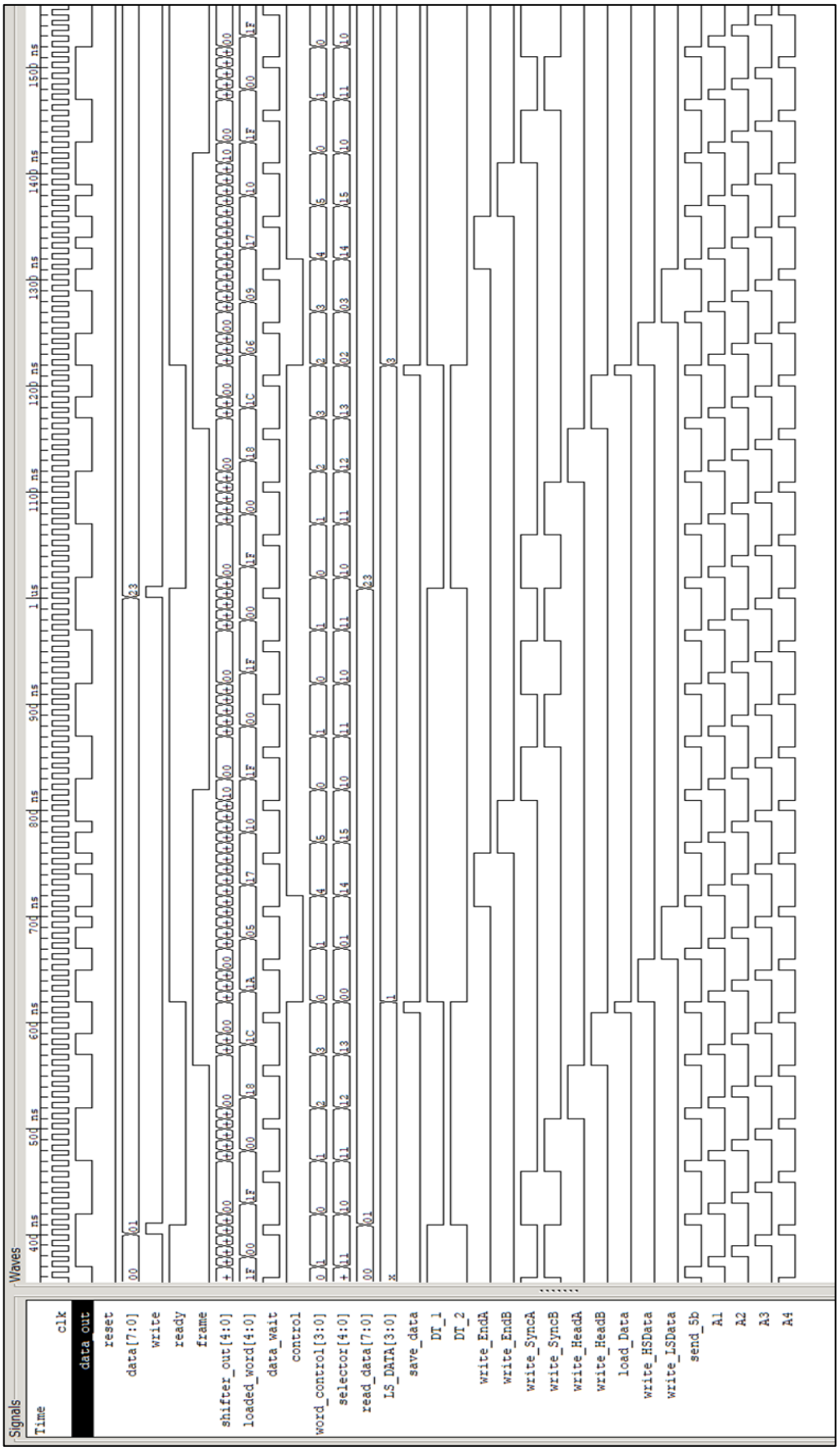


Figura 41: Vista global de la transmisión de los 4 primeros mensajes.

Justo cuando se activa la señal *"write"* se cargan los mensajes 0 y 1 en la señal *"data"*. Hay que tener en cuenta que se está visualizando la señal *"data"* en formato hexadecimal por eso nos aparece 01.

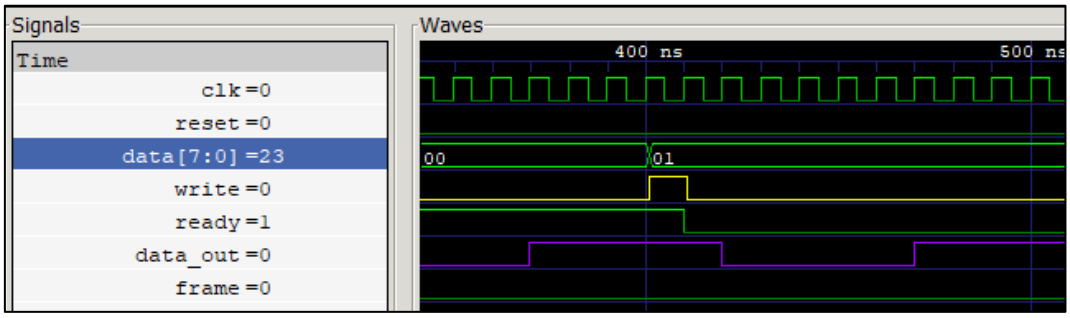


Figura 42: Activación de la señal "write" y asignación de mensajes 0 y 1 a la señal "data" (visualización formato hexadecimal).

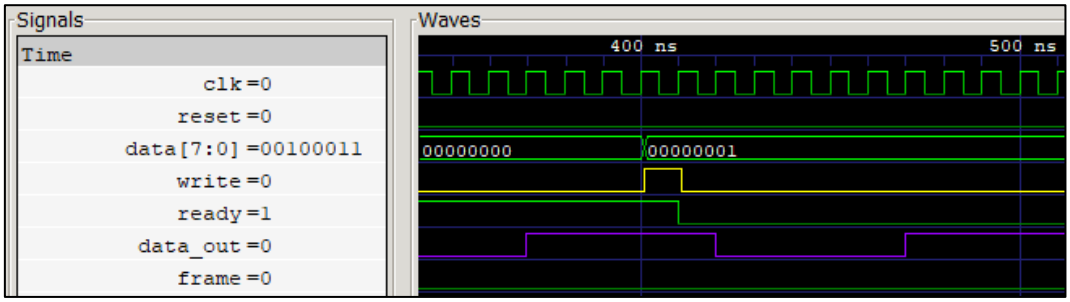


Figura 43: Activación de la señal "write" y asignación de mensajes 0 y 1 a la señal "data" (visualización formato binario).

Una vez se activa la señal *"write"* el circuito responde enviando las palabras de inicio y después los paquetes de 5 bits decodificados. Nos vamos a la tabla en la que declaramos las combinaciones de bits de cada mensaje y comprobamos que se está transmitiendo correctamente.

#define MSG_0_5B	5'b11010
#define MSG_1_5B	5'b00101

Figura 44: Codificación mensajes 0 y 1.

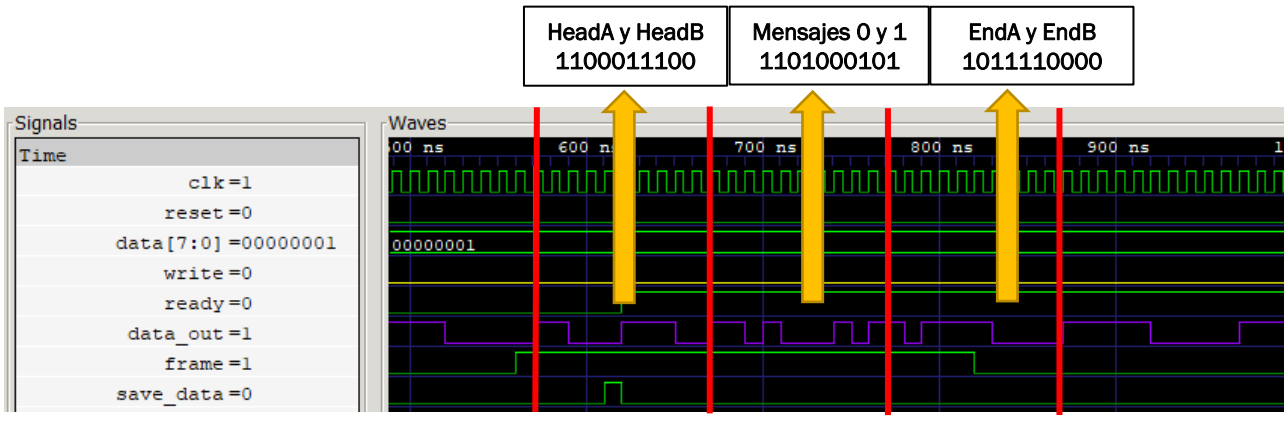


Figura 45: Transmisión de los mensajes 0 y 1.

```
#define MSG_2_5B    5'b00110
#define MSG_3_5B    5'b01001
```

Figura 46: Codificación mensajes 2 y 3.

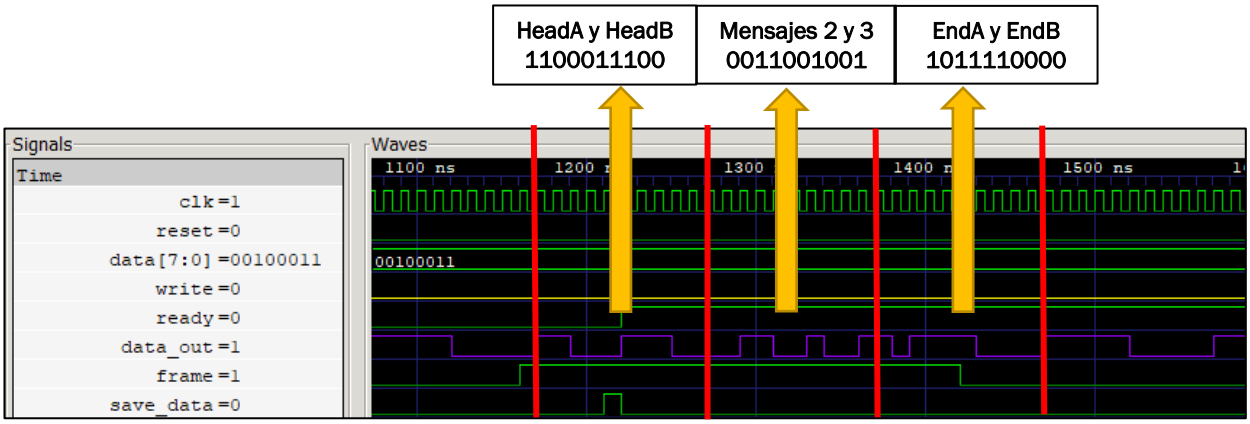


Figura 47: Transmisión de los mensajes 2 y 3.

```
#define MSG_9_5B    5'b10001
#define MSG_10_5B   5'b10010
```

Figura 48: Codificación mensajes 9 y 10.

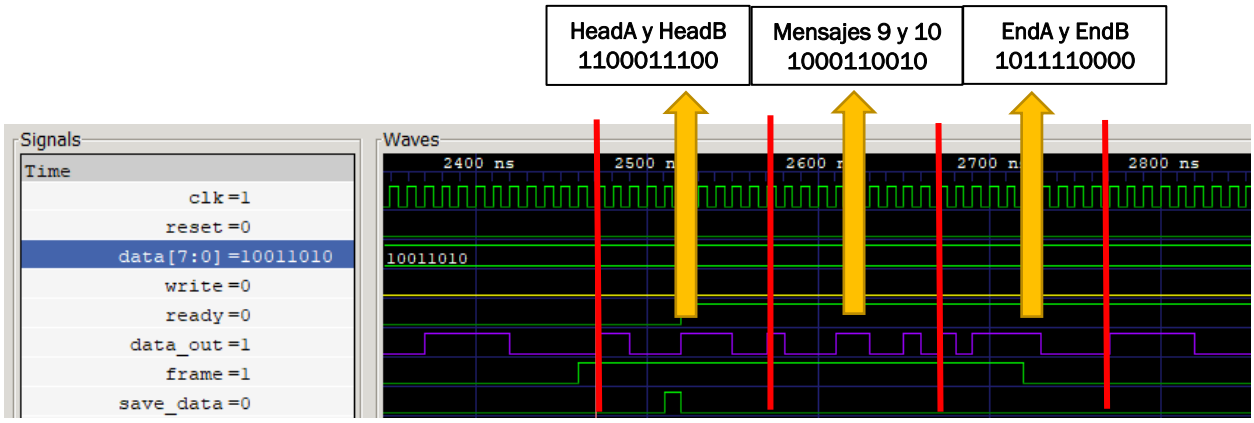


Figura 49: Trasmisión mensajes 9 y 10.

3.7 DESARROLLO DEL CIRCUITO DE RESINCRONIZACIÓN

Una vez que los datos fueron enviados por el transmisor tarda un pequeño tiempo en viajar hasta el receptor dependiendo de la longitud del cañal de transmisión. Como estamos trabajando con señales del orden de nanosegundos, cualquier pequeño retraso causará desajustes en la sincronización, es por ello que la señal que entra por el receptor, muestreada a la misma frecuencia de trabajo que el transmisor presentara un pequeño desfase. Ambos circuitos funcionan a la misma frecuencia de reloj y sus señales estarán casi sincronizadas, es por ello que estos sistemas se comportan de forma plesiocrona. Como no podemos controlar cual será el desfase exacto de la señal, surgirán problemas a la hora de detectar las transiciones de los bits, por lo que es necesario implementar un circuito que se encargue de sincronizar de forma exacta el receptor con la señal que justamente este entrando.

Dicho circuito toma muestras de cuatro en cuatro utilizando cuatro señales de reloj desfadas entre si 90º y luego decide cuál de las cuatro muestras es la más fiable por lo que funciona como una pequeña memoria que exactamente recuerda en qué estado se ha producido la anterior transición y lo compara con el estado donde se están produciendo. Si se verifica 2 veces una variación de la posición de las transiciones, se cambia de estado y como consecuencia se cambia la muestra que se está cargando en la señal *“data_Out”* (se muestra más adelante), es decir esta es la forma de elegir cual es la muestra más fiable de las 4 tomadas. Seguramente os estaréis preguntando por qué se hace todo esto, pues bien, debido a la naturaleza plesiocrona del sistema existirá una rotación cíclica de las posiciones de las transiciones a medida que avance la recopilación de datos, por lo que ira variando también la posición de la muestra más fiable. A continuación, se adjuntan unas imágenes para entender un poco mejor todo lo descrito.

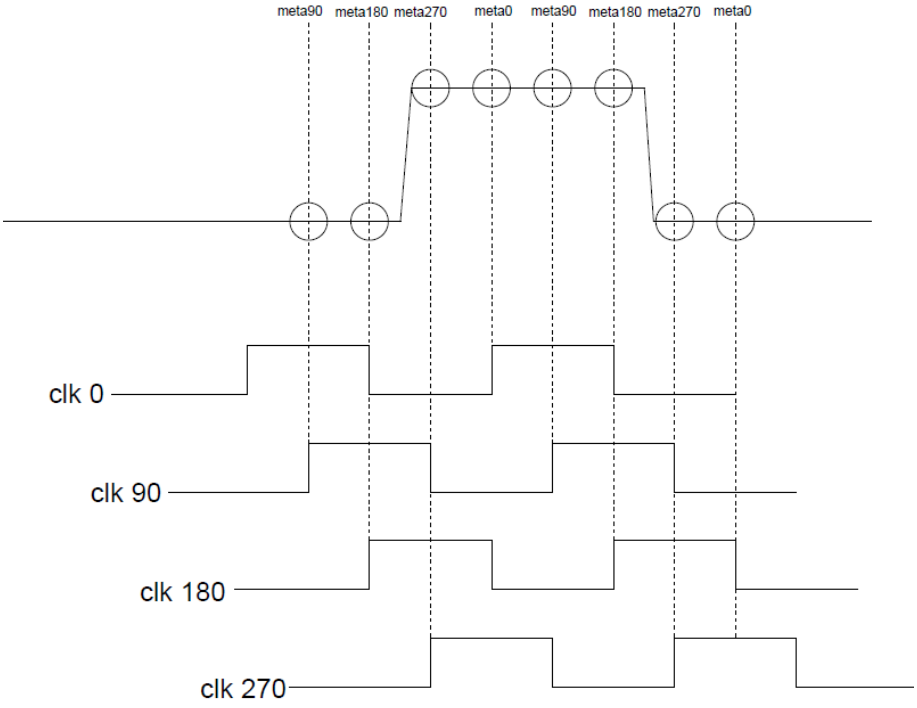


Figura 50:Transición entre meta180 y meta270.

A medida que avanza el tiempo la posición de las transiciones nos cambia, las señales creadas se van quedando atrás respecto de la señal real.

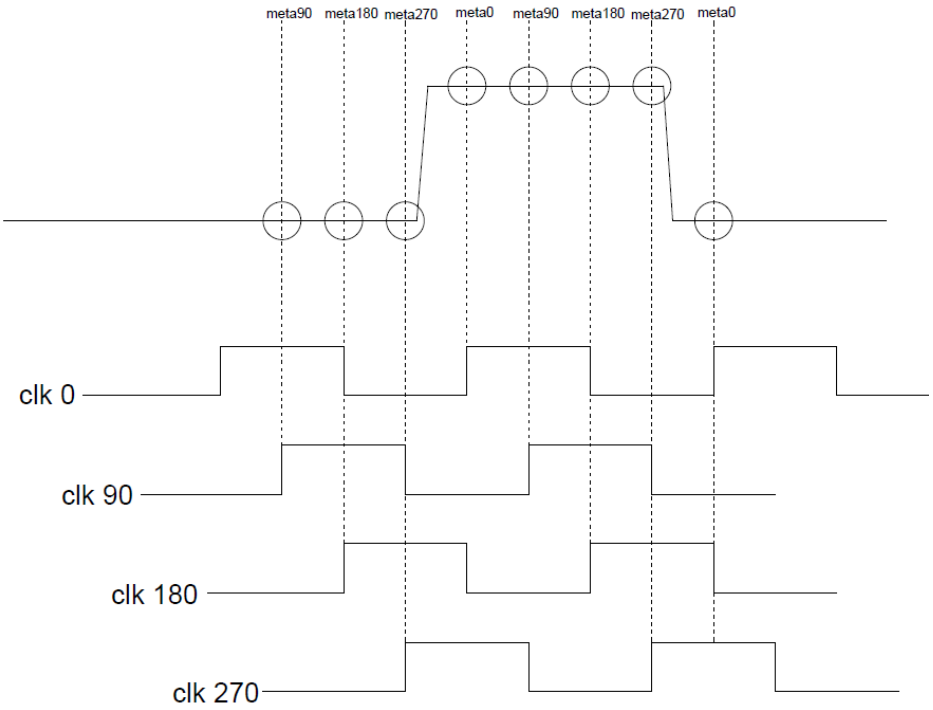


Figura 51: Transición entre meta270 y meta0.

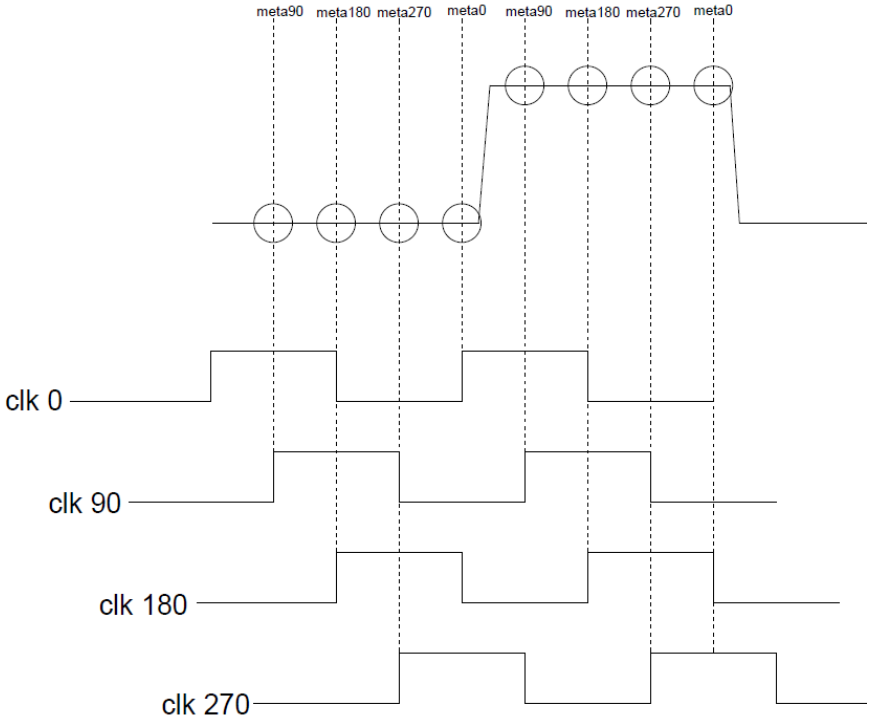


Figura 52: Transición entre meta0 y meta90.

Antes de comenzar con las descripciones del circuito resincronizador es necesario mencionar que la plataforma elegida para el proyecto permite generar las señales de reloj desfasadas 90º entre sí que se necesitan para dicho circuito. A continuación, se muestra el módulo que se encarga de gestionar las señales de reloj provenientes de la plataforma de trabajo *Atlys.* Este módulo necesita de una serie de parámetros de entrada para que la FPGA pueda gestionar las señales, todo estos serán definidos en el diseño del *clock manager.*

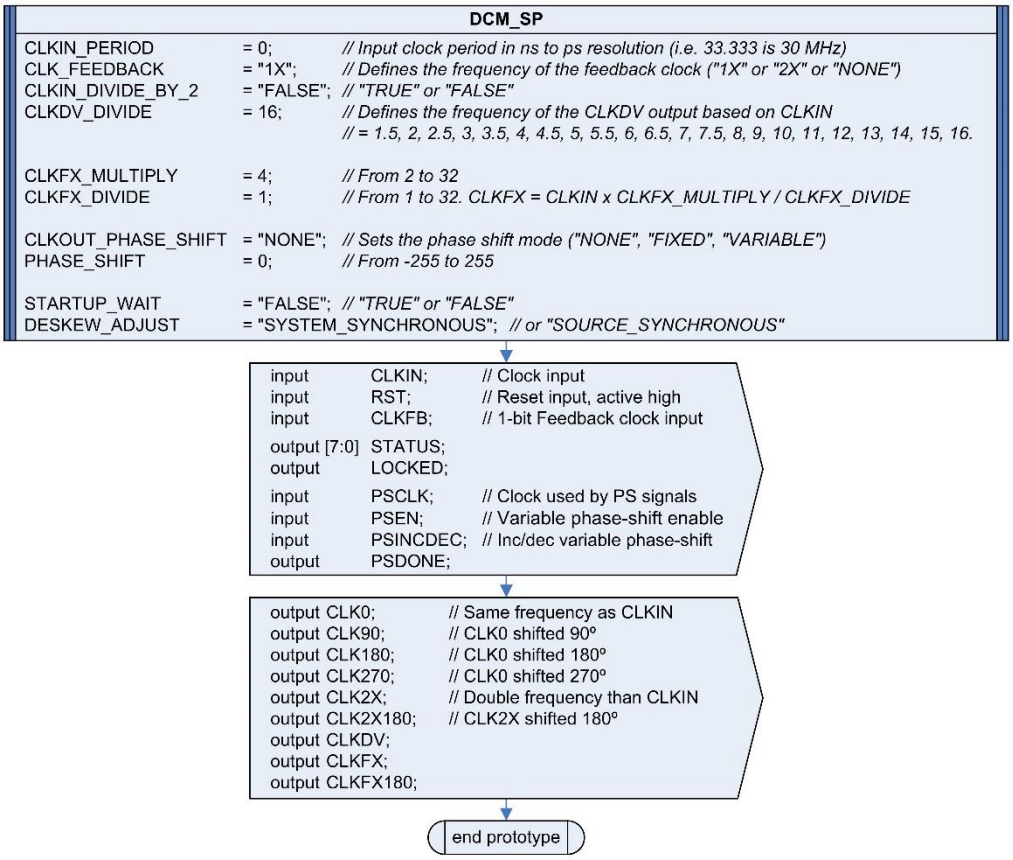


Figura 53: Modulo que controla las señales de reloj de la plataforma Atlys.

A continuación, se muestra el circuito que utiliza las señales generadas por el módulo que se acaba de describir para gestionar la rotación cíclica de las transiciones.

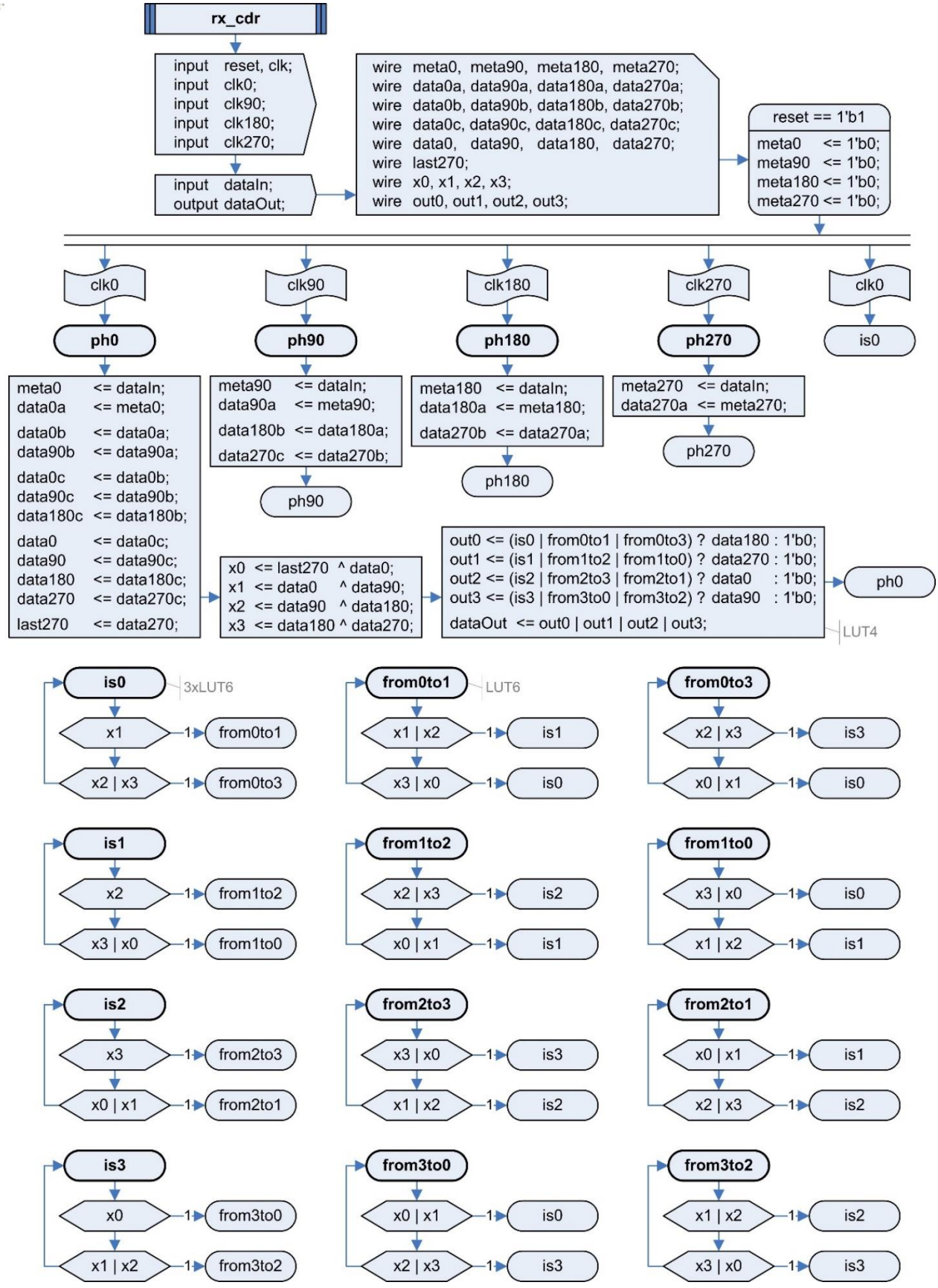


Figura 54: Diseño del circuito resincronizador.

Como se ha dicho antes, este circuito lo que hace es tomar 4 muestras de cada bit trasmitido cada ciclo de reloj. Para tomar las muestras se utilizan 4 señales de reloj desfasadas 0,90,180 y 270 grados respecto de la señal de reloj del transmisor. Esto es posible gracias a la placa ATLYS equipada con una FPGA Xilinx Spartan-6 LX45, la cual tiene integrado un módulo que se le suministra una señal de reloj y te saca de forma exacta las cuatro señales desfasadas 0,90,180 y 270 y grados. A continuación, se puede observar una representación gráfica de cómo funciona la toma de muestras.

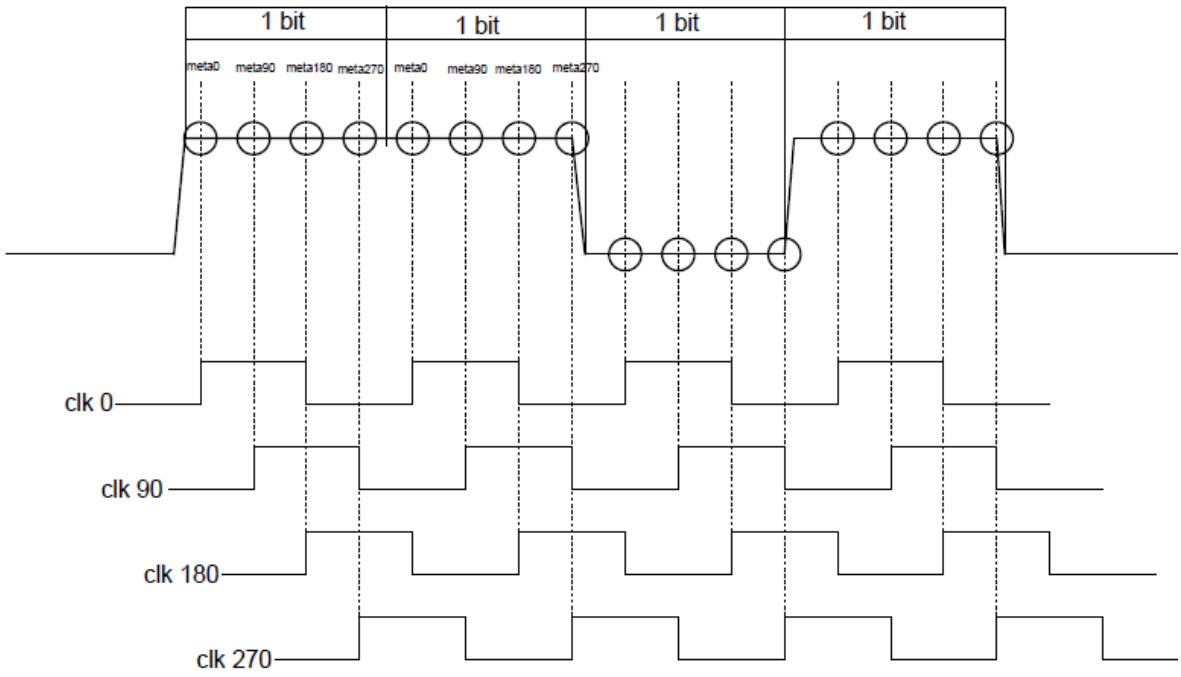


Figura 55: Toma de muestras con 4 señales desfasadas 90º entre ellas.

Una vez tomadas las muestras se registran en un DFF, es posible que el primer valor registrado presente problemas de metaestabilidad si justamente se está muestreando una transición.

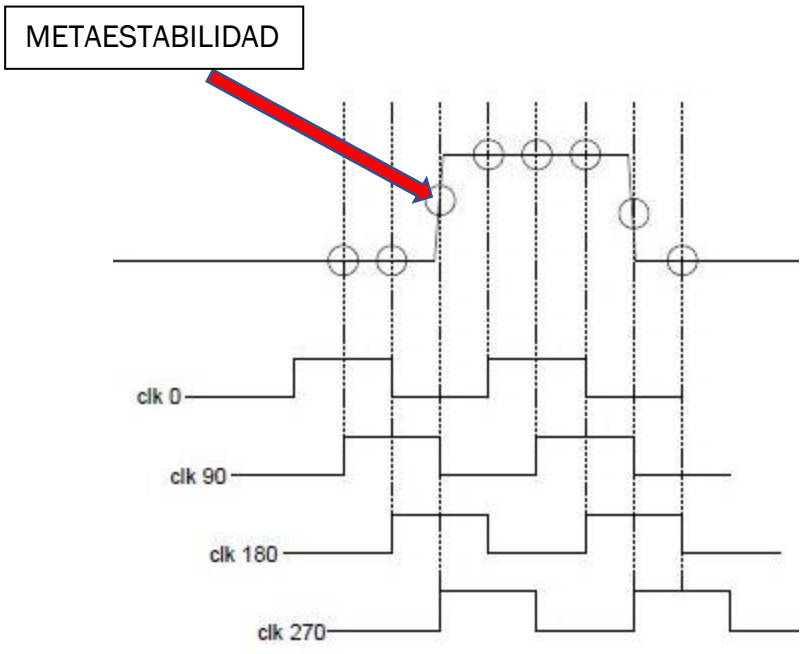


Figura 56: Metaestabilidad.

Por este motivo se vuelve a pasar por otro un biestable la señal, ya que pasados unos pocos nanosegundos la metaestabilidad se resuelve y se almacena en el segundo biestable un valor que puede ser correcto o incorrecto, más adelante se determinará si el valor registrado es válido o no.

```
meta270   <= dataIn;
data270a  <= meta270;
```

Figura 57: Ejemplo de pasar por dos biestables para resolver la metaestabilidad.

Pasados un cierto número de ciclos de reloj todas las muestras realizadas con distintas señales de reloj se terminan sincronizando con la señal “clk0”, para ello se van traspasando los datos de unos hilos a otros, lo muestro en la siguiente imagen.

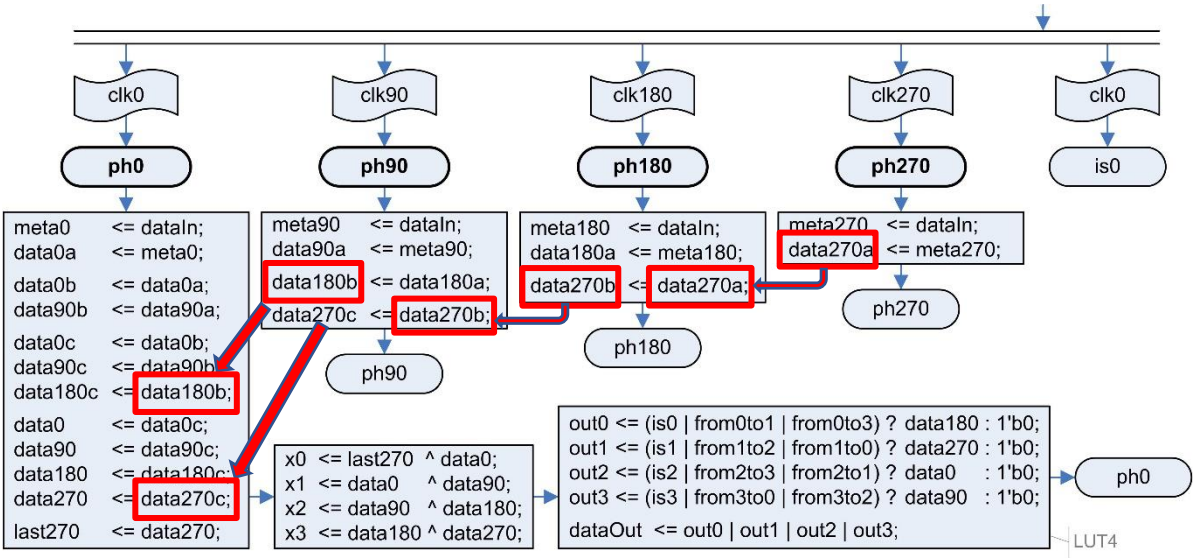


Figura 58:Traspaso de datos de unos hilos a otros.

Una vez se tienen todas las muestras sincronizadas con “clk0” lo que se hace es un XOR de las distintas muestras para detectar en que posición hay una transición.

```
x0 <= last270 ^ data0;
x1 <= data0   ^ data90;
x2 <= data90  ^ data180;
x3 <= data180 ^ data270;
```

Figura 59: XOR de las distintas lecturas.

La siguiente caja que continua después del XOR se encarga de asignar a la señal *“dataOut”* el valor más preciso de las mediciones tomadas.

```
out0 <= (is0 | from0to1 | from0to3) ? data180 : 1'b0;
out1 <= (is1 | from1to2 | from1to0) ? data270 : 1'b0;
out2 <= (is2 | from2to3 | from2to1) ? data0   : 1'b0;
out3 <= (is3 | from3to0 | from3to2) ? data90  : 1'b0;
dataOut <= out0 | out1 | out2 | out3;
```

Figura 60: Asignacion del valor más fiable a la señal "dataOut"

Para elegir cual es lectura más fiable se utiliza la siguiente máquina de estados que funciona como una pequeña memoria que recuerda donde se están produciendo las transiciones de los bits.

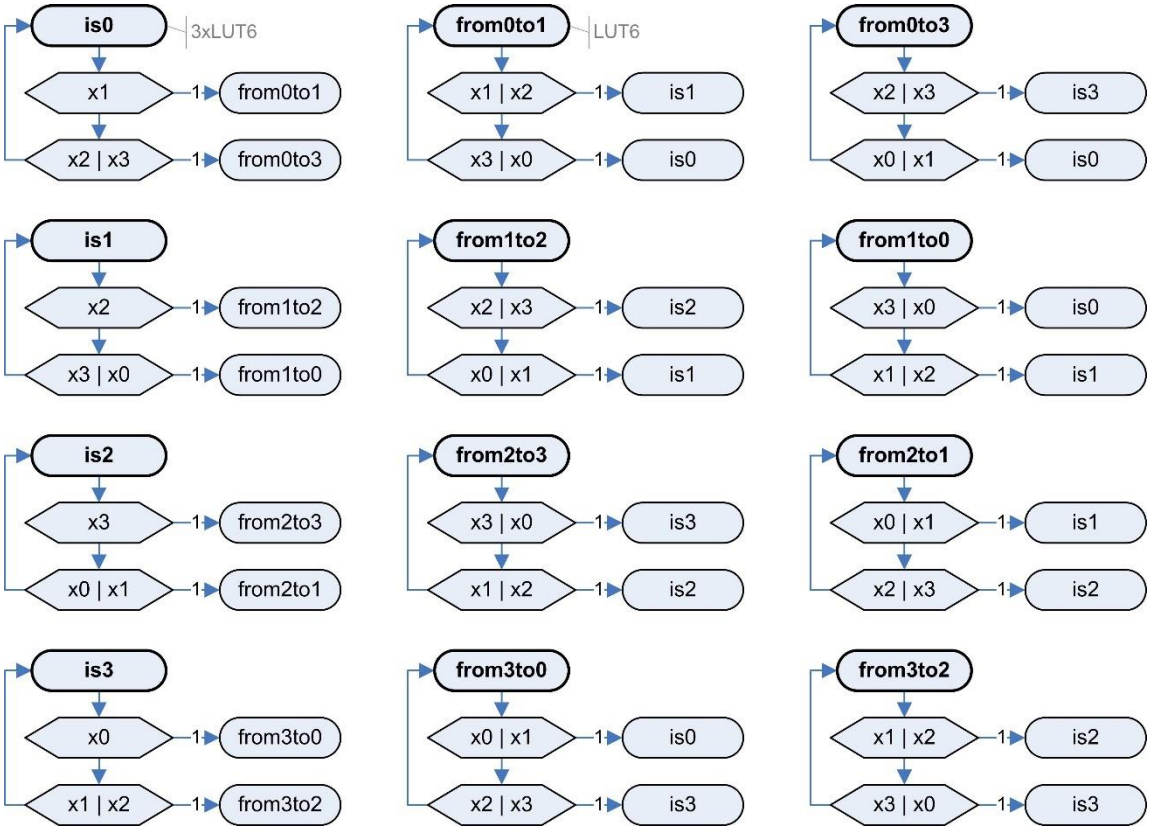


Figura 61: Maquina de estados que determina donde se producen las transiciones.

3.8 DESARROLLO DEL CIRCUITO RECEPTOR

El receptor es el último componente de lo que se considera estrictamente un canal de comunicaciones. El circuito del receptor se encarga de visualizar y chequear un frente de onda que entra secuencialmente. A continuación, se muestran unas imágenes que engloban todo el diseño, después se explicaran detalladamente cada uno de los aspectos de este:

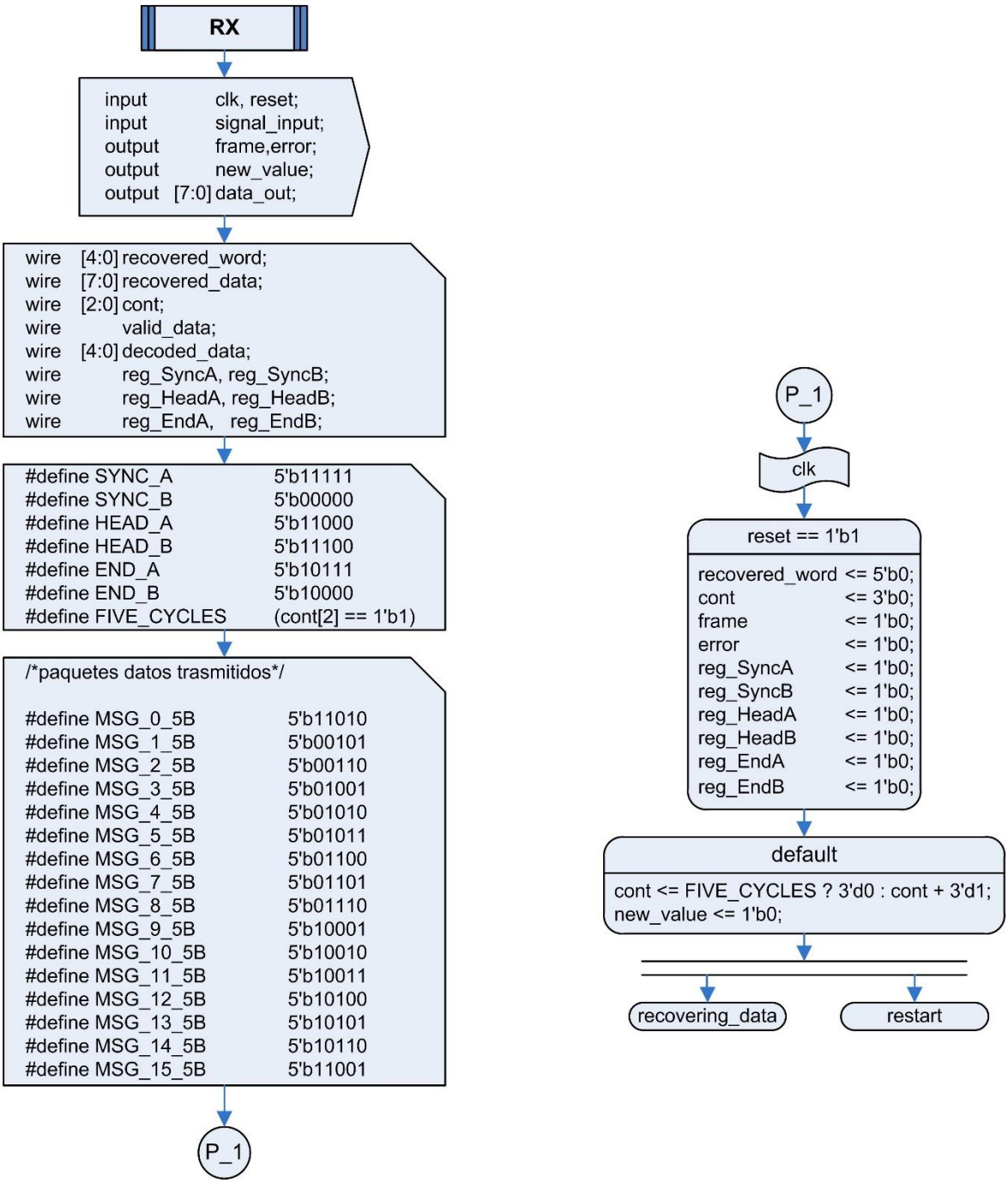


Figura 62: Pagina 1 diseño del receptor.

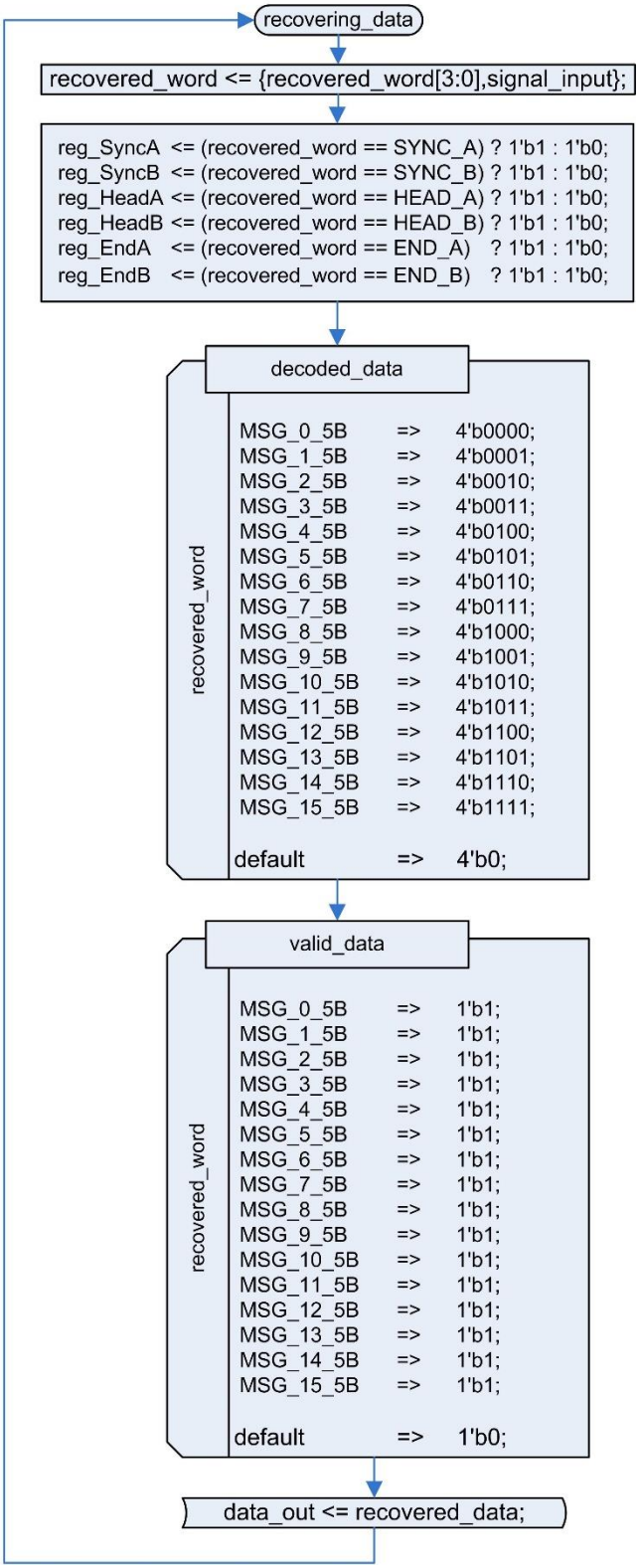


Figura 63: Pagina 2 diseño del receptor.

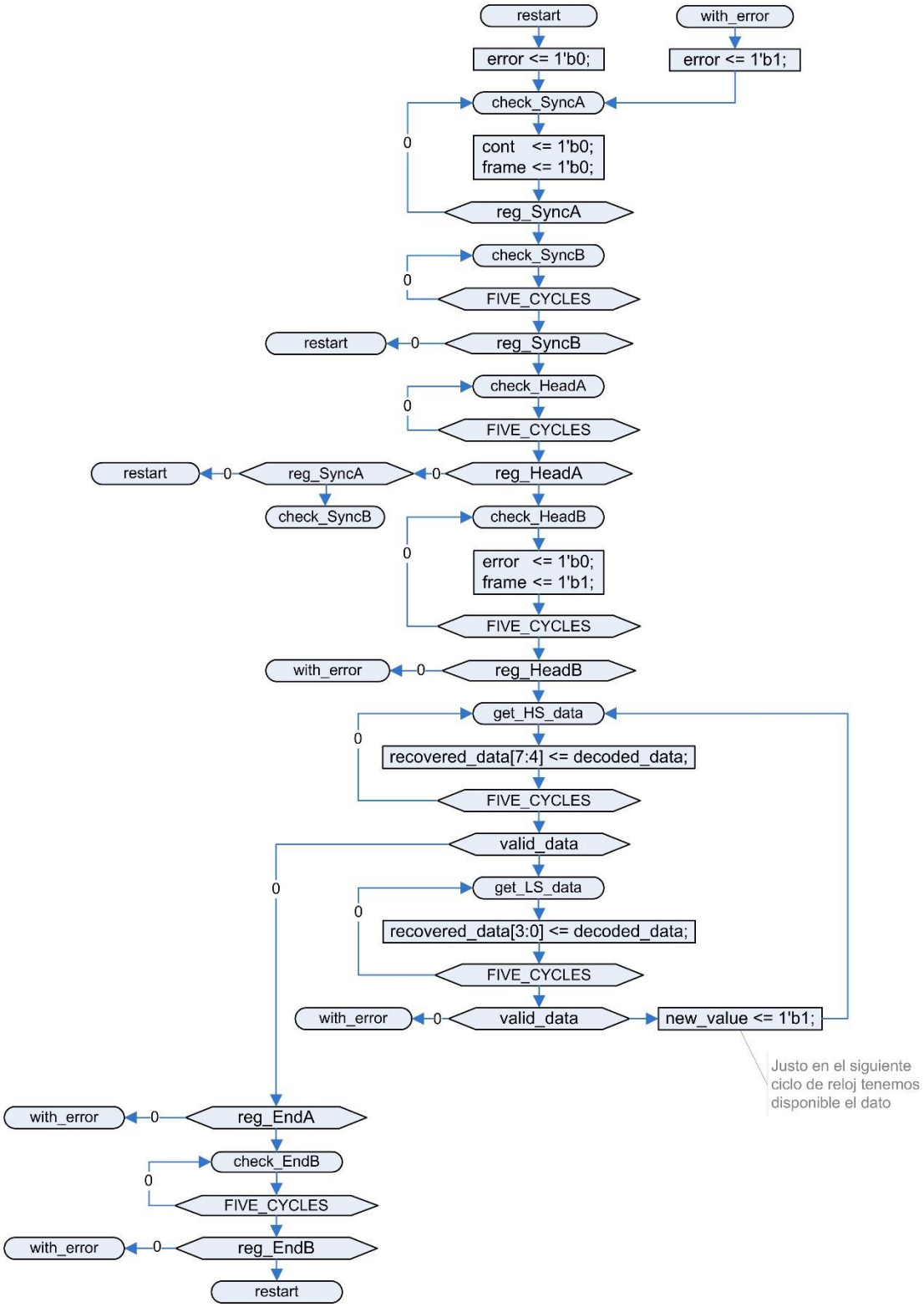


Figura 64: Pagina 3 diseño del receptor.

Para el funcionamiento del receptor se necesitan las siguientes señales de entrada y salida:

```
input   clk,reset;
input   signal_input;
output  frame,error,new_value;
output  [7:0] data_out;
```

Figura 65: Entradas y salidas del circuito receptor.

La señal de entrada "clk" se encarga de sincronizar todo el circuito y la señal *"signal_input"* introduce al circuito el frente de onda secuencial, es decir es por donde entran los datos.

Las señales de salida tienen los siguientes usos:

- ***"frame"*:** se utiliza para indicar cuando se está recibiendo un dato útil. Se activa cuando comienza un paquete de datos y se desactiva cuando finaliza la recepción de este o cuando se detecta una señal de error.
- ***"error":*** esta señal se utiliza para avisar de que se ha producido un error en la recepción de un paquete de datos.
- ***"new_value":*** esta señal se encarga de informar al siguiente circuito de que está disponible un nuevo paquete de datos sin errores.
- ***"data_out":*** con esta señal se muestran aquellos paquetes de ocho bits que ya han sido validados por el circuito receptor.

El circuito también cuenta con las siguientes señales internas que hacen posible su correcto funcionamiento:

```
wire   [4:0] recovered_word;
wire   [7:0] recovered_data;
wire   [2:0] cont;
wire   valid_data;
wire   [4:0]decoded_data;
wire   reg_SyncA,reg_SyncB,reg_HeadA;
wire   reg_HeadB,reg_EndA,reg_EndB;
```

Figura 66: Señales internas del circuito receptor.

- ***"recovered_word"*** es una señal de 5 bits que va guardando los datos que entran de forma secuencial.
- ***"recovered_data"*** se encarga de almacenar internamente paquetes de datos que ya han sido chequeados, para posteriormente cederlos a la señal de salida *"data_out"*.
- ***"cont"*** esta señal se utiliza como contador.

- ***"valid_data"*** esta señal se utiliza para validar si los paquetes de datos que llegan al receptor son correctos.
- ***"decoded_data"*** se utiliza para guardar internamente el valor decodificado (valor de 4 bits) procedente de los paquetes de 5 bits.
- ***"reg_SyncA", "reg_SyncB", "reg_HeadA", "reg_HeadB", "reg_EndA", "reg_EndB"*** se utilizan como señales de verificación, es decir son activadas o desactivadas si se registra o no el paquete de datos al que corresponda cada señal.

Para el diseño del receptor se utilizarán las mismas cajas de definiciones que en el diseño del transmisor:

```
#define SYNC_A          5'b11111
#define SYNC_B          5'b00000
#define HEAD_A          5'b11000
#define HEAD_B          5'b11100
#define END_A           5'b10111
#define END_B           5'b10000
#define FIVE_CYCLES     (cont[2] == 1'b1)
```

Figura 67: Caja de definición de las palabras de control.

```
/*paquetes datos trasmitidos*/

#define MSG_0_5B        5'b11010
#define MSG_1_5B        5'b00101
#define MSG_2_5B        5'b00110
#define MSG_3_5B        5'b01001
#define MSG_4_5B        5'b01010
#define MSG_5_5B        5'b01011
#define MSG_6_5B        5'b01100
#define MSG_7_5B        5'b01101
#define MSG_8_5B        5'b01110
#define MSG_9_5B        5'b10001
#define MSG_10_5B       5'b10010
#define MSG_11_5B       5'b10011
#define MSG_12_5B       5'b10100
#define MSG_13_5B       5'b10101
#define MSG_14_5B       5'b10110
#define MSG_15_5B       5'b11001
```

Figura 68: Caja de definición paquetes de datos.

Estas definiciones se usarán para que el código tenga un carácter más intuitivo, ya que se utilizarán las palabras en vez de las combinaciones de bits.

Una de las partes fundamentales del receptor se encuentra en la caja de ***default:***

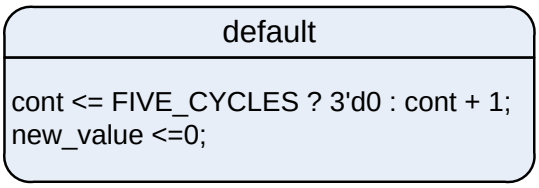


Figura 69: Caja de default diseño del receptor.

La operación **cont <= FIVE_CYCLES ? 3'd0 : cont + 1** es una forma compacta de programar que si el contador se corresponde con la definición de *FIVE_CYCLE* se pone a cero y si no cumple esa condición aumenta el contador en una unidad.

Una vez pasadas todas estas cajas el diseño se divide en dos hilos, el que comienza con el estado ***"recovering_data"*** (es el único estado del hilo por lo que se ejecuta durante todos los ciclos de reloj) que se encarga de recuperar la señal que entra al sistema de forma secuencial, para ello hemos utilizado la señal *"recovered_word"* y lo hemos programado de la siguiente forma:

```
recovered_word <= {recovered_word[3:0],signal_input};
```

Figura 70: Caja síncrona utilizada para recuperar la señal secuencial.

En cada ciclo de reloj la señal *"recovered_word"* cambia su bit menos significativo por el valor de *"signal_input"* y desplaza una posición hacia la izquierda los cuatro bits menos significativos. Al ser una caja síncrona hay que tener en cuenta que dichas variaciones serán visibles un ciclo de reloj más tarde. A continuación, se muestra un esquema del funcionamiento de esta caja:

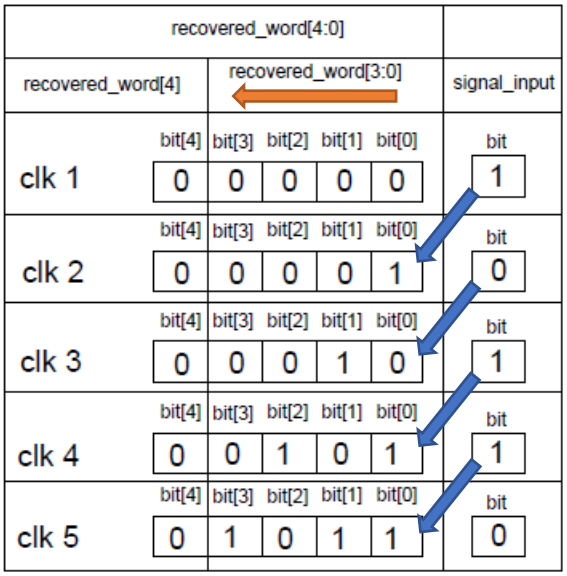


Figura 71: Esquema de cómo se van recuperando los bits.

Continuando en el mismo hilo, nos encontramos con la siguiente caja síncrona:

```
reg_SyncA  <= (recovered_word == SYNC_A) ? 1'b1 : 1'b0;
reg_SyncB  <= (recovered_word == SYNC_B) ? 1'b1 : 1'b0;
reg_HeadA <= (recovered_word == HEAD_A) ? 1'b1 : 1'b0;
reg_HeadB <= (recovered_word == HEAD_B) ? 1'b1 : 1'b0;
reg_EndA   <= (recovered_word == END_A)   ? 1'b1 : 1'b0;
reg_EndB   <= (recovered_word == END_B)   ? 1'b1 : 1'b0;
```

Figura 72: Caja síncrona encargada de chequear la señal "recovered_word".

En esta caja lo que hacemos es chequear durante todos los ciclos de reloj (el hilo solo tiene un estado) si la señal *"recovered_word"* se corresponde con alguno de los valores definidos, cuando se cumpla la condición del registro que corresponda, en este se almacena el valor 1 y lo muestra un ciclo de reloj más tarde, sino se cumple, se guarda un 0. Se utilizan todas estas señales como si fueran registros que almacenan y muestran al resto del circuito cuando se ha detectado un determinado bloque de bits.

Las siguientes operaciones que se realizan este hilo funcionan como dos *switch:*

- EL primero se encarga de asignar un valor u otro a la señal *"decoded_data"* en función del valor de la señal *"recovered_word".*

```
decoded_data
recovered_word
MSG_0_5B      =>   4'b0000;
MSG_1_5B      =>   4'b0001;
MSG_2_5B      =>   4'b0010;
MSG_3_5B      =>   4'b0011;
MSG_4_5B      =>   4'b0100;
MSG_5_5B      =>   4'b0101;
MSG_6_5B      =>   4'b0110;
MSG_7_5B      =>   4'b0111;
MSG_8_5B      =>   4'b1000;
MSG_9_5B      =>   4'b1001;
MSG_10_5B    =>   4'b1010;
MSG_11_5B    =>   4'b1011;
MSG_12_5B    =>   4'b1100;
MSG_13_5B    =>   4'b1101;
MSG_14_5B    =>   4'b1110;
MSG_15_5B    =>   4'b1111;

default          =>   4'b0;
```

Figura 73: Asignación de datos a la señal "decoded_data" en función de "recovered_word".

- El segundo se encarga de almacenar en la señal *"valid_data"* el valor 1 cuando la señal *"recovered_word"* se corresponde con uno mensajes codificados y el valor 0 cuando no se corresponde con ningún valor de

la tabla. Esta caja es una forma sencilla de chequear si los paquetes de datos (datos útiles) que se están recibiendo son válidos o no.

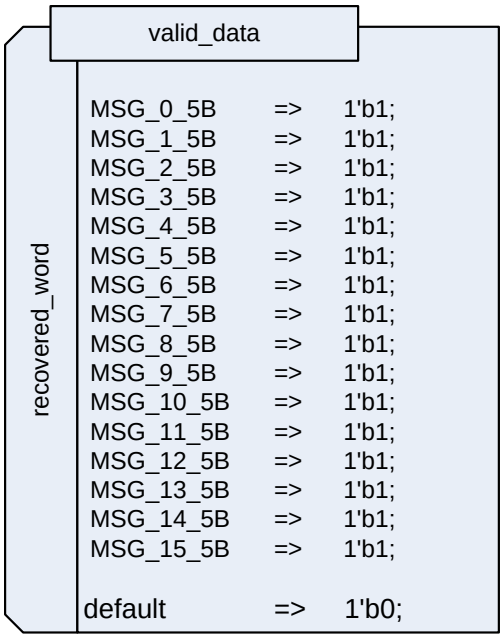


Figura 74: Caja de validación de mensajes.

La última operación que se realiza en este hilo es asíncrona, y es la encarga de mostrar al exterior del circuito paquetes de 8 bits decodificados y correctamente chequeados:

```
data_out <= recovered_data;
```

Figura 75:Caja asíncrona encargada de mostrar paquetes de datos.

El segundo hilo del diseño comienza con el estado ***"starting"***, su función es la de gestionar la información de los registros anteriormente descritos y función de estos se irá pasando de unos estados a otros. En el primer estado de este hilo se pone la señal *"error"* a cero y no se vuelve a pasar por el hasta que se reinicia el circuito, se hace esto para que cuando arranca el sistema dicha señal tenga un valor definido antes de empezar a procesar datos. Los siguientes estados se encargan de chequear que la activación de los registros del primer hilo coincida con la entrada de un mensaje.

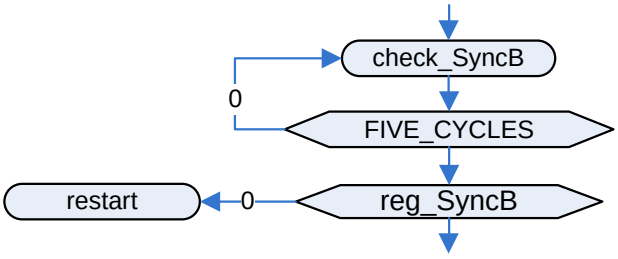


Figura 76: Ejemplo de chequeo de registros.

Una vez se llegue al estado **"check_HeadB"** la señal *"error"* tomara el valor 0 y la señal *"frame"* tomara el valor 1, a partir de este se comenzarán a recibir datos de hasta que se detecte la combinación de fin de mensaje. Para saber cuándo se está recibiendo un mensaje se utiliza la señal *"frame",* que como he dicho antes se activa en el estado *"check_HeadB"* y se desactiva cuando se detecta la terminación del mensaje. A mayores se ha programado que si se detecta un fallo cuando se están recibiendo mensajes se activa la señal de *"error"* y un ciclo de reloj más tarde se desactiva la señal *"frame".* Se han tomado estas consideraciones para que el circuito que vaya justo después del receptor pueda identificar si un mensaje es correcto o incorrecto, tan solo visualizando si la señal *"frame"* se ha desactivado con o sin errores. También se ha implementado la señal *"new_value"* que sirve para avisar al circuito que vaya después del receptor de la posición temporal exacta en la que estará disponible un nuevo paquete de datos (mensaje), es decir avisa del ciclo de reloj exacto en el que se visualiza el paquete.

Se puede observar en el diseño un estado que a primera vista podría pensarse que es omisibles:

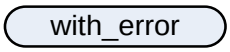


Figura 77: Estado intermedio del diseño RX.

A nivel de diseño este estado sí que podría ser omitido si no se tiene en cuanta su implementación física. Se ha decidido incluirle para reducir el número de entradas de las que depende el estado *"check_SyncA",* ya que todos los estados están programados para volver a este si se produce un error. Introduciendo el estado intermedio las posibilidades de llegar a "check_SyncA" desde cualquier zona del circuito se reduce considerablemente, consiguiendo reducir el número de entradas de las que dependería dicho estado. El concepto "número de entradas" condiciona la implementación física de los circuitos según el dispositivo utilizado, nosotros vamos a trabajar con el modelo de FPGA **Spartan-6 LX45** de la marca *Digilent*, cuyas celdas básicas esta formadas por memorias LUT 6, por lo que sería conveniente diseñar los circuitos para que ningún estado tenga más de seis entradas.

3.9 SIMULACIÓN CIRCUITO RECEPTOR

Para probar que el circuito receptor funciona correctamente, antes de implementarlo en físicamente se ha realizado un diseño de prueba al igual que con los circuitos descritos anteriormente. Dado que la entrada de datos al circuito receptor es de un solo bit, para hacer el test bench se ha tenido que declarar el valor de cada bit en cada ciclo de reloj. A continuación, se muestran unas imágenes de como seria la programación.

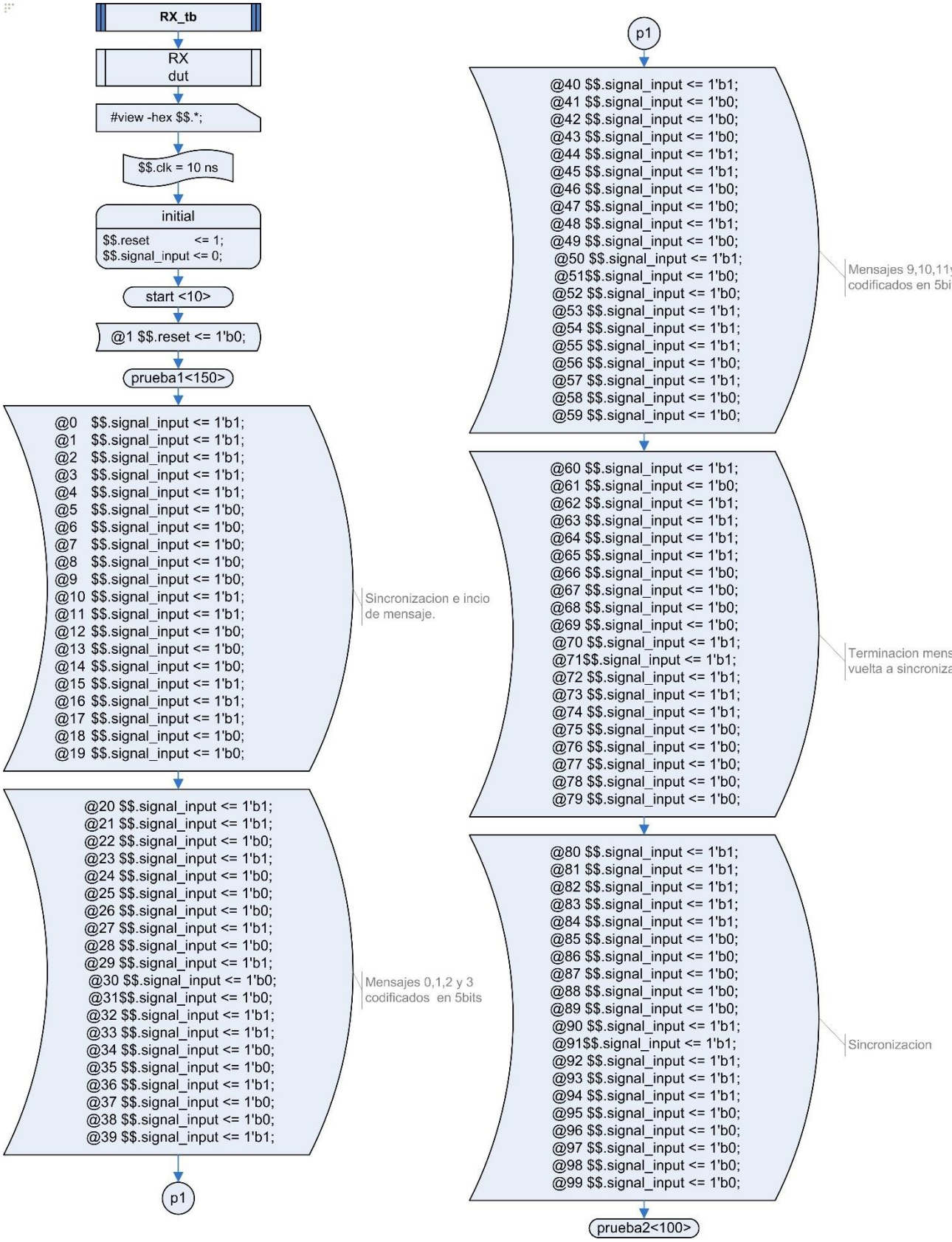


Figura 78: Pagina 1 y prueba 1 del diseño de test bench.

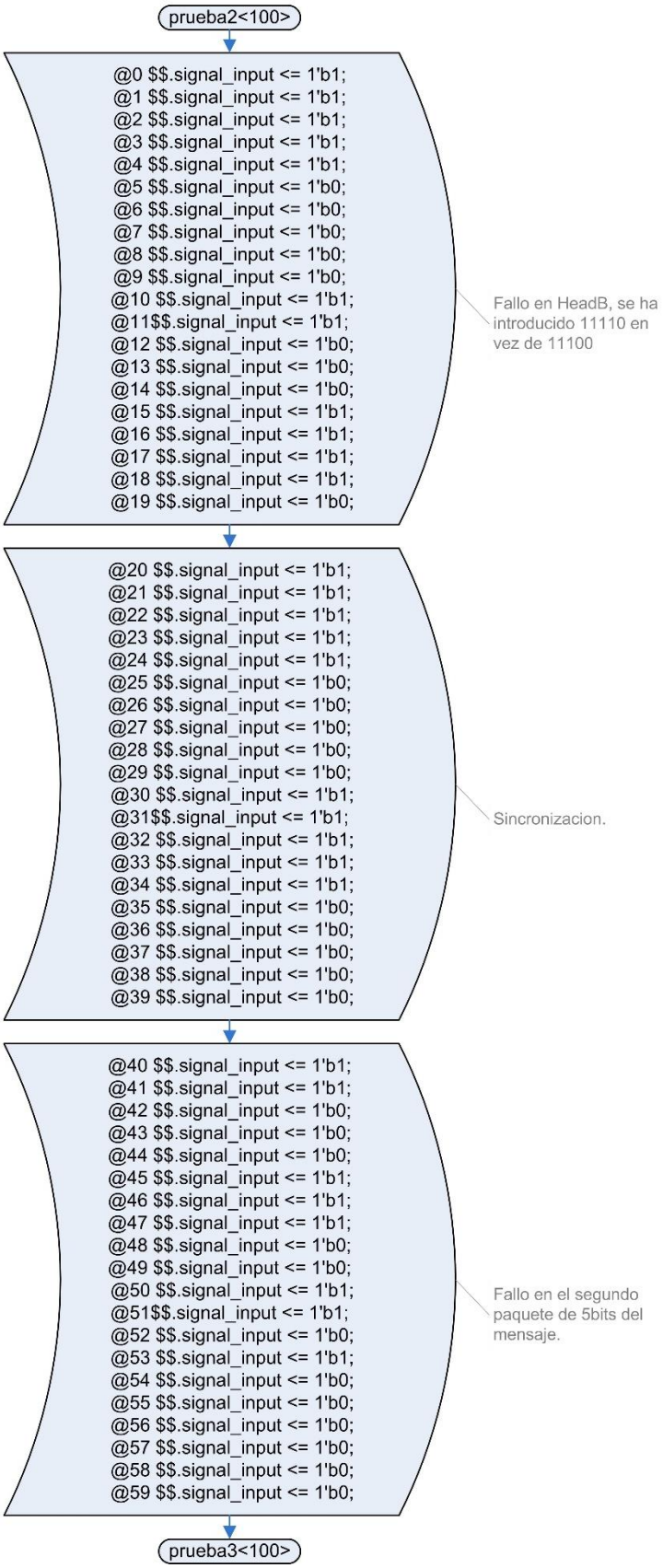


Figura 79: Pagina 2 y prueba 2 del diseño de test bench.

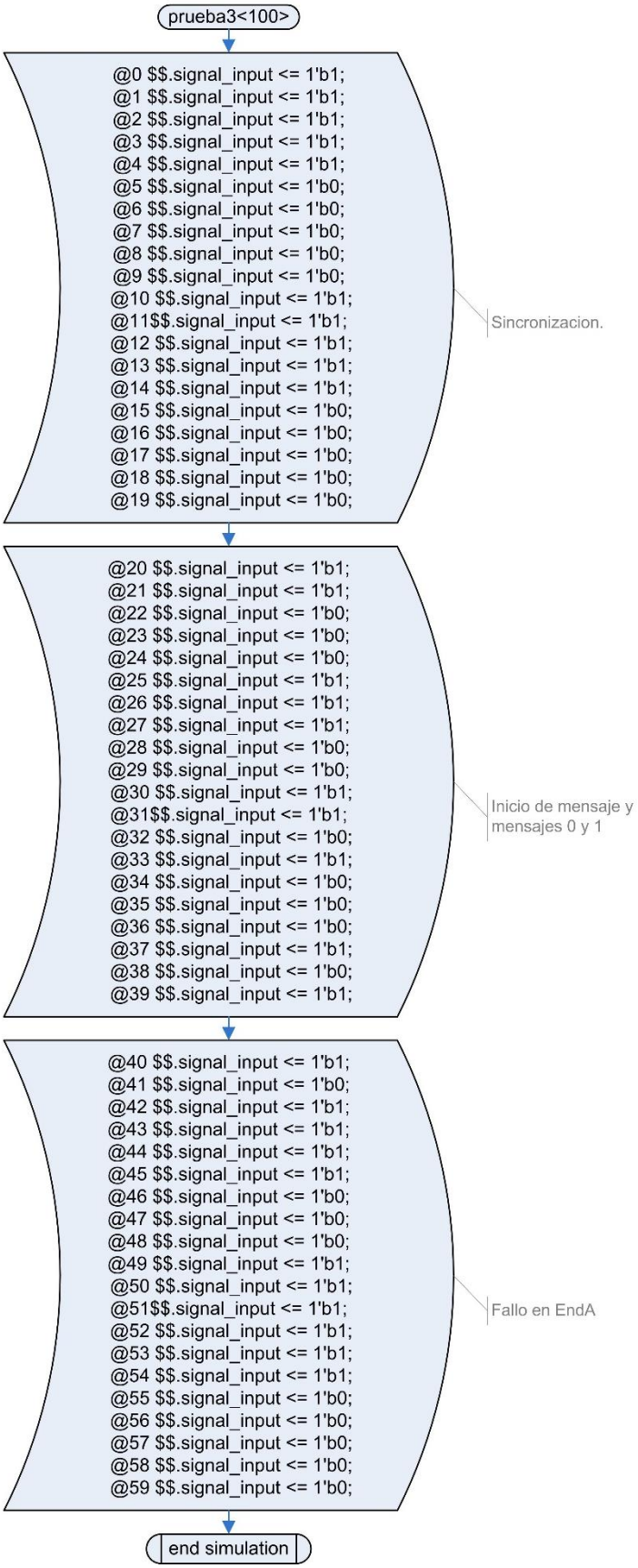


Figura 80: Pagina 3 y prueba 3 del diseño test bench.

Una vez se tiene el diseño de test bench se compila con la herramienta **ASM++ Compiler** para verificar que todo está correcto y que se puede visualizar la simulación con la herramienta *GTKWave.*

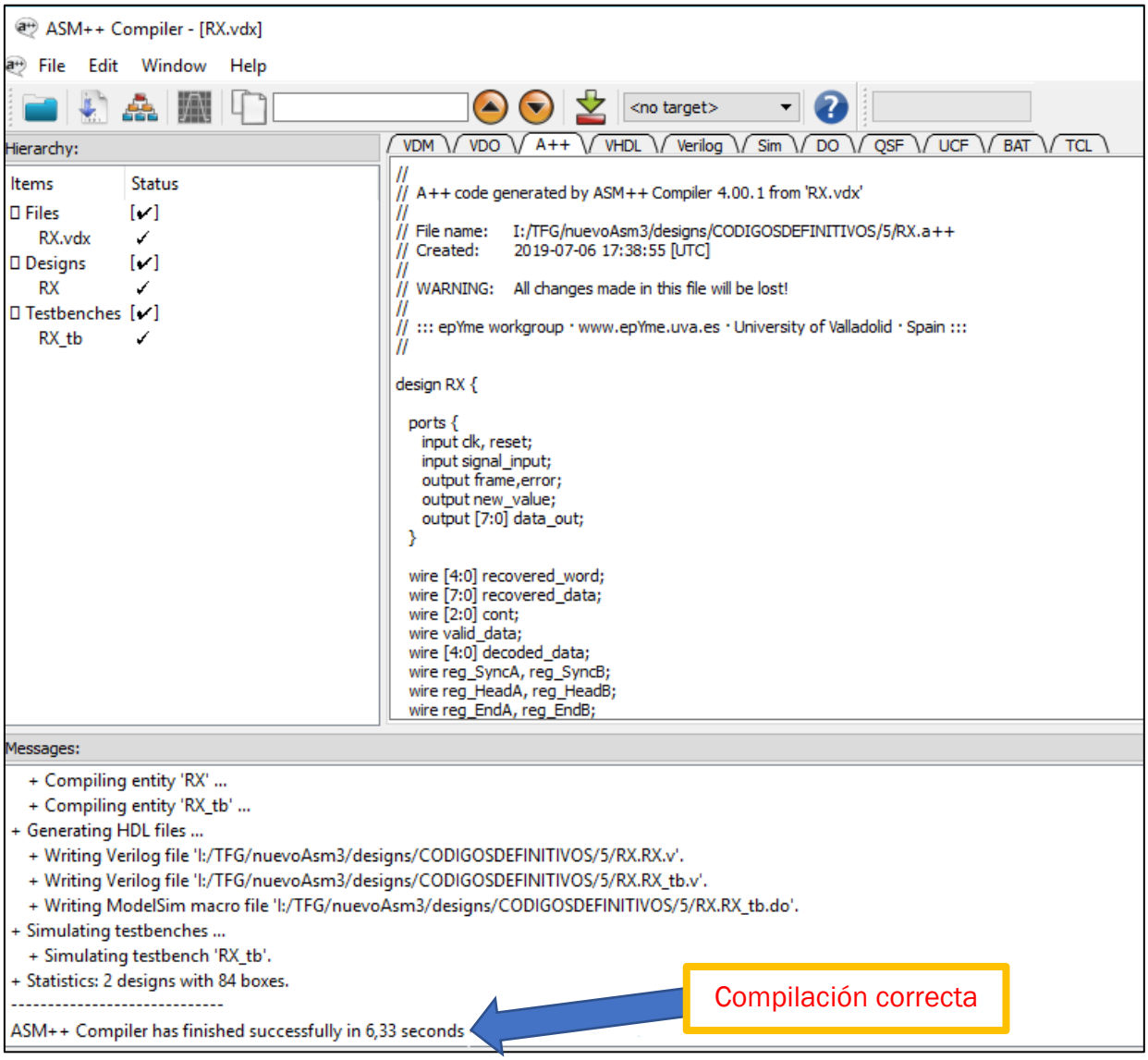


Figura 81: Compilación de los diseños RX y test bench.

Cuando todo está correctamente compilado, ya se puede acceder a la herramienta *GTKWave.* A continuación, se adjunta una imagen global de toda la simulación diseño de test bench, más adelante se mostrarán más detalladamente cada prueba realizada.

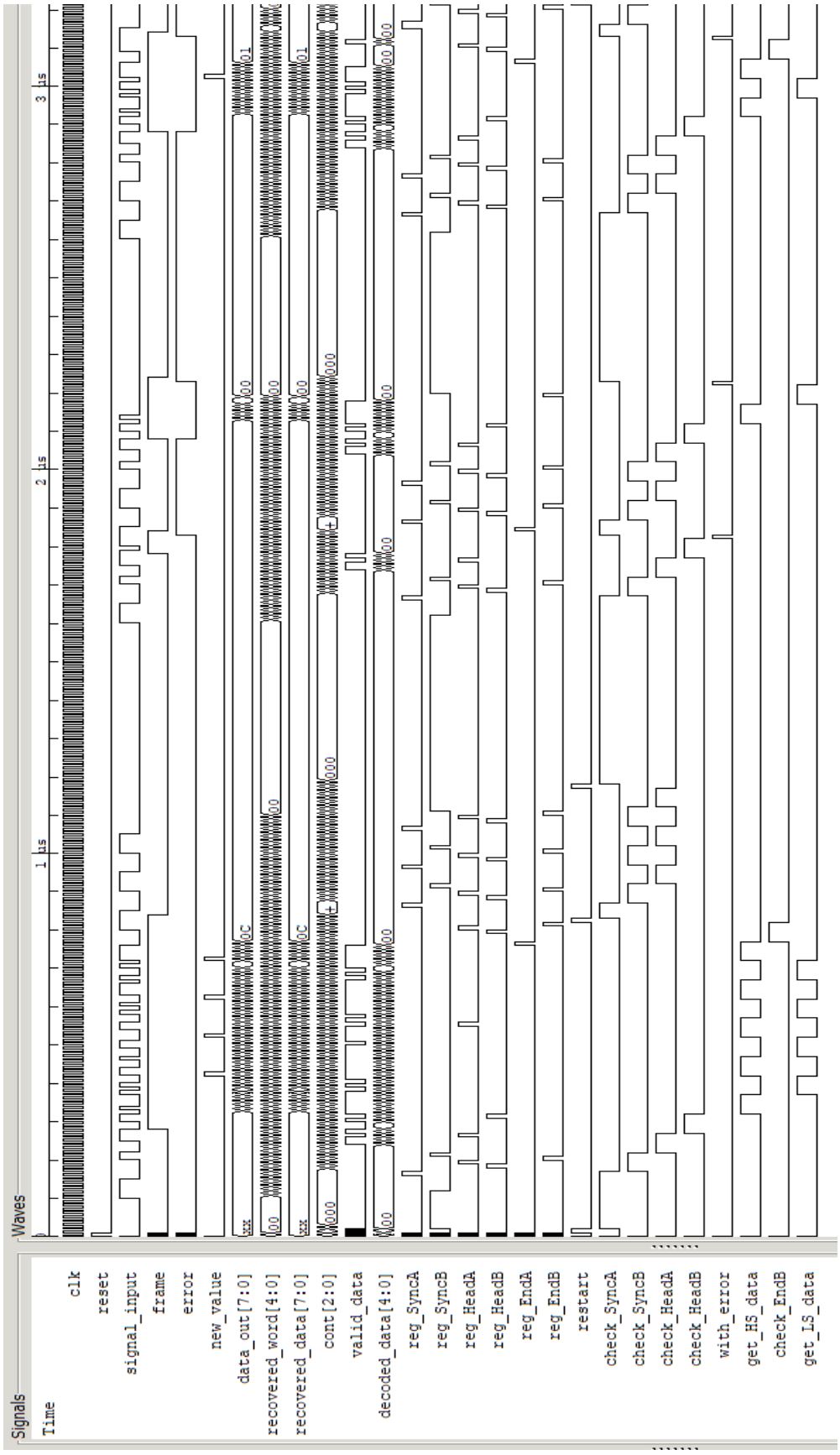


Figura 82: Simulación del diseño test bench.del circuito RX

En la primera página del test bench tenemos las cajas obligatorias para generar el fichero tipo RX y poder visualizar todas las señales, al igual que en otros diseños expuestos anteriormente. En la primera prueba que está programada se envía la señal de sincronización, los paquetes de inicio de mensaje (*HeadA y HeadB*), 4 paquetes de mensajes de 10bits, puestos en el siguiente orden:

- Mensaje 0
- Mensaje 1
- Mensaje 2
- Mensaje 3
- Mensaje 9
- Mensaje 10
- Mensaje 11
- Mensaje 12

Se finaliza la prueba con los paquetes de finalización de mensaje (*EndA y EndB*). A continuación, se puede observar el correcto funcionamiento de lo descrito. Se muestra en dos imágenes para poder visualizarlo con más detalle.

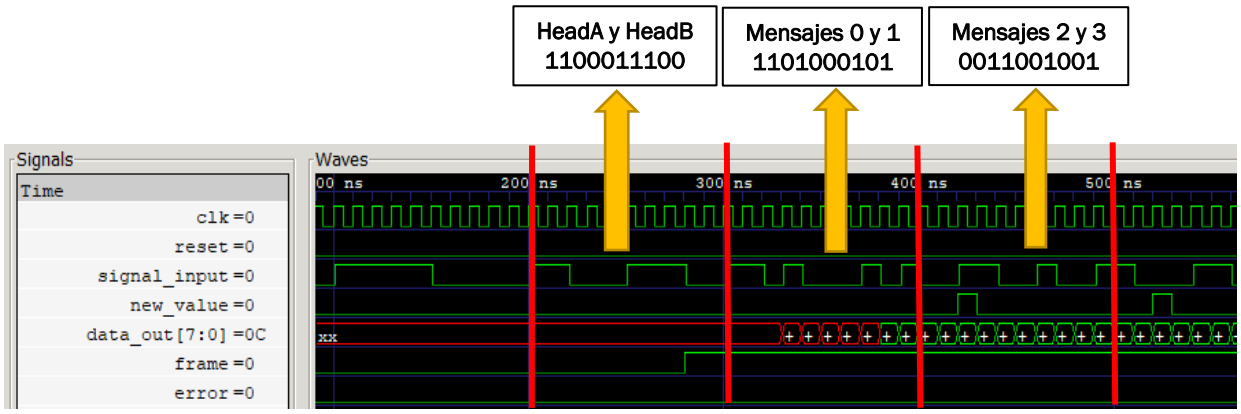


Figura 83: Simulación prueba 1: mensajes 0,1,2,3 y 9.

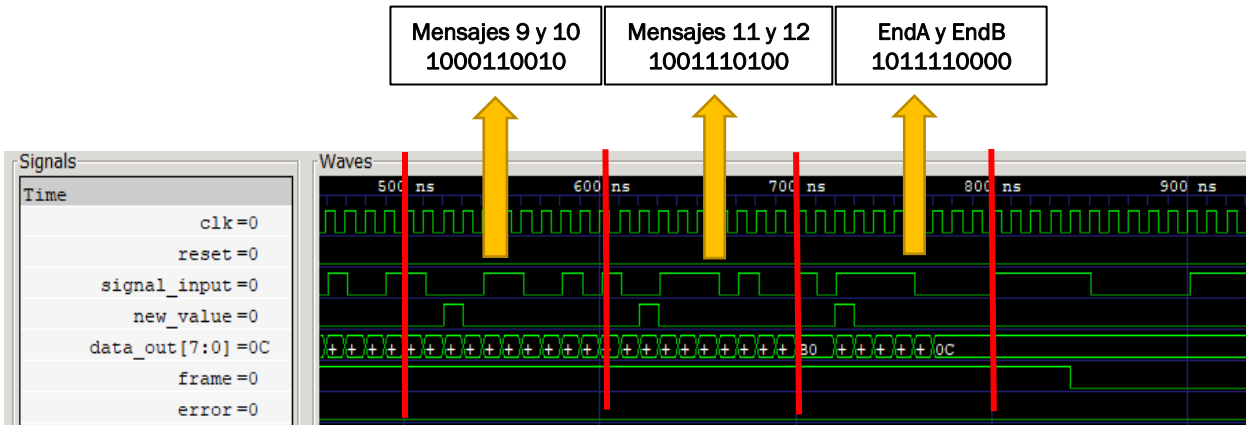


Figura 84: Simulación prueba 1: mensaje 9,10,11 y 12.

Ampliando un poco más las señales que se muestran con la herramienta GTKWave y mostando la señal *"data_out"* en formato se puede ver que justamente cuando se activa la señal de *"new_value"* está disponible el paquete de 8 bits que corresponda. Según se ha diseñado este circuito el valor de los 8bits del paquete de datos solo estará visible durante el ciclo de reloj en que esta activa la señal *"new_value"*.

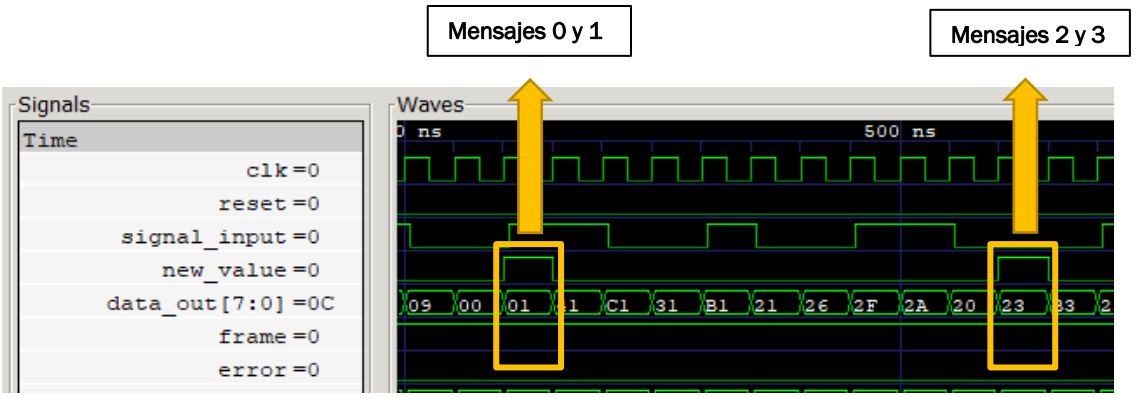


Figura 85: Visualización de los mensajes cuando esta activa "new_value".

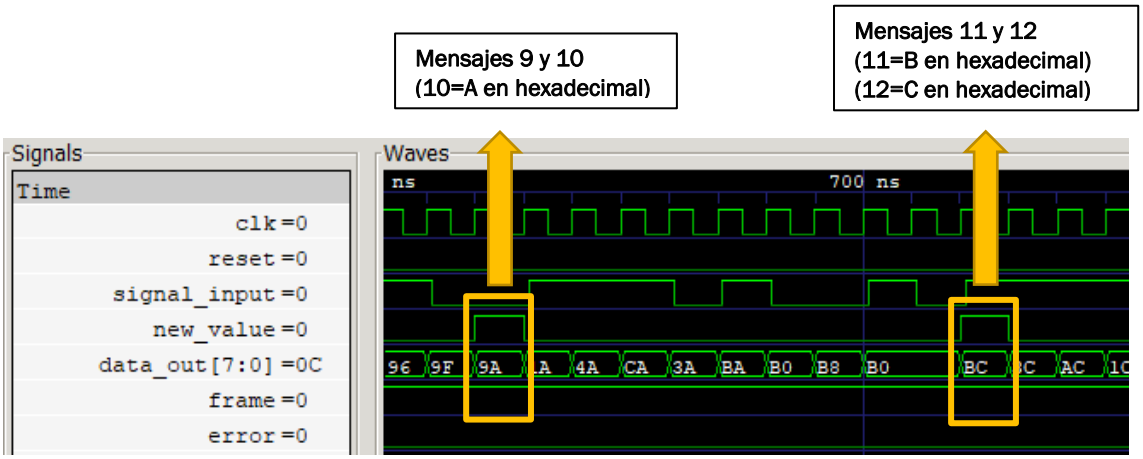


Figura 86: Visualización de los mensajes cuando esta activa "new_value".

En la segunda prueba se ha simulado un fallo en los paquetes de inicio de mensaje, exactamente en la palabra *HeadB*. Después se ha vuelto a transmitir sincronización y se ha simulado un fallo en medio de un paquete de datos, exactamente en el segundo bloque de 5 bits del paquete de datos.

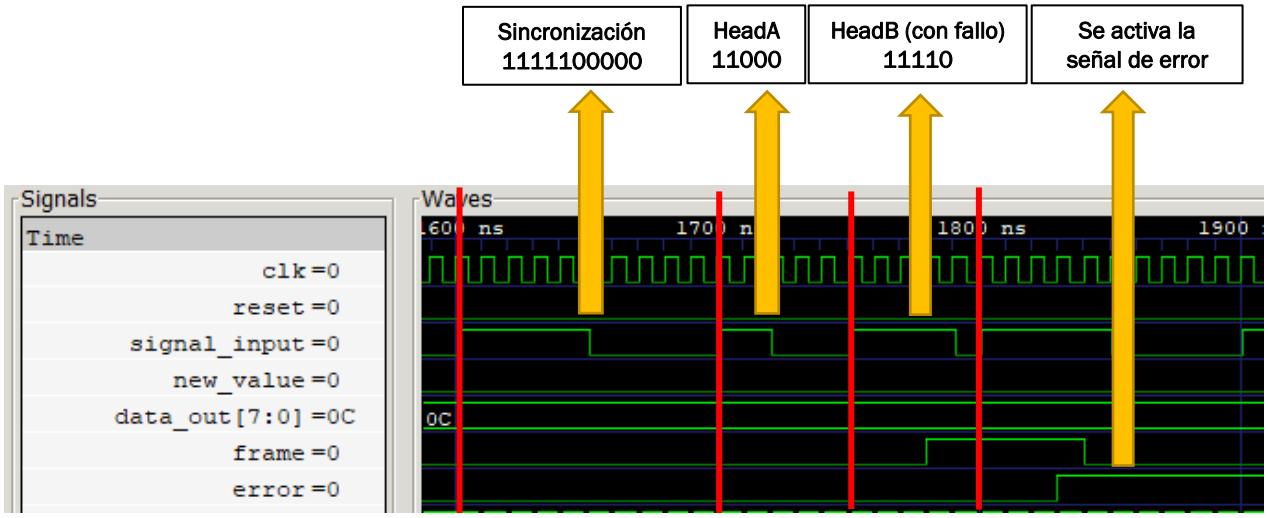


Figura 87: Simulación de fallo en el paquete de bits HeadB.

Se puede observar cómo se activa la señal de *"frame"* cuando comienza a llegar un mensaje y como se activa la señal de error en cuanto se detecta el fallo y un ciclo de reloj más tarde la señal de *"frame"* se desactiva. Por lo cual este mensaje sería despreciado por el circuito que sucede al RX al ver que la señal *"frame"* se ha desactivado con errores. A continuación, se muestra otro error, pero esta vez en el medio del paquete de datos, como se puede ver en la simulación, se activa la señal de *"error"* y se desactiva la señal de *"frame"*, invalidándose dicho mensaje. Como los dos errores se han simulado uno seguido de otro sin reiniciar el circuito se puede observar como la señal de *"error"* continua activa hasta que comienza un nuevo paquete de datos.

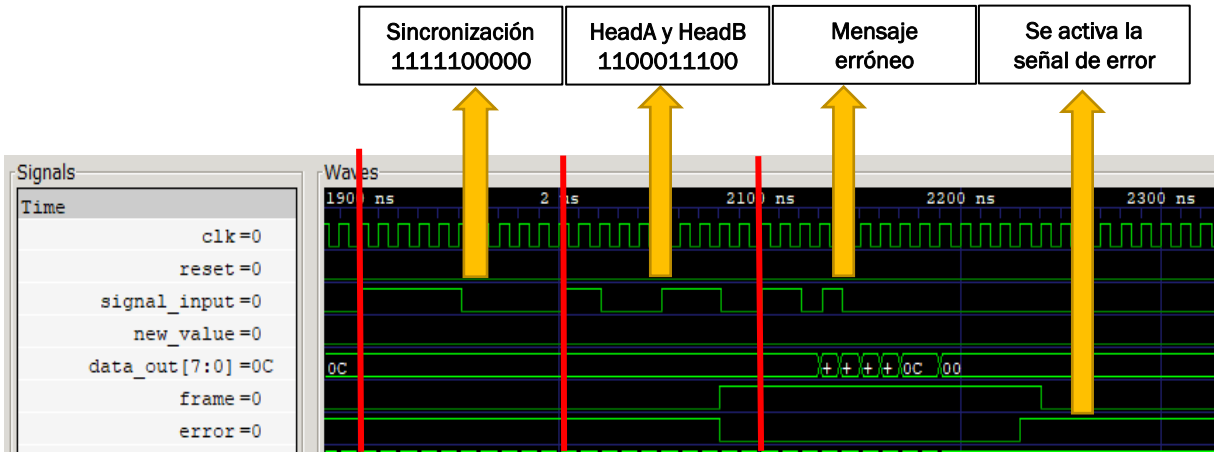


Figura 88: Simulación de error en el medio de un paquete de datos.

Para comprobar que todas las zonas del circuito funcionan correctamente, se ha realizado una tercera prueba en la que se ha provocado un error en los paquetes que indican terminación de mensaje, exactamente en en el paquete *EndA*. En la simulación se puede observar cómo se activa la señal *"new_value"* cuando se tiene el paquete de datos disponible, pero como al terminar el mensaje falla, se activa la señal de *"error"* y se desactiva la señal *"frame"*.

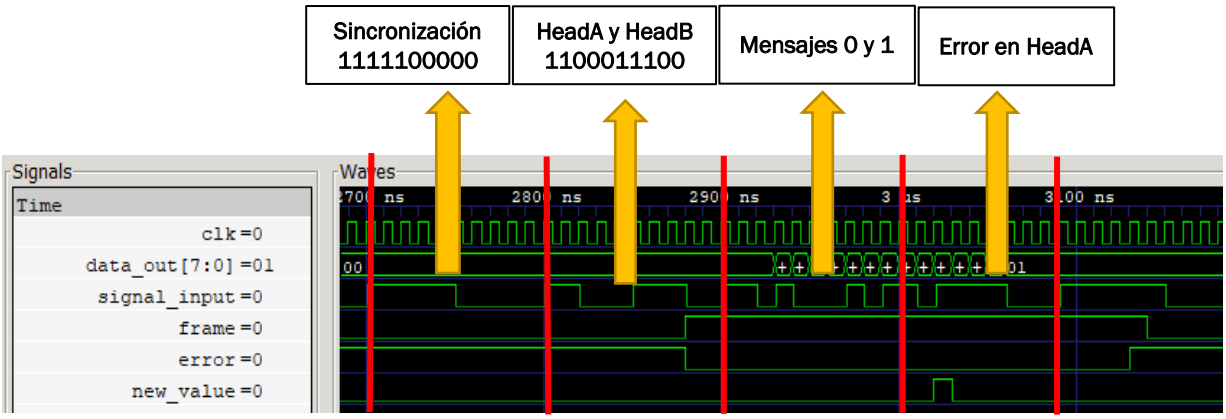


Figura 89: Simulación fallo en el paquete de bits HeadA.

4 PRUEBAS FÍSICAS

Una vez se han cargado todos los módulos en el diseño TxRxAtlys y se han asignado los valores de todas las entradas, ya se puede compilar el diseño, obteniéndose los ficheros de los diferentes lenguajes de programación y como se mencionó en el apartado 3.3 de este documento, los tres nuevos tipos de ficheros, como se puede ver en la siguiente imagen.

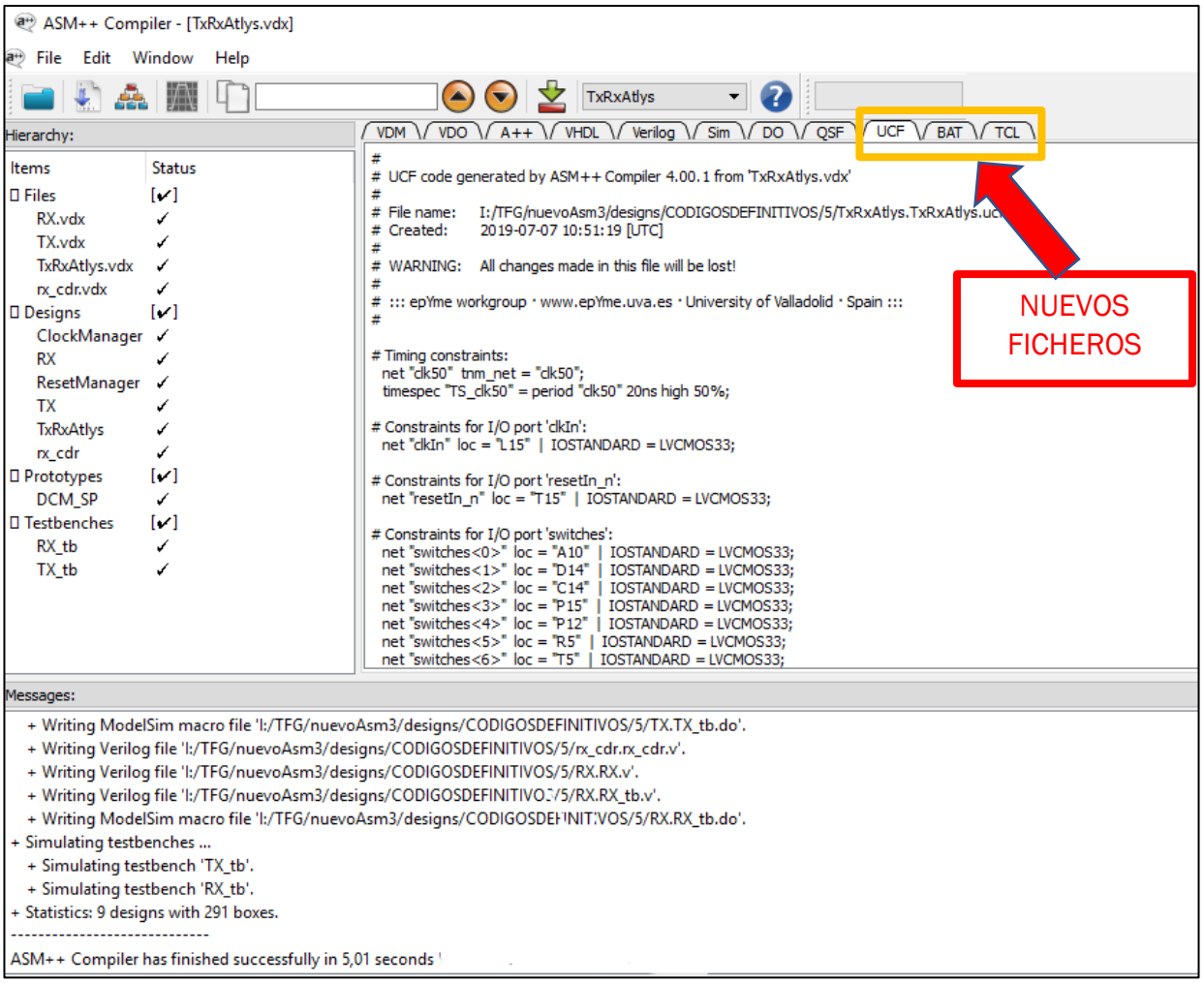


Figura 90: Herramienta de compilación ASM++ compiler.

Si todo ha funcionado correctamente se habrán creado de forma automática en el mismo directorio en el que se encuentra el fichero del diseño, todos los ficheros necesarios para el proceso de síntesis.

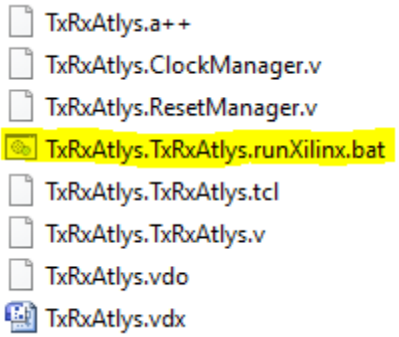


Figura 91: Ficheros generados tras la compilación del diseño TxRxAtlys.

Una herramienta del *ASM compiler* junto con el entorno de trabajo de Xilinx, realiza toda la síntesis de los circuitos. Simplemente basta con hacer doble clic en el fichero **[.bat]** y aparecerá una pantalla negra tipo MS2 como la que se adjunta continuación.

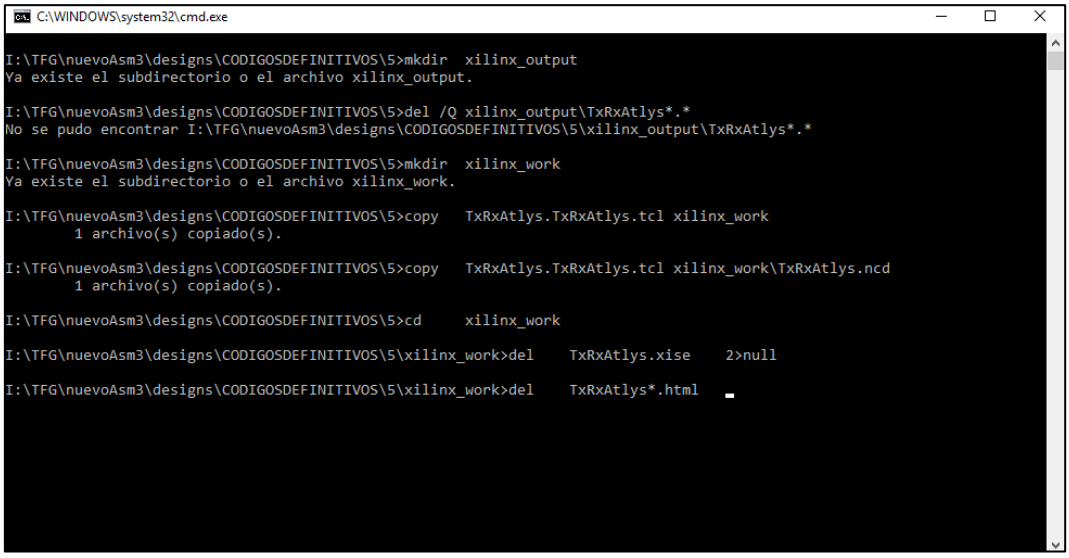


Figura 92: Pantalla generación del archivo de síntesis.

Dicha acción puede tardar unos minutos hasta que se procesen todos los datos. Una vez termine se habrá creado una carpeta llamada *xilinx_output* en el mismo directorio que se estaba trabajando y nos aparecerá el siguiente entorno de trabajo de *Xilinx* en el que podremos ver especificaciones de la implementación de los circuitos en la FPGA. A continuación, se muestra una imagen de dicho entorno, en la que se puede ver que el tiempo crítico del circuito.

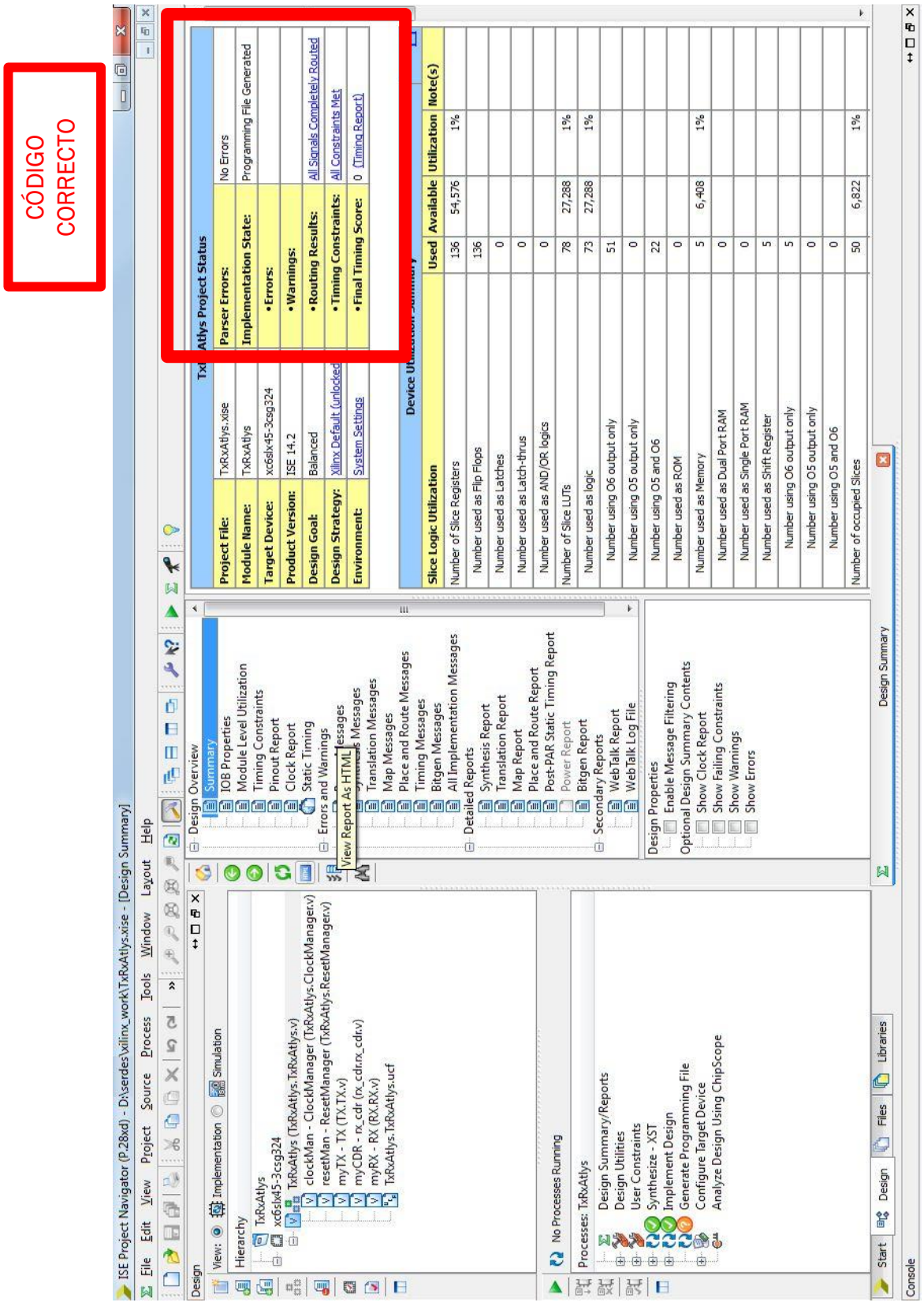


Figura 93: Entorno de programación de Xilinx.

En interior de esta carpeta se habrá creado un fichero **[.bit]** que será el que nos pida la herramienta de síntesis para poder cargar el trabajo realizado dentro de la FPGA. A continuación se adjunta una imagen del entorno en el que se utiliza el fichero **[.bit]** para ser cargado en la FPGA.

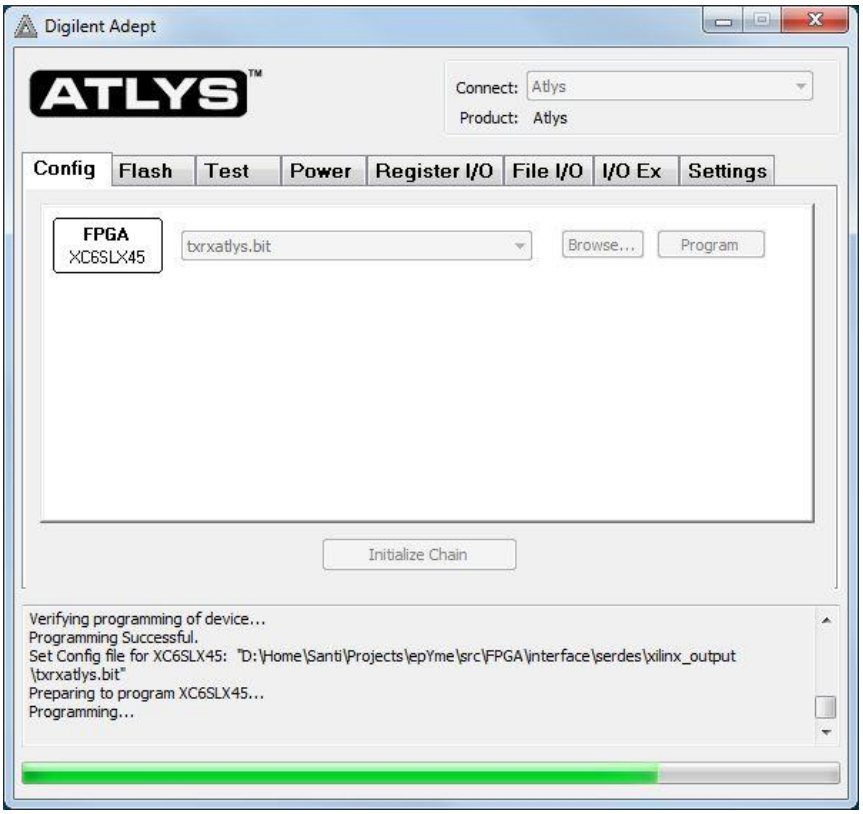


Figura 94: Entorno de programación de la FPGA.

En cuanto termina la barra de proceso, la plataforma estar cargada con los códigos y funcionando correctamente si todo ha ido bien. En nuestro caso comenzamos usando una señal de reloj de 250 ns (4 Mbps) ya que el analizador lógico del que disponíamos en el departamento podía muestrear hasta *24 MSample/sec* por lo que si de primeras cargábamos el circuito funcionando a 50 Mbps no podríamos haber visualizado si funcionaba o no correctamente. Las primeras pruebas realizadas a 4 Mbps fueron correctas, a medida que se cambiaban los *switches* se puede ver como se encienden o se

apagan los leds, además con el analizador lógico verificamos dicha transmisión como se puede ver a continuación.

Figura 95: Tranmision de datos 4Mbps.

Se realizo una segunda prueba a la frecuencia máxima de muestreo del analizador lógico, es decir a 24 Mbps, obteniendo otra vez un correcto funcionamiento del sistema.

Finalmente se probó el circuito funcionando a la velocidad de transmisión que nos habíamos marcado como objetivo, es decir a 50Mbps, obteniéndose resultados satisfactorios. Hay que mencionar que por contratiempos no se ha podido utilizar un cable de fibra óptica, por lo que se ha utilizado uno de cobre para hacer pruebas. A través del conector P-mod de la plataforma *Atlys* se han sacado diferentes señales de los circuitos para poder visualizarlas con el analizador. A continuación, se muestra una imagen del montaje funcionando a 50Mbps, en el que según la entrada de los *switches* se encienden unos leds u otros.

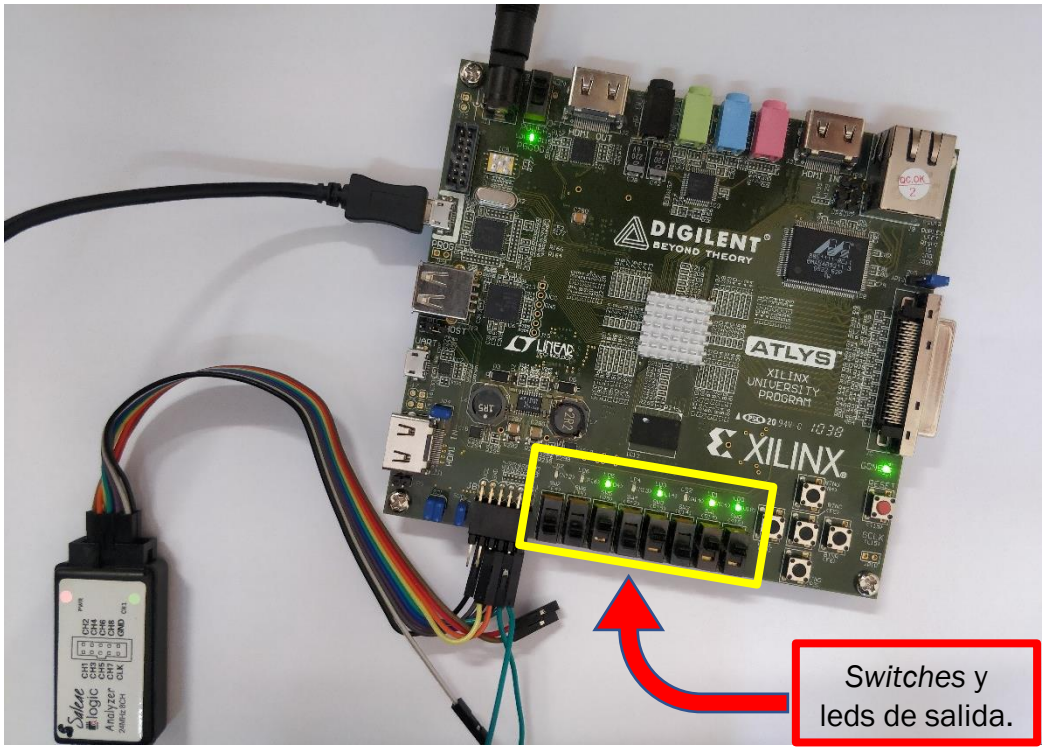


Figura 96: Plataforma Atlys con los códigos cargados, funcionando a 50 Mbps.

5 CONCLUSIONES

5.1 RESULTADOS DEL PROYECTO

Tras probar a conciencia las distintas combinaciones de mensajes que se pueden enviar a través de los *switches* de la plataforma de trabajo *Atlys*, se puede afirmar que el enlace de comunicaciones funciona a una velocidad de transmisión de 50 Mbps.

Desde el punto de vista docente, este proyecto ha servido para aprender numerosos conceptos del entorno de las telecomunicaciones y tomar partido de ellos para implementarlos en los diseños. Además de aprender una nueva metodología de diseño de circuitos mediante diagramas ASM++ y demostrar que son una potente herramienta a la hora de diseñar circuitos con lenguajes de descripción de *Hardware*.

Todo el trabajo de este proyecto será reutilizable para líneas futuras de investigación.

5.2 FUTURAS LÍNEAS DE INVESTIGACIÓN

La primera mejora que continuará a este proyecto será sustituir el cable de pruebas de cobre por el de fibra óptica, para ello se recomienda usar los siguientes componentes:

- **Transmisor (Tx)** fabricante: Broadcom Limited. Velocidad de transmisión 50Mbps. Referencia fabricante: AFBR-2624Z.
- **Receptor (Rx)** fabricante: Broadcom Limited. Velocidad recuperación de datos 50Mbps. Referencia fabricante: AFBR-1624Z.
- **Latiguillo (POF)** fabricante Broadcom Limited. Referencia fabricante: HFBR-RLS001Z.

En el **anexo 1** se adjuntan las especificaciones técnicas de estos componentes.

Continuando con las líneas futuras, lo segundo que se podría mejorar del enlace de comunicaciones es introducir un código que se encargue de muestrear cuantos errores se producen por unidad de tiempo, siendo esto un factor de vital importancia debido a la aplicación que tendrá el enlace de comunicaciones como componente esencial para que se realicen las conmutaciones necesarias del convertidor distribuido.

La tercera mejora podría ser la implementación de un código de verificación por redundancia cíclica (CRC) que se encargara de verificar que los mensajes que se enviaron por el transmisor son los mismos que están llegando al receptor, es decir una forma de comprobar que los mensajes no han sufrido ninguna alteración durante la transmisión.

La siguiente mejora antes de lanzarse hacia el siguiente nivel del proyecto sería intentar transmitir a más velocidad los datos, lo que conllevaría optimizar aún más los códigos.

El siguiente nivel después del enlace de comunicaciones seria diseñar los circuitos que gestionen los paquetes de datos que se deben enviar en cada momento en función de las necesidades de la instalación, es decir diseñar una CPU que actúe como un cerebro central y luego en cada módulo independiente habrá que pensar en implementar los correspondientes controladores de menor tamaño que se encarguen de gestionar la información y actuar sobre el módulo de la forma que proceda.

6 BIBLIOGRAFÍA

A lo largo del trabajo de fin de grado se han consultado los documentos y páginas web.

- [1] DEEPER research group - Renewable Energies, Power Electronics and Electronic Design (2002-2018). University of Valladolid. Spain. //Web: www.deeper.uva.es
- [2] Yongheng Yang, Katherine A. Kim,Frede Blaabjerg, Ariya Sangwongwanich(2019). Andvances in Grid-Connected Photovoltaic Power Conversion Systems. Woodhead Publishing Series in Energy
- [3] Jonathan Burdalo Gil. (2009). Proyecto de fin de carrera: Test, simulación y realización de un entorno de desarrollo para un DSP integrado en FPGA realizado mediante diagramas ASM++. Universidad de Valladolid. España.
- [4] David Sanz Cabreros. (2017). Trabajo de fin de grado: Desarrollo de un simulador de circuitos digitales integrado en un compilador de diagramas ASM. Universidad de Valladolid. Españana.
- [5] Guide To Fiber Optics & Premises Cabling (2014). The Fiber Optic Association //Web: www.thefoa.org
- [6] Asis Rodrigez (10 Junio 2012). Nombre articulo: Fibra Optica, que es y cómo funciona. //Web: www.fibraopticahoy.com
- [7] Digi-Key Electronics (2019) //Web: www.digikey.es
- [8] DIGILENT A National Instruments Company // Web: https://store.digilentinc.com/
- [9] P. Teehan, M. Greenstreet, G. Lemieux (October 2007). A Survey and Taxonomy of GALS Design Styles
- [10] S. Johnson, S. Scott (1995). A Supercomputer System Interconnect and Scalable IOS