ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

TRABAJO FIN DE GRADO

Evaluación de un sistema de adquisición de datos para uso en espacio basado en la tecnología Zynq UltraScale+ MPSoC

Autor:

Cristina Pérez Lemus

Director:

Rafael Toledo Moreo

Codirectores:

David Pérez Lizán

Jaime Gómez Saenz de Tejada

Abstract

The appearance of the CubeSats has revolutionized space exploration. In recent years, its popularity has increased due to its low cost and time of development, which has made easier the access to space to other entities than the traditional ones. These small satellites, have caused an increase in the possibilities of research and development of future applications.

This project evaluates a possible application for its future use in a CubeSat. In particular, it tests the development of an image acquisition system, made with the ZynqUltraScale + architecture. For this purpose, a generator and transmitter of data at a specific rate is shown with a high level of detail.

First, the technical requirements and design specifications to be met are defined. Afterwards, the logical design of the global system, and of each of the modules that form it, is carried out in accordance with these requirements. Later, the system is implemented using VHDL language. Finally, the developed system is evaluated to verify that it achieves with the technical specifications and works properly in an architecture representative of the possible model to be used in the nanosatellite.

After consider several alternatives for its achievement and evaluate its pros and cons, we opt for the idea developed thoroughly in this study. Finally, it is concluded that the use of technologies such as ZynqUltraScale + of Xilinx for space missions carried out by CubeSats, is a very interesting alternative to be taken into account; since it opens a wide range of possibilities to new applications by providing a large number of resources and flexibility.

Keywords

CubeSat, FPGA, VHDL, SoC, ZynqUltraScale+, APB, space, system design.

Resumen

La aparición de los CubeSats ha revolucionado la exploración espacial. En los últimos años ha aumentado su popularidad debido al bajo coste y tiempo de desarrollo, lo que ha facilitado el acceso al espacio a otras entidades diferentes de las tradicionales. Estos pequeños satélites, han provocado un incremento en las posibilidades de investigación y de desarrollo de aplicaciones futuras.

El presente proyecto evalúa una posible aplicación para su uso futuro en un CubeSat. En particular, prueba el desarrollo de un sistema de adquisición de imágenes, realizado con la arquitectura ZynqUltraScale+ . Para ello, se expone a un gran nivel de detalle un sistema generador y transmisor de datos a una tasa determinada.

En primer lugar se definen los requisitos técnicos y especificaciones de diseño a cumplir. Posteriormente, se realiza el diseño lógico del sistema global, y de cada uno de los módulos que lo conforman, acorde con estos requisitos. A continuación, se implementa el sistema haciendo uso del lenguaje VHDL. Finalmente, el sistema desarrollado se somete a evaluación para comprobar que cumple con las especificaciones técnicas y que funciona adecuadamente en una arquitectura representativa del posible modelo a emplear en el nanosatélite.

Tras plantear varias alternativas para su consecución y evaluar sus pros y contras, se opta por la idea desarrollada en profundidad en este estudio. Finalmente, se llega a la conclusión de que el uso de tecnologías como ZynqUltraScale+ de Xilinx para misiones espaciales realizadas por CubeSats, es una alternativa muy interesante a tener en cuenta; ya que abre un amplio abanico de posibilidades a nuevas aplicaciones al proporcionar un gran número de recursos y flexibilidad.

Palabras clave

CubeSat, FPGA, VHDL, SoC, ZynqUltraScale+, APB, espacio, diseño de sistemas.

Agradecimientos

Quisiera mostrar mi agradecimiento al SSEL (Space and Science & Engineering Lab), en concreto a su director Rafael Toledo Moreo por darme la oportunidad de realizar este proyecto. A David Lizán, por su inestimable ayuda y al 'Departamento de Electrónica, Tecnología de Computadoras y Proyectos' de la Escuela Técnica Superior de Ingeniería de Telecomunicación de la UPCT, por aportarme los conocimientos necesarios para afrontar este reto.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTADO DE ACRÓNIMOS

AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASM	Algorithmic State Machine
AXI	Advance eXtensible Interface
ESA	European Space Agency
FPGA	Field-Programmable Gate Array
NASA	National Aeronautics and Space Administration
SoC	System On a Chip
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Capítulo 1:

Introducción al problema

El 4 de octubre de 1957, sucedió un hecho que cambiaría la historia de la exploración espacial para siempre. La antigua Unión Soviética lanzaba al espacio el primer satélite artificial de la historia, el Sputnik I. Éste estaba formado por una esfera de aluminio de 58.5 cm de diámetro y 83.6 kg y contaba con dos trasmisores de radio. Su finalidad era estudiar la densidad de las capas altas de la atmósfera y la propagación de las ondas de radio en la ionosfera. El lanzamiento del Sputnik I se llevó a cabo en plena Guerra Fría entre los Estados Unidos y la Unión Soviética y fue el detonante del inicio de la carrera espacial que libraron ambos países durante 18 años.

Desde entonces, miles de satélites han sido puestos en órbita por un gran número de países y con fines tanto de investigación, como comerciales o militares

Programa OPAL de la Universidad de Stanford

El Laboratorio de Desarrollo de Sistemas Espaciales (SSDL) de la universidad de Stanford fue fundado en 1994 con el fin de proporcionar programas de aprendizaje, basados en proyectos que ayudasen a los estudiantes de ingeniería a ganar experiencia en el desarrollo de sistemas, para su uso en misiones espaciales reales. Se esperaba poder completar el diseño de un microsatélite en un tiempo inferior al período del paso de los estudiantes por la universidad.

El primer satélite desarrollado en este programa fue 'SAPHIRE' que, aunque en un principio no estaba prevista su puesta en órbita, finalmente si se realizó en el año 2001.

En abril de 1995, comenzó el diseño del segundo satélite llamado 'OPAL'. Se trababa de un satélite contenedor que albergaba en su interior otros de inferior tamaño. Los cuáles eran liberados al espacio cuando el satélite principal alcanzaba su órbita. El satélite OPAL fue lanzado el 22 de enero del año 2000 y transportaba en su interior seis picosatélites, cuatro con unas dimensiones aproximadas de 10x8x2,5 cm y otros dos de 20x8x2,5 cm.

Nacimiento del CubeSat

El programa OPAL, encabezado por el profesor Robert Twiggs, no sólo sirvió para la adquisición de conocimientos y experiencia en el lanzamiento de picosatélites; sino que, abrió la puerta a una nueva era de experimentación espacial. Es entonces, cuando gracias a la colaboración entre la Universidad de Stanford y la Universidad Estatal Politécnica de California, nace el concepto de 'CubeSat'.

Robert Twiggs y Jordi Puig-Suari crearon el estándar de desarrollo de estos pequeños satélites, el cuál ha sido adoptado desde entonces por multitud de organizaciones de todo el mundo y para diversos fines, tanto de investigación, como comerciales.

Un CubeSat es un tipo de nanosatélite compuesto de una o varias unidades cúbicas de 10 cm de arista y 1 Kg de peso, cuyo coste, tiempo de desarrollo y complejidad es considerablemente inferior que la de un satélite tradicional. Debido a esto, es de especial interés investigar nuevas tecnologías y su potencial explotación en futuras misiones.

Figura 1: *Despliegue de un conjunto de CubeSats* (NASA, 2017)

Componentes de un CubeSat

Los componentes de un CubeSat pueden variar en función de la misión a acometer. Pero, por lo general, se destacan los siguientes subsistemas descritos a continuación.

Estructuras

Son el chasis del satélite y contienen el resto de componentes del sistema. La mayoría están fabricadas a partir de un tipo de aluminio específico que soporta las condiciones extremas y pueden adoptar diferentes configuraciones, dependiendo de los requisitos de la misión y el conjunto de unidades cúbicas que alberguen.

Energía

El sistema de energía gestiona la generación, almacenamiento y gestión de la energía eléctrica y, normalmente, constituye un tercio de la masa total del satélite. Habitualmente la energía es generada gracias a células fotovoltaicas, que transforman la energía lumínica en energía eléctrica y es almacenada gracias a las baterías de las que dispone el sistema. Por último, los sistemas de gestión y distribución de energía del dispositivo permiten controlar el flujo de energía de los componentes y suelen estar diseñados a medida según el tipo de misión.

Orientación, navegación y control

Este subsistema consta de numerosos componentes que permiten determinar la posición y orientación del sistema; así como el control de las mismas. Entre otros elementos, se encuentran:

- Ruedas de reacción: permiten que el satélite pueda realizar pequeños giros.

- Rastreador de estrellas: mediante la comparación de una imagen digital capturada por el satélite y un catálogo de estrellas se puede estimar la orientación del mismo.

- Magnetómetros: son capaces de medir el campo magnético local. Este resultado puede ser empleado para estimar tanto la orientación del satélite como la órbita en la que se encuentra.

- Sensores solares y terrestres : usados para hallar la ubicación del satélite respecto del Sol, o la Tierra respectivamente y, de esta forma, estimar su posición.

- Receptores GPS: son el método principal empleado para determinar la órbita para satélites de órbita baja terrestre (Low Earth Orbit).

Comunicaciones

Este subsistema permite al satélite transmitir datos a la Tierra y recibir tanto información, como comandos de ésta. La comunicación entre ambos se realiza empleando el espectro de radiofrecuencia entre 30MHz y 40 GHz.

La potencia de transmisión decrece con la distancia, por lo que normalmente las naves espaciales suelen emplear antenas parabólicas, ya que éstas son capaces de focalizar la transmisión en una dirección concreta y precisa. Sin embargo, debido a su envergadura y peso, estas antenas no son viables para un CubeSat. Por este motivo, una alternativa frecuentemente usada es el uso de antenas desplegables.

Control y manejo de datos

Los desarrolladores de naves espaciales pequeñas, especialmente CubeSats continúan usando microcontroladores y FPGAs que soportan varios núcleos de procesador. Estas últimas ya han sido empleadas con éxito en misiones espaciales con anterioridad y continúan haciéndolo gracias a su flexibilidad, alto rendimiento e integración; proporcionando periféricos, memorias on-chip y una mayor eficiencia energética.

Fabricantes como Xilinx ofrecen un gran número de componentes electrónicos para su uso específico en el espacio, los cuáles han sido diseñados para cumplir con las exigencias de rendimiento, fiabilidad y ciclo de vida en entornos extremos, incluyendo la exposición a la tan temible radiación cósmica. Esto permite reducir el tiempo de diseño, el coste económico y proporcionar mayor flexibilidad frente a los dispositivos ASIC. Además de estos componentes diseñados específicamente para soportar condiciones extremas, existen otras tecnologías de uso comercial muy interesantes para ser usadas en futuras misiones llevadas a cabo por CubeSats.

Entre estas tecnologías destaca ZynqUltraScale+ de Xilinx. Esta es la nueva familia de SoC (System-on-Chip) de Xilinx que integra un procesador Quad ARM Cortex-A53 de 64 bits, Dual Cortex-R5 de 32 bits para aplicaciones de tiempo real, GPU Mali 400 y una parte de lógica programable.

La familia Zynq UltraScale+ MPSoC consta de tres variantes distintas, lo que proporciona una gran flexibilidad de aplicación. Para uso aeroespacial y de defensa destacan los dispositivos EG, los cuales están equipados con procesadores Quad-Core. Su lógica programable es reconfigurable a través del software, permitiendo la actualización de parte del sistema sintetizado durante el transcurso de la misión. El diagrama de bloques completo está recogido en la *figura 2*.

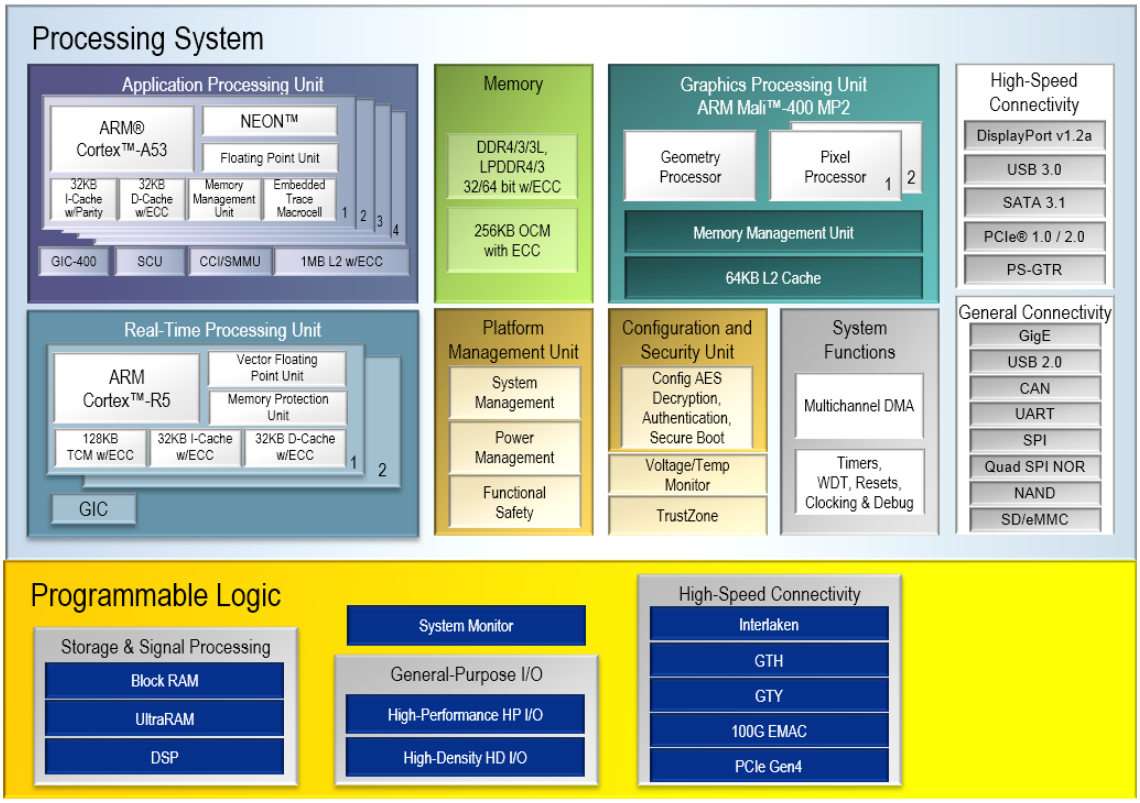


Figura 2: *Diagrama de bloques del dispositivo Zynq UltraScale+ EG* (Xilinx)

Capítulo 2 :

Objetivo y fases de desarrollo

2.1. Objetivo

Tradicionalmente, el llevar a cabo misiones espaciales estaba reservado a grandes instituciones con una gran cantidad de recursos. Este paradigma ha cambiado por completo en los últimos años con la aparición de los CubeSats. El bajo coste y tiempo de desarrollo de estos pequeños satélites ha facilitado el acceso al espacio a otras entidades aparte de las tradicionales, tanto con fines de investigación como comerciales. Cabe destacar el gran aumento de proyectos con fines académicos llevados a cabo por universidades de todo el mundo que ha tenido lugar en los últimos años para investigar posibles aplicaciones útiles de este tipo de nanosatélites. Uno de estos proyectos es el presente Trabajo Fin de Grado, cuyo objetivo es evaluar una posible aplicación futura para un CubeSat.

En particular, la finalidad perseguida detrás de la realización de este proyecto es la de testar un sistema de adquisición de imágenes para su uso en el espacio, realizado con la arquitectura ZynqUltraScale+. Para poder llevarlo a cabo será necesaria la creación de un sistema que suministre una cantidad de datos a una tasa determinada. Por este motivo, se desarrollará un generador de datos basado en FPGA y una aplicación en lenguaje C que se ejecutará en un PC Windows.

El generador de datos será implementado en lenguaje VHDL y deberá cumplir las características para ser un sistema fiable, robusto y satisfacer las especificaciones técnicas requeridas. Por otra parte, la aplicación C desarrollada debe permitir la completa evaluación del sistema creado; además de su control.

2.2. Fases de desarrollo del proyecto

Una vez que se han identificado los objetivos y finalidad del proyecto, hay que identificar las fases que conlleva la realización del mismo.

Fase I: Definición de requisitos técnicos.

- Estudio de la especificación AMBA y el protocolo de comunicaciones APB.

Fase II: Solución técnica.

- En este apartado se realizará el diseño lógico del sistema y el estudio a nivel de bloques funcionales y de comunicación.
  - Diseño de un módulo de transmisión de datos en paralelo hacia el sistema de adquisición.
  - Diseño de un módulo de cola FIFO.
  - Desarrollo de un módulo Arbiter.

Fase III: Realización del diseño

- Implementación
  Se implementarán en lenguaje VHDL todos los módulos diseñados en el apartado anterior empleando la herramienta 'ISE Design Suite'.

- Integración
  Se incorporarán todos los módulos diseñados, junto con el módulo USB-APB proporcionado, en un mismo proyecto VHDL y se realizará la síntesis del mismo.

Fase IV: Evaluación del sistema

- Verificación
  - Simulación individual de cada módulo.
  - Simulación global del sistema.

- Validación
  - Desarrollo de una aplicación capaz de comunicarse mediante USB con el sistema desarrollado..
  - Se probará el sistema desarrollado en una arquitectura hardware representativa del posible modelo a emplear en un CubeSat.

Capítulo 3:

Definición de requisitos técnicos

Existen muchos tipos de requisitos a cumplir. Éstos pueden ser a nivel funcional, de rendimiento, de comunicaciones, etc. Debido a su especial relevancia en el desarrollo del sistema, a continuación se hace especial hincapié en el bus AMBA APB.

3.1. Estudio de la especificación AMBA

Para que todos los componentes del diseño se comuniquen entre sí de una forma eficiente, será necesario definir la arquitectura de bus adecuada. Se deberá escoger teniendo en consideración el carácter espacial del presente proyecto. El estándar de comunicaciones espacial adoptado en este tipo de misiones por la Agencia Espacial Europea es AMBA AHB/APB de ARM y es también el que se adoptará en el desarrollo del sistema.

Todas las comunicaciones entre los diferentes módulos han sido realizadas siguiendo la especificación del protocolo AMBA 3 APB. Por este motivo, es primordial dedicar un apartado a explicar la especificación del bus AMBA, y más en concreto, el protocolo de la interfaz APB.

3.1.1. AMBA(Advanced Microcontroller Bus Architecture)

AMBA es un estándar abierto de ARM para la conexión y gestión de bloques funcionales en un SoC. Básicamente, engloba un conjunto de protocolos de comunicaciones que define cómo se comunican los bloques funcionales entre ellos. Esta especificación está bastante extendida y adoptada y proporciona una interfaz estándar que favorece la reutilización de IP (Intelectual Property).

La *figura 3* muestra la evolución de la especificación AMBA a lo largo de tiempo, según el año en el que ARM presentó cada uno de los buses.

Los más significativos son:

APB (Advanced Peripheral Bus): ideado para la conexión de periféricos que demanden poco ancho de banda.

AHB (Advanced High-performance Bus): bus empleado para conectar componentes que requieren de gran ancho de banda como pueden ser procesadores o memorias on-chip.

AXI (Advance eXtensible Interface): diseñado para sistemas de muy alta frecuencia y alto rendimiento.

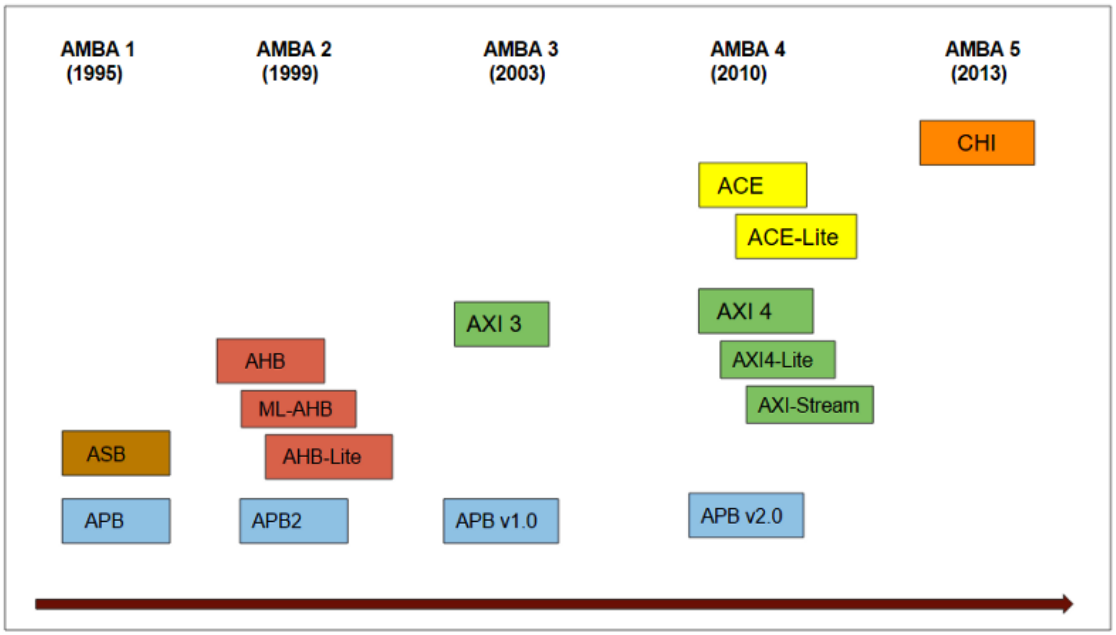


Figura 3: *Evolución de la especificación AMBA* (IJRASET, 2018)

A continuación, se describirá con detalle el funcionamiento de la versión APB v1.0 de la familia AMBA 3 (año 2003), la cuál ha sido tomada de referencia para el diseño de las interfaces de todos los módulos.

3.1.2. APB (Advanced Peripheral Bus)

El protocolo APB proporciona una interfaz sencilla y con un bajo consumo de energía orientada a la interconexión de periféricos de bajo ancho de banda que no requieran un alto rendimiento. Es un protocolo 'unpipelined', lo que conlleva que una nueva transferencia no puede ser realizada hasta que la anterior se haya completado.

Está basado en la arquitectura de red de Maestro/Esclavo, cuyas características básicas grosso modo son:

- Un único maestro accede al bus en un momento determinado.

- Uno o varios módulos esclavos pueden estar conectados al mismo maestro.

- El maestro es el que inicia la comunicación.

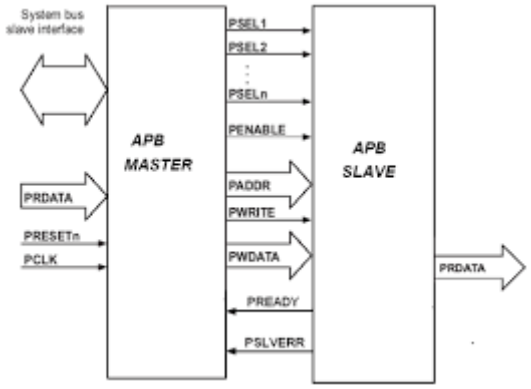


Figura 4: *Diagrama de la interfaz APB Maestro-Esclavo* **(IJSEAS, 2015)**

3.1.2.1. Transferencias APB

El bus APB permite realizar transferencias con o sin estados de espera. La diferencia entre ambas radica en el tiempo en el que el esclavo comandado tarda en responder una vez se ha accedido a él. La duración mínima de una transferencia APB es de dos ciclos de reloj y corresponde a una transferencia sin estados de espera. Para una transferencia con estados de espera este tiempo es variable desde dos, hasta el número de ciclos que se consideren oportunos; aunque, es recomendable que este número sea reducido.

Transferencia sin estados de espera

Tanto la transferencia de escritura, como de lectura se inician con el cambio de la dirección (PADDR), la señal de escritura/lectura (PWRITE) y la señal de selección (PSEL) en el flanco ascendente de reloj. En el caso de la operación de escritura, también lo hará el bus con los datos a escribir (PWDATA). Esta primera fase de la transferencia se denomina 'SETUP'. En el ciclo inmediatamente posterior de reloj, la transferencia se encontrará en la fase de 'ACCESS', fase en la que la señal 'PENABLE' estará activa y el maestro podrá leer los datos procedentes del esclavo en el bus 'PRDATA'. La transferencia concluye cuando el maestro recibe la señal de 'PREADY' procedente del esclavo.

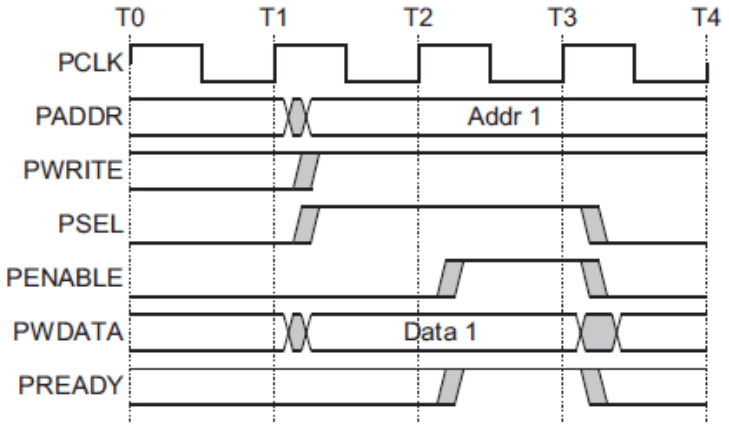


Figura 5: *Transferencia de escritura APB sin estados de espera* (ARM, Write transfer with no wait states, 2004)

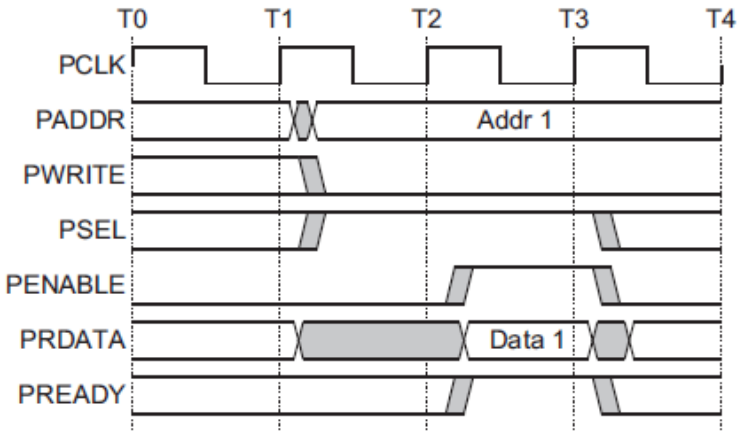


Figura 6: *Transferencia de lectura APB sin estados de espera* (ARM, Read transfer with no wait states, 2004)

Transferencia con estados de espera

La única diferencia respecto a las transferencias sin estados de espera, es que en este caso la fase de 'ACCESS' dura más de un ciclo de reloj. Sucede cuando el esclavo extiende esta fase, activando la señal de 'PREADY' más tarde. Los datos procedentes del esclavo seguirán estando disponibles en el último ciclo de reloj de la transferencia APB, al igual que la señal de error 'PSLVERR'.

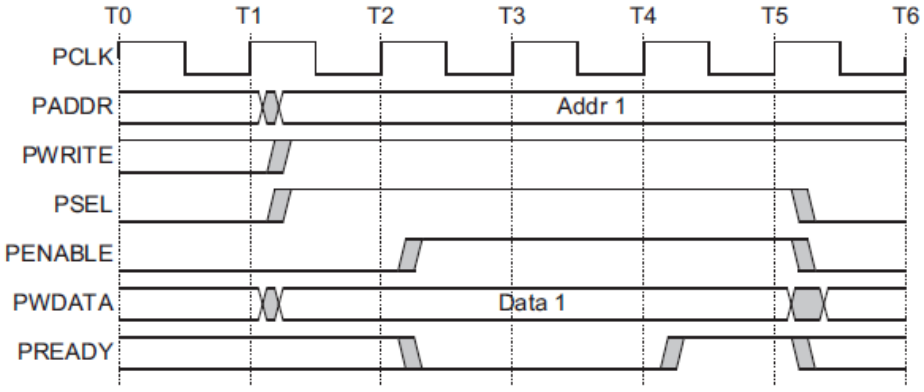


Figura 7: *Transferencia de escritura APB con estados de espera* (ARM, Write transfer with wait states, 2004)

Respuesta de error

Para indicar una condición de error en la transferencia se emplea la señal 'PSLVERR', proporcionada por el esclavo en el último ciclo de la transferencia APB. Las condiciones de error pueden darse tanto en operaciones de lectura, como de escritura y dependen del diseño del sistema.

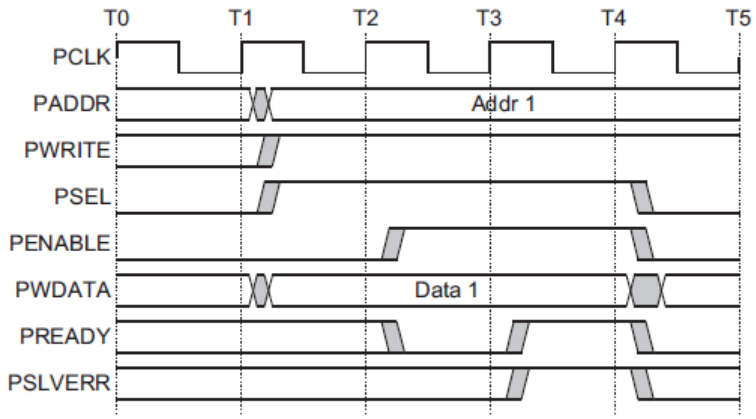


Figura 8: *Ejemplo de una transferencia APB de escritura fallida* (ARM, Example failing write transfer, 2004)

3.1.2.2. Estados de Operación

La interfaz APB maestro de los componentes del diseño ha sido implementada teniendo en cuenta los estados de operación definidos en la especificación. La máquina de estados de la especificación original ha sido modificada ligeramente para que sea necesario pasar por el estado 'IDLE' para realizar cada transacción APB, aunque ésta provenga del mismo esclavo.

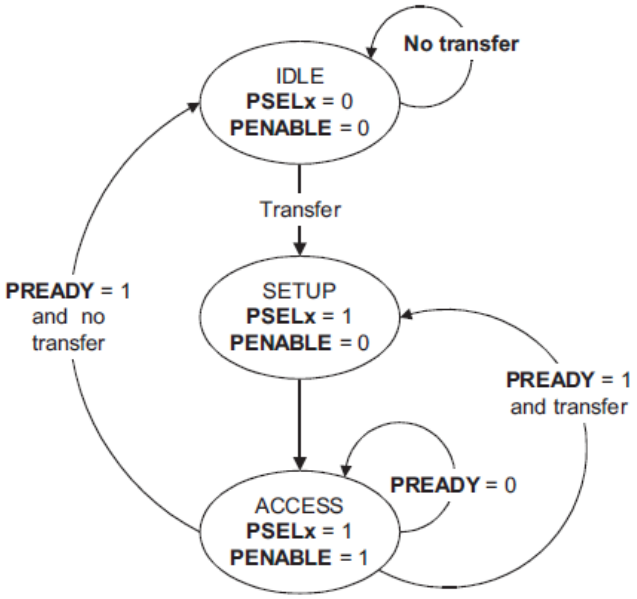


Figura 9: *Diagrama de estados de operación de la especificación APB* (ARM, State diagram, 2004)

3.1.2.3. Descripción de señales de la interfaz APB

Las señales que conforman la interface APB están descritas en la *tabla 1*, junto con el tamaño en bits con el cual han sido implementadas en los diferentes módulos que conforman el sistema. En los consecutivos capítulos, cuando se haga referencia a esta interfaz, se tendrán en consideración las características aquí descritas.

Señal	Fuente	Tamaño (en bits)	Descripción
PCLK	Maestro	1	Señal de reloj. El resto de señales están basadas en el flanco ascendente.
PRESETn	Maestro	1	Reset activo a nivel bajo.
PADDR	Maestro	8	Bus de dirección
PSELx	Maestro	15	Bus generado por el maestro dirigido a cada esclavo del sistema, cuya utilidad es la de seleccionar un esclavo determinado. Emplea codificación 'one hot'.
PENABLE	Maestro	1	Indica el segundo y consecutivos ciclos de la transferencia APB.
PWRITE	Maestro	1	Cuando se encuentra a nivel lógico alto indica una operación de escritura. Cuando está en nivel bajo de lectura.
PWDATA	Maestro	32	Bus de datos a escribir en el esclavo durante los ciclos de escritura y mientras que la señal PWRITE esté a nivel alto.
PREADY	Esclavo	1	El esclavo usa esta señal para completar la transferencia APB o prolongarla con estados de espera.
PRDATA	Esclavo	32	La lectura de este bus sólo es válida en una operación de lectura y siempre que la señal PREADY esté en alto.
PSLVERR	Esclavo	1	Indica un fallo en la transferencia.

Tabla 1: *Listado descriptivo de las señales APB*

Capítulo 4

Solución Técnica

4.1. Descripción general del sistema

El sistema generador de datos desarrollado consta de los siguientes módulos:

- Módulo de cola FIFO que almacena los datos a transmitir.
- Módulo transmisor de datos en paralelo hacia el sistema de adquisición.
- Módulo Arbiter encargado de gestionar el bus APB.
- Módulo USB-APB y drivers en C proporcionados para comandar el resto de módulos con la ayuda de la aplicación en lenguaje C desarrollada.

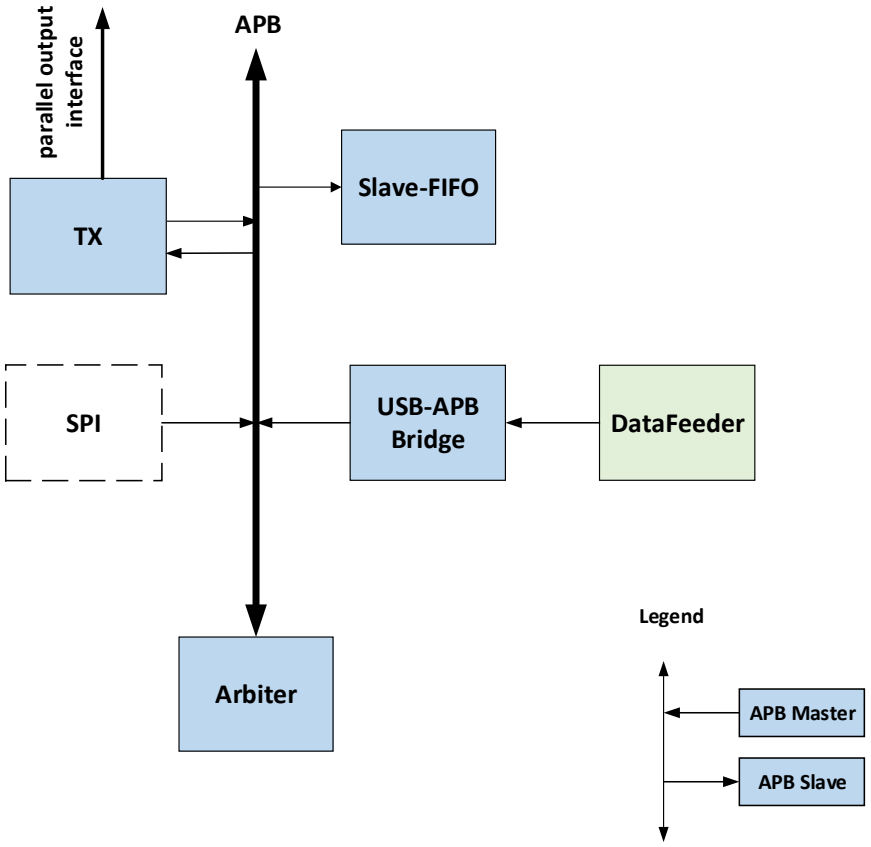


Figura 10: *Diagrama completo del sistema*

Además, el sistema ha sido diseñado para poder albergar en un futuro un módulo APB maestro más, el módulo SPI. Es por este motivo, por el que en el diagrama de la figura anterior este componente aparece con una línea discontinua.

La siguiente tabla recoge el mapa de memoria del sistema en su totalidad.

PSEL	Dirección	Módulo	Registro
0x0001	0x0000	Slave-FIFO	STATUS register
0x0001	0x0001	Slave-FIFO	FIFO
0x0010	0x0000	TX	STATUS register
0x0010	0x0001	TX	TRANSFER register

Tabla 2: *Mapa de memoria del sistema*

4.1.1. Secuencia de operación básica

Previamente a describir en detalle cada uno de los módulos que conforman este sistema y su funcionalidad, se expone cuál va a ser la secuencia de comunicación esperada del sistema.

1. El usuario haciendo uso de la aplicación desarrollada en lenguaje C para entorno Windows empleará el módulo USB-APB proporcionado para introducir un número de datos en el buffer FIFO del módulo Slave-FIFO.

2. El usuario entonces volverá a hacer uso de la aplicación en C y del módulo USB-APB para comandar al módulo TX la lectura de un número determinado de datos del Slave-FIFO y su posterior transmisión en paralelo hacia el sistema de adquisición.

Esta sería la secuencia funcional básica; sin embargo, gracias a las características de los módulos del sistema y a la aplicación en lenguaje C, el usuario puede realizar otras acciones como consultar los registros de cada módulo o vaciar el búfer FIFO. Los detalles de estas operaciones serán explicados en los siguientes apartados.

4.2. Descripción detallada de los módulos del sistema

4.2.1 Módulo Slave-FIFO

4.2.1.1. Características

Este módulo posee una interfaz esclavo APB de 32 bits, un buffer FIFO basado en registro como componente jerárquico, una máquina de estados que controla tanto el acceso al búfer, como la respuesta por parte del módulo y un registro de estado, además de toda la lógica necesaria para su adecuado funcionamiento. El diagrama de bloques completo en el que se aprecian todos los elementos del módulo está recogido en la *figura 12*.

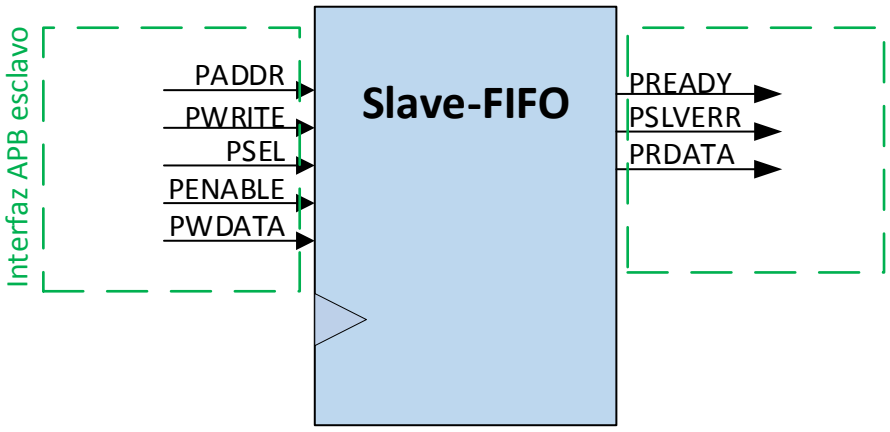


Figura 11: *Interfaz APB del módulo Slave-FIFO*

4.2.1.2. Funcionalidad

La finalidad del módulo Slave-FIFO es la de almacenar un conjunto de datos introducidos por el usuario hasta que el módulo transmisor o el propio usuario los solicite. Dispone de un registro de estado, el cual muestra el estado actualizado del componente. Cualquier comportamiento anómalo será notificado mediante la activación de la señal de 'PSLVERR' y se almacenará un código identificativo de error en el registro para una posterior consulta.

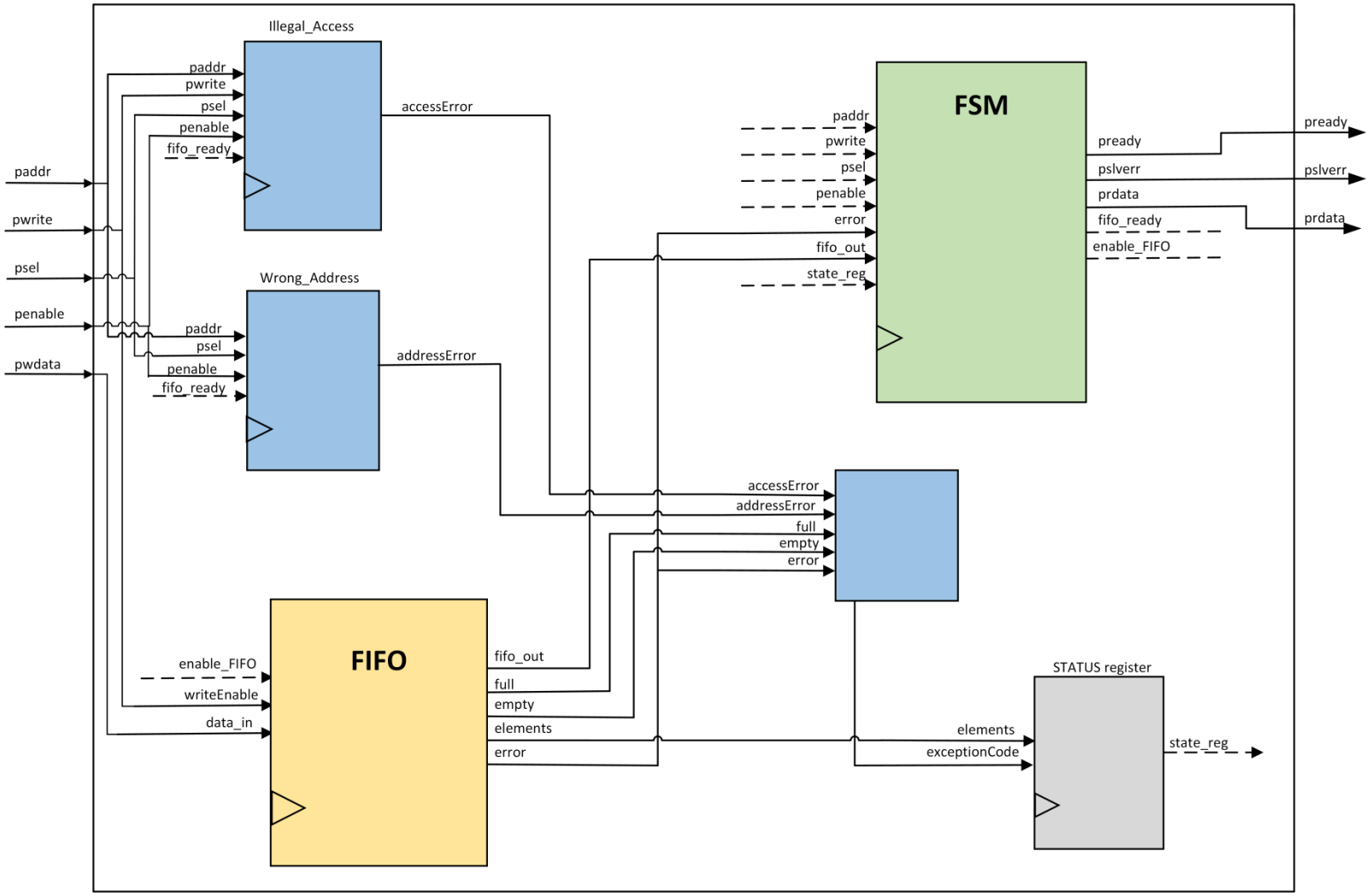


Figura 12: *Diagrama de bloques del módulo Slave-FIFO*

4.2.1.3. Registros

- **STATUS REGISTER (ADDRESS 0x0000)**

Este registro de estado, cuyo tamaño es de 32 bits contiene el número de elementos disponibles en el buffer y un código de excepción que indica el estado actual del módulo. Únicamente puede ser modificado por el módulo propietario que lo alberga, en este caso, el Slave-FIFO. Sin embargo, sí puede ser consultado por el resto de módulos del sistema. Previene que se cometan posibles errores de acceso a la FIFO; ya que, puede ser consultado previamente a comandar una orden de escritura o lectura en ésta y saber de cuántos elementos dispone.

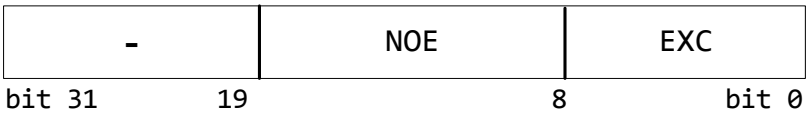


Figura 13: *Registro de estado del módulo Slave-FIFO*

Descripción de los campos del registro:

bit 31-19 **Reserved**

bit 18-8 **NOE :** Number Of Elements

Indica el número de elementos disponibles en la FIFO.

bit 7-0 **EXC :** Exception Code

Contiene el estado actual del módulo, indicado por el código de excepción correspondiente.

4.2.1.4. Búfer FIFO

FIFO es el acrónimo de 'First In, First Out'. Es un método de procesamiento y recuperación de datos en el que los datos son servidos según el orden en el que han sido introducidos. Es decir, los primeros elementos del búfer son los primeros en ser procesados.

Es una estructura bastante común debido a su sencillez conceptual y suele emplearse como medida de almacenamiento temporal entre dos sistemas que operan a diferentes velocidades.

En el sistema desarrollado se ha implementado basada en una RAM de doble puerto y siguiendo la lógica de un búfer circular con *'roll-over'*. El acceso a la posición de la memoria en la que se introducirá el dato está controlada por un puntero de escritura y de la que se extraerá por uno de lectura. Las posiciones de ambos punteros varían dinámicamente, conforme se leen o escriben datos del búfer y la distancia máxima entre ellos puede ser desde cero, hasta el número máximo de elementos almacenados. Estos dos últimos estados en los que se puede encontrar el búfer son de especial interés y es necesario implementar algún tipo de lógica que permita diferenciarlos del resto, como por ejemplo dos señales que adviertan cuando se han alcanzado sendas condiciones. Dichas señales son la señal de 'Empty', que indica que el búfer se encuentra vacío y la señal de 'Full', que advierte de que éste está lleno.

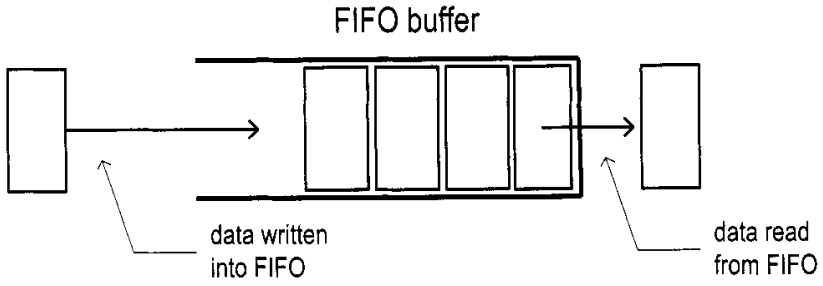


Figura 14: *Diagrama conceptual de un búfer FIFO* **(P.Chu, 2008)**

Para comprender el funcionamiento del búfer mejor se ilustra un ejemplo sencillo de la lógica empleada para una FIFO con capacidad para cuatro datos, dónde 'rd' y wr' son los punteros de las siguientes posiciones a leer y escribir, respectivamente.

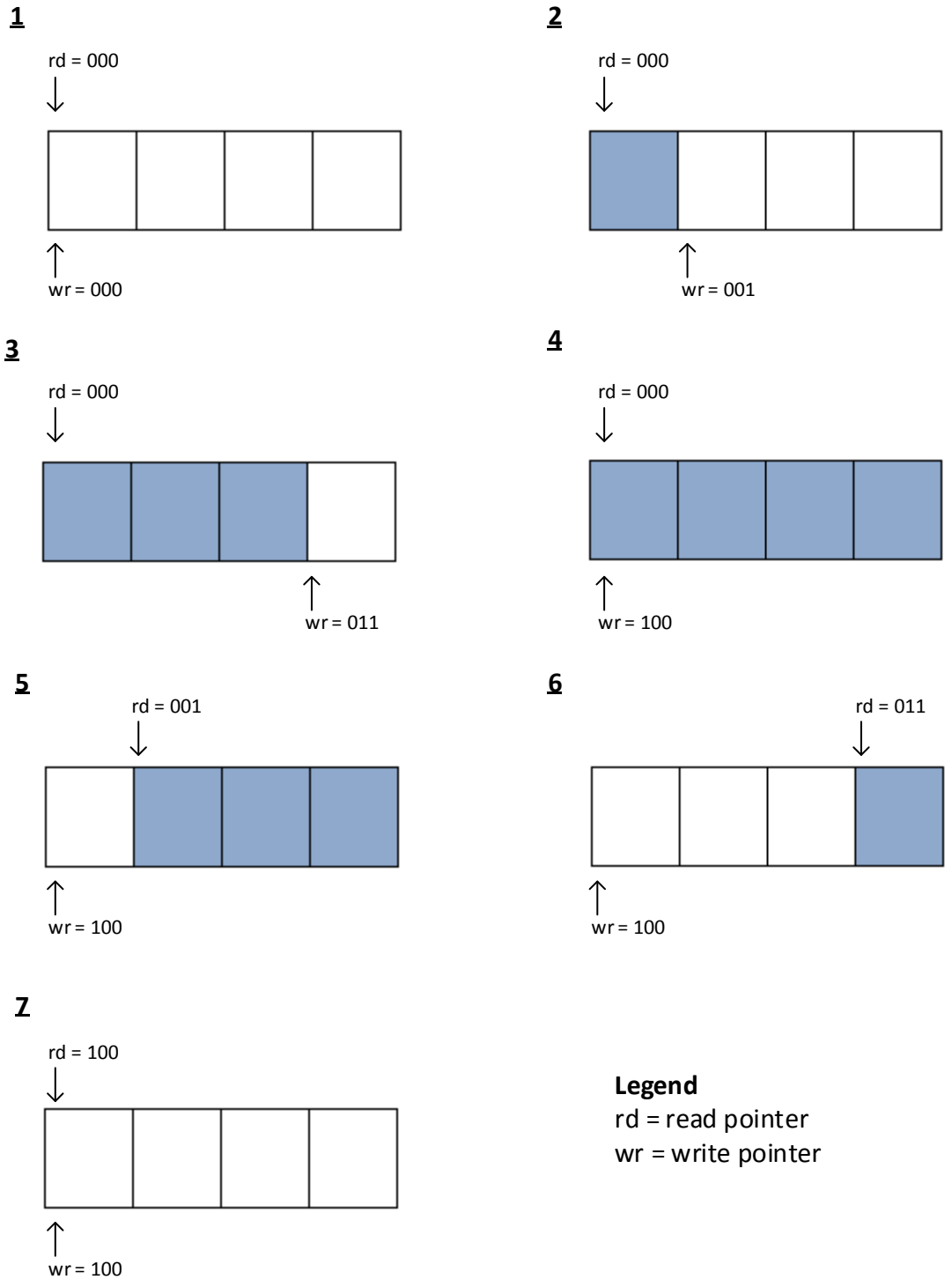


Figura 15: *Ejemplo teórico ilustrativo de una secuencia de acceso al búfer FIFO.*

La secuencia de operación llevada a cabo en el ejemplo anterior es la siguiente:

1. El búfer se encuentra en su estado inicial vacío. La señal de 'Empty', que advierte esta condición, estará a nivel lógico alto.

2. Se introduce un dato en el búfer, lo que conlleva a que el puntero de escritura se incremente en una unidad. El puntero de lectura no sufre variación alguna y la señal de 'Empty' pasará a estar a nivel lógico bajo.

3. En este punto se introducen dos datos más. El puntero de escritura aumenta en dos unidades y no se produce ningún otro cambio.

4. Se introduce un dato más, provocando que todas las posiciones del búfer estén ocupadas. Como se puede apreciar en la imagen, en este instante el valor del puntero de escritura y el de lectura coinciden bit a bit, exceptuando el MSB. Cuando se cumple esta condición, el búfer se encuentra lleno y la señal de 'Full' que advierte de esta condición debe estar activa.

5. Se lee un dato del búfer, aumentando el índice de lectura y provocando la desactivación de la señal de 'Full'.

6. Se leen dos datos más del búfer, conllevando el aumento del índice de lectura y no produciendo ningún efecto más en el componente.

7. Se lee el último dato disponible en la FIFO y, por lo tanto, se vacía el búfer. Cuando esto sucede, el índice de lectura y de escritura coinciden bit a bit, incluyendo el MSB. La señal de 'Empty' volverá a encontrarse a nivel alto en esta situación.

Fijándonos en el ejemplo, se puede apreciar que el puntero de escritura y el de lectura contienen un bit más que los necesarios para direccionar todas las posiciones de memoria del búfer. Este bit adicional se emplea para diferenciar entre la condición de 'Empty' y 'Full' de la FIFO cuando se haya producido 'roll-over'; es decir, cuando se haya alcanzado el máximo. Si no se hiciese uso de este bit adicional, en el caso 4 de la figura anterior no se podría diferenciar si el búfer se encuentra vacío o lleno únicamente comparando ambos punteros. En este caso, tendríamos que disponer de algún elemento de memoria adicional para conocer si la operación inmediata anterior

ha sido de escritura o de lectura y, de esta forma, conocer en cuál de los dos estados particulares se encuentra.

Conforme a la aproximación propuesta, el número de elementos de la FIFO deberá ser potencia de dos. Si se cumple esta condición, el número de bits necesarios de ambos punteros para poder direccionar todas las posiciones será:

$$indexbits = \log_2(2 * FIFOdepth)$$

Implementación

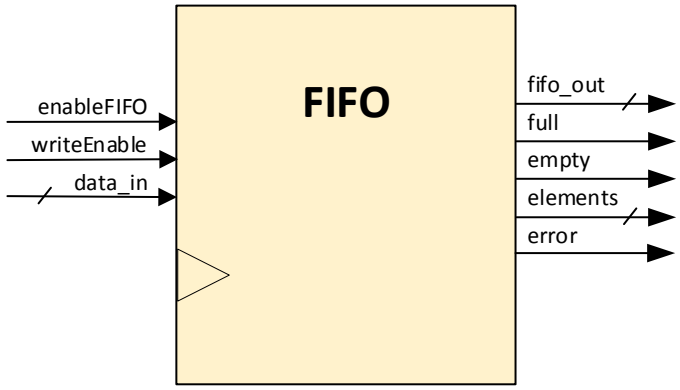


Figura 16: *Interfaz del búfer FIFO*

El módulo Slave-FIFO desarrollado contiene como componente una FIFO basada en registro con capacidad para 1024 datos de 32 bits. Siguiendo el criterio de acceso anteriormente descrito, el número de bits necesarios de ambos punteros para que se cumplan las condiciones descritas será:

$$indexbits = \log_2(2 * 1024) = 11$$

A continuación en la siguiente tabla se describen las señales, tanto de entrada, como de salida de la interfaz del componente

Señal	Descripción
ENABLE_FIFO	Habilita el acceso al búfer, tanto en operación de escritura, como de lectura.
WRITEENABLE	Cuando se encuentra a nivel lógico alto indica una operación de escritura. Cuando está en nivel bajo de lectura.
DATA_IN	Bus de entrada de 32 bits que contiene los datos a introducir en el búfer.
FIFO_OUT	Bus de salida de 32 bits con los datos leídos del búfer.
FULL	Activo cuando el búfer esté completamente lleno y no haya espacio para almacenar ningún elemento más.
EMPTY	Indica que el búfer se encuentra vacío.
ELEMENTS	Bus de 11 bits que contiene el número de datos almacenados en el búfer.
ERROR	Informa que se ha producido un error al intentar leer de un búfer vacío o introducir un dato en un búfer lleno.

Tabla 3: *Listado descriptivo de la interfaz del búfer FIFO*

4.2.1.5. Máquina de estados finitos del módulo Slave-FIFO

Al ser el búfer FIFO basado en registro un elemento síncrono, siempre será necesario al menos un ciclo de reloj para acceder a él, ya sea para escribir un dato o leerlo. Debido a esto, la transferencia APB presentará estados de espera. En el caso de no acceder al componente de la FIFO de nuestro módulo, sino a cualquier otro, como por ejemplo el registro de estado, esta transferencia sí podría ser llevada a cabo sin emplear estos estados. Sin embargo, el módulo Slave-FIFO ha sido construido para poder ser empleado en un sistema determinista, por lo que la respuesta siempre ha de tardar siempre lo mismo.

Para satisfacer esta condición, ha sido necesaria la creación de una máquina de estados para controlar el acceso a la FIFO y las señales de salida de la interfaz esclavo APB.

En la *figura 17* se representa el diagrama de estados simplificado de la máquina de estados en que únicamente aparecen las transiciones entre los diferentes estados y la condición necesaria para ello. No se han incluido las salidas de cada uno, que sí están detalladas en el diagrama ASM de la *figura 18*.

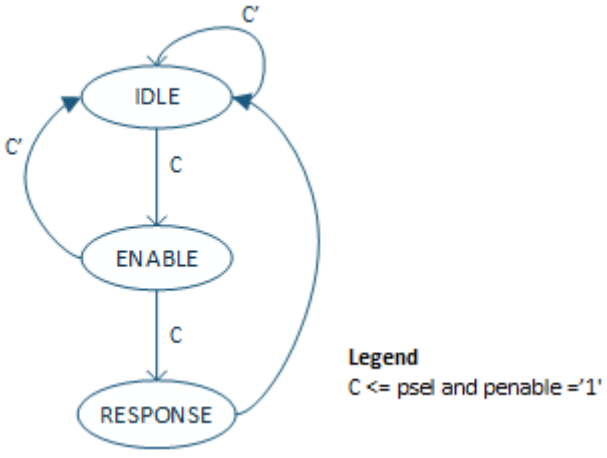


Figura 17: *Diagrama simplificado de estados del módulo Slave-FIFO*

La máquina de estados finitos operará entre los siguientes estados:

- **IDLE**
  Este es el estado inicial por defecto de la máquina de estados.

- **SETUP**
  Cuando se cumple la condición 'C', es decir las señales de PSEL y PENABLE están activas, significa que se está comandando una transferencia APB al módulo Slave-FIFO, por lo que la máquina de estados cambia al estado de ENABLE, en el que habilita la FIFO en el caso de que la dirección del bus PADDR se corresponda.

- **RESPONSE**
  En este estado se deshabilita la FIFO, en el caso de que lo esté y se devuelve la señal de PREADY, además de los valores correspondientes para PSLVERR y PRDATA, dependiendo de a qué parte del módulo se haya accedido. El módulo sólo se mantiene en el estado de RESPONSE por un ciclo de reloj y siempre cambia al estado de IDLE en el siguiente flanco de reloj ascendente.

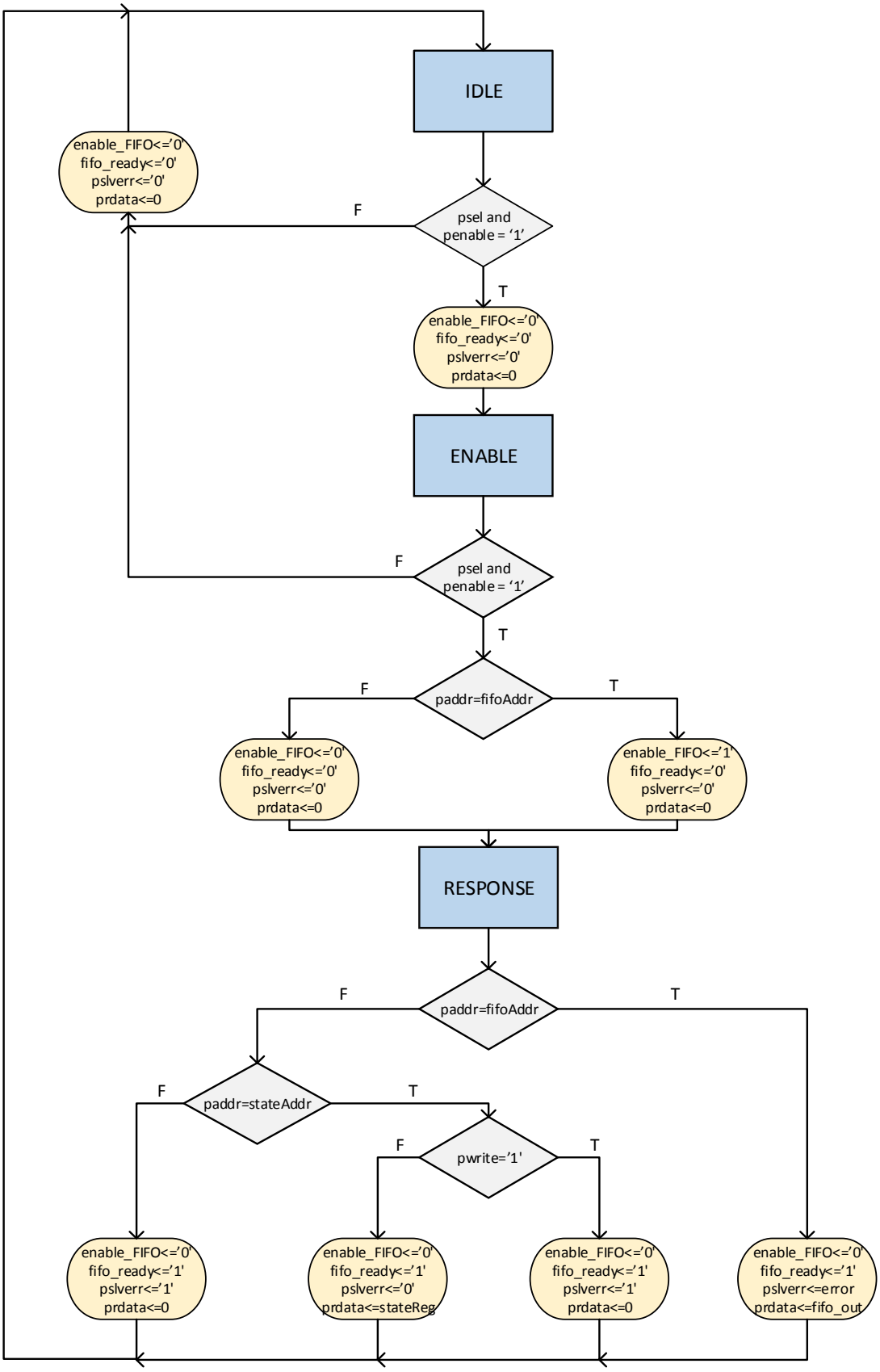


Figura 18: *Diagrama ASM del módulo Slave-FIFO*

4.2.1.6. Códigos de Excepción

El código de excepción refleja las posibles situaciones en las que se puede encontrar el módulo, ya sean éstas anómalas o normales. Las posibles situaciones de error contempladas, en las que la señal 'PSLVERR' de la interfaz APB cambiaría su estado lógico a alto se indican a continuación.

1. Intento de escritura en una FIFO llena

Aunque el usuario conoce el tamaño de la FIFO y puede saber el número de elementos disponibles en ella, no se da por supuesto que ordene la escritura de un número inferior de elementos que el espacio disponible en el buffer.

2. Intento de lectura de una FIFO vacía.

El usuario es capaz de conocer el número de elementos disponibles en la FIFO consultando el registro de estado. Sin embargo, no se presupone que éste ordene la lectura de un número inferior al existente, ya sea de forma directa extrayendo los datos directamente de él, o indirectamente cuando comanda la transmisión de datos al módulo transmisor y es éste quién los lee.

3. Intento de escritura en el registro de estado.

El registro de estado únicamente puede ser modificado por el propio módulo, nunca por cualquier otro componente del sistema. Es esencial que esté protegido frente a este hecho, además de ser notificado al usuario.

Este módulo está protegido frente a una posible escritura en la FIFO por parte del módulo de transmisión; ya que, como se verá cuando se detalle el módulo de transmisión, por diseño del mismo no es posible que éste pueda acceder a otra dirección del Slave-FIFO que no sea la del buffer FIFO y con otra operación que no sea la de lectura. Aún así, no está garantizado que otro módulo, como el propio usuario accediendo a él gracias al USB-APB, pueda intentarlo. Por este motivo, es necesario contemplar esta situación y que sea tratada como un error.

4. Intento de acceso a una dirección de memoria fuera de rango en nuestro componente.

Si se intenta acceder a una dirección inexistente en el módulo, se activará la señal de error y se actualizará el correspondiente código del registro de estado.

5. Escritura y lectura simultánea de la FIFO.

Otro posible error, sería el intento de escritura y lectura del búfer en el mismo instante de tiempo. Sin embargo, este caso es imposible que suceda por dos razones. La primera es que la señal que habilita la lectura o escritura del módulo Slave-FIFO es de un único bit y la segunda es que sólo un módulo maestro APB que puede tomar el bus en un momento determinado.

Existen otras dos situaciones que son de especial interés, pero no son consideradas errores; y por lo tanto, no conllevan la activación de la señal APB 'PSLVERR', pero sí es provechoso identificarlas y hacerlas constar. Éstas se dan cuando la FIFO se encuentra llena o vacía. En ambos casos sí se actualizará el código de excepción en el registro de estado, pero no se activará la señal de error.

A cada situación de excepción, conlleve ésta un error o no, se ha asociado un código de 8 bits que será recogido por el registro de estado para su posterior consulta. En la *tabla 4* se muestran los diferentes códigos de excepción para todos los casos contemplados, junto con su nombre, el tipo al que pertenecen y una breve descripción de los mismos.

Código	Nombre	Tipo	Descripción
0x01	FIFO full	Error	FIFO llena. No se permite escribir más en ella, hasta que se realice una operación de lectura
0x02	FIFO empty	Error	FIFO vacía. No es posible leer datos de ella.
0x03	FIFO Full Error	Error	Intento de escritura en una FIFO llena
0x04	FIFO Empty Error	Error	Intento de lectura en una FIFO vacía
0x05	Illegal Access	Excepción	Intento escritura en registro de Estado
0x06	Wrong Address	Excepción	Intento de lectura o escritura en una dirección no contemplada en el módulo.
0x00	Normal Status	Normal	Estado normal.

Tabla 4: *Listado de los códigos del registro de estado 'STATUS Register' del módulo Slave-FIFO*

4.2.2 Módulo TX

4.2.2.1 Características

Este módulo posee una interfaz de salida de datos en paralelo hacia el sistema de adquisición, dos interfaces APB, una maestro y otra esclavo y dos registros, uno de configuración de la transmisión y otro de estado. Además de toda la lógica necesaria para su adecuado funcionamiento. El diagrama de bloques completo, en el que se aprecian todos los elementos del módulo, está recogido en la *figura 20*.

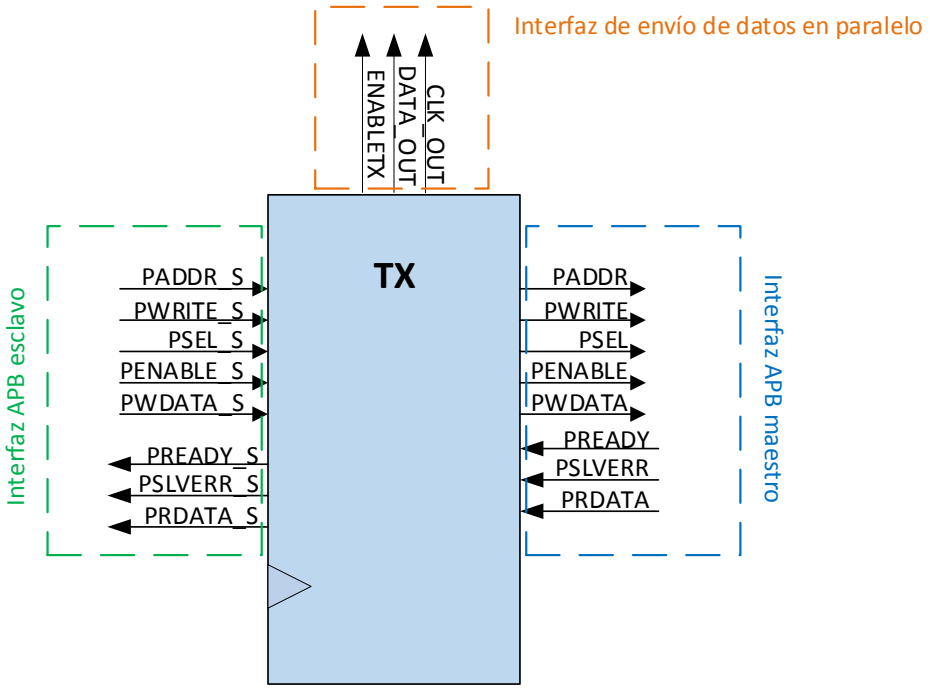
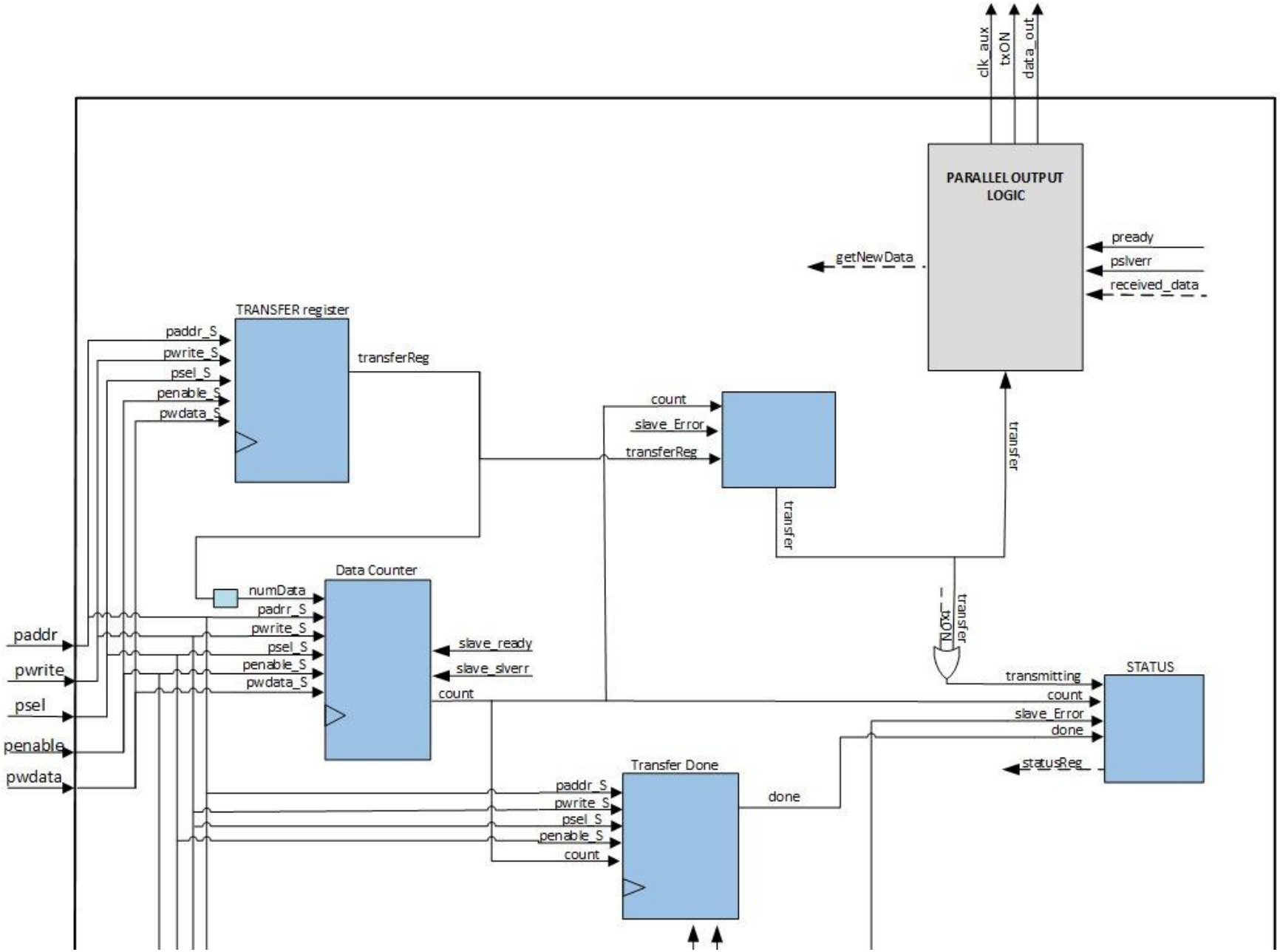


Figura 19: *Interfaz APB del módulo TX*


clk_aux
txON
data_out
PARALLEL OUTPUT LOGIC
pready
pslverr
received_data
getNewData
TRANSFER register
paddr_S
pwrite_S
psel_S
penable_S
pwdata_S
transferReg
count
slave_Error
transferReg
transfer
transfer
transfer
txON
Data Counter
numData
padrr_S
pwrite_S
psel_S
penable_S
pwdata_S
slave_ready
slave_slverr
count
paddr
pwrite
psel
penable
pwdata
STATUS
transmitting
count
slave_Error
done
statusReg
Transfer Done
paddr_S
pwrite_S
psel_S
penable_S
count
done

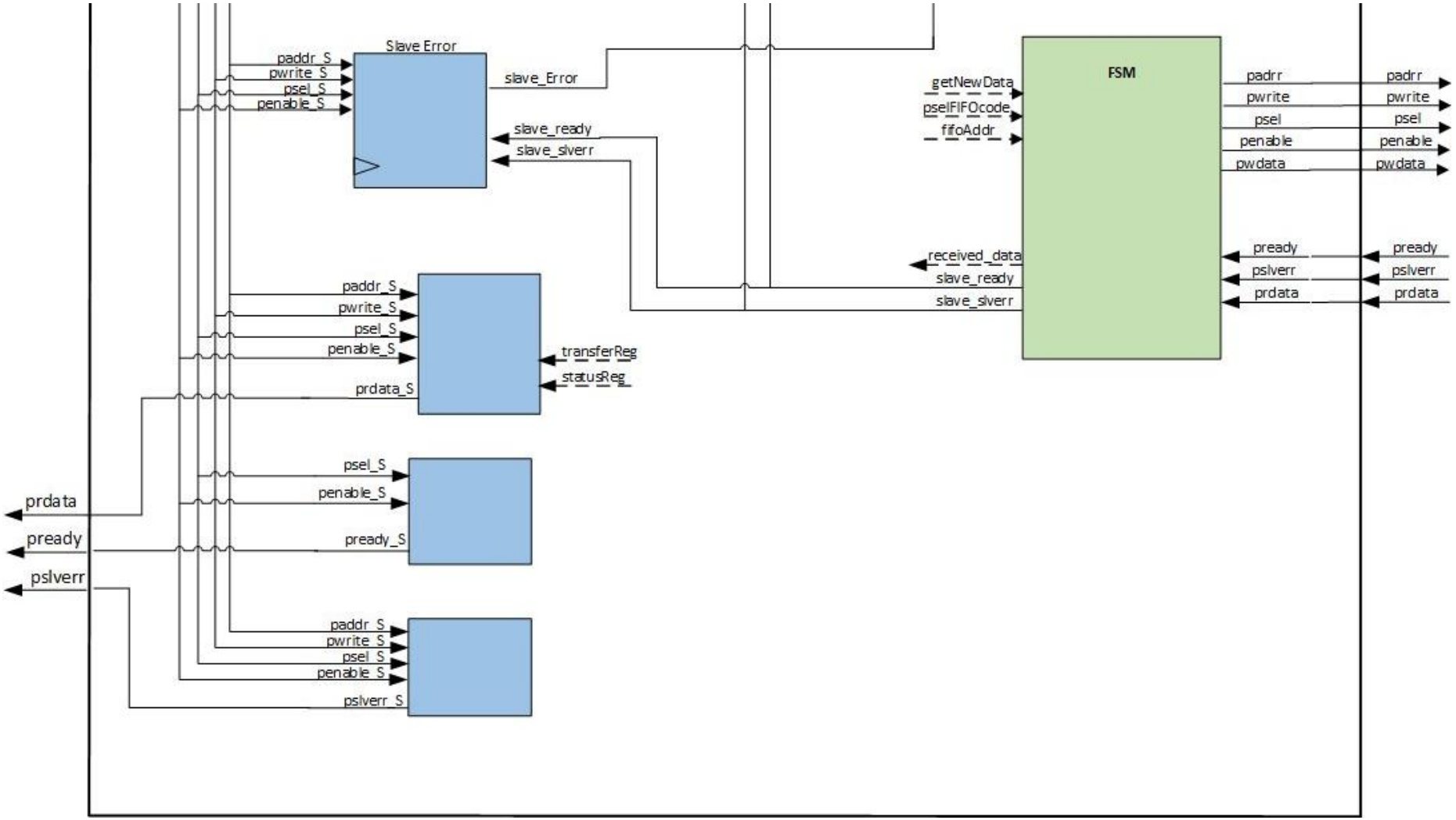


Figura 20: *Diagrama de bloques del módulo TX*

4.2.2.2 Funcionalidad

El módulo diseñado permite enviar al sistema de adquisición un número de datos, determinado por el usuario, a una tasa constante de 160 Mbits/s.

El usuario comandará al módulo de transmisión el número de elementos que desea que sean leídos y enviados al sistema receptor. Para ello, empleará el registro de configuración denominado 'TRANSFER Register' destinado a tal efecto. Entonces, el módulo transmisor se comunicará mediante el bus APB con el módulo Slave-FIFO para obtener los datos a transmitir.

En todo momento, el módulo de transmisión contiene información acerca del estado en el que se encuentra la misma o si ha ocurrido algún error; en cuyo caso sería interrumpida. Además, el envío de datos también puede ser cancelado en cualquier instante por el usuario, si así lo demandase. En este caso, si ya se ha extraído el dato del búfer FIFO, éste sí es transmitido para evitar su pérdida, pero ya no se solicitaría ninguno más.

Aunque es lógico pensar que el usuario es conocedor de la secuencia de órdenes comandadas a los diferentes módulos, no se ha dado por supuesto. Por este motivo, tanto el registro de configuración de la transmisión, como el de estado pueden ser consultados por el mismo.

El módulo TX cuenta con una protección adicional que le salvaguarda tanto a él mismo como a los demás elementos del sistema. La máquina de estados del módulo transmisor que gestiona el acceso al bus APB ha sido diseñada de tal forma que el módulo transmisor, haciendo uso de su interfaz de maestro APB, sólo puede comandar la lectura del módulo Slave-FIFO. Esta funcionalidad le salvaguarda a él de la posibilidad del envío de órdenes a sí mismo, evita que pueda modificar los datos alojados en la FIFO o los registros del módulo Slave-FIFO e impide posibles accesos erróneos a otros módulos.

4.2.2.3 Registros

El módulo transmisor posee dos registros de 32 bits para configurar y controlar el envío de datos hacia el sistema de adquisición.

Dirección	Nombre	Módulo con acceso	
		Lectura	Escritura
0x0000	STATUS Register	Todos	Módulo TX
0x0001	TRANSFER Register	Todos	Todos

Tabla 5: *Resumen de los registros del módulo transmisor*

- **STATUS REGISTER (ADDRESS 0x0000)**

Este registro contiene el estado en el que se encuentra la transferencia de datos comandada y si ha ocurrido algún error en el esclavo, como podría ser, por ejemplo, un intento de lectura de una FIFO vacía. A diferencia del registro de estado del módulo Slave-FIFO, este no almacena código alguno que identifique el tipo de error. En el caso de que se desease saber, habría que consultar el registro de estado del otro módulo.

Como es de esperar, este registro sólo puede ser modificado por el propio módulo transmisor y nunca por el resto de componentes del sistema, como está indicado en la *tabla 5*. No obstante, la lectura de su contenido sí está permitida para cualquier elemento del sistema. Por consideraciones de diseño, no se distingue entre los errores de intento de escritura en el registro de estado e intento de acceso a una dirección inexistente. En ambos casos se activa la señal del bus APB 'PSLVERR', pero no se guarda ninguna información que los diferencie.

Figura 21: *Registro de estado del módulo transmisor*

Descripción de los campos del registro:

bit 31-14 **Reserved**

bit 13-3 **DTX : Number Of Data Transmitted**

Indica el número de datos que ya han sido transmitidos hacia el sistema de adquisición.

bit 2 **TXIP : Transmission in progress**

Estará a nivel lógico alto cuando se haya comandado una transferencia y ésta aún no se haya completado, ni cancelado.

bit 1 **SERR: Slave Error bit**

Indica si se ha producido algún error en el módulo Slave-FIFO, del que se extraen los datos.

bit 0 **DONE: Transfer Completed**

Este bit se pondrá a uno cuando se haya completado con éxito la transferencia de todos los datos comandados.

- **TRANSFER REGISTER (ADDRESS 0x0001)**

El usuario realiza una escritura APB a este registro del módulo de transmisión para establecer el número de datos que desea que sean transmitidos y ordenar el inicio de la misma. En el supuesto de que quisiese cancelar la orden, también accedería a este mismo registro, pero en este caso establecería el Transfer Bit a '0'. A este registro puede acceder cualquier otro módulo, tanto para realizar una operación de lectura, como de escritura.

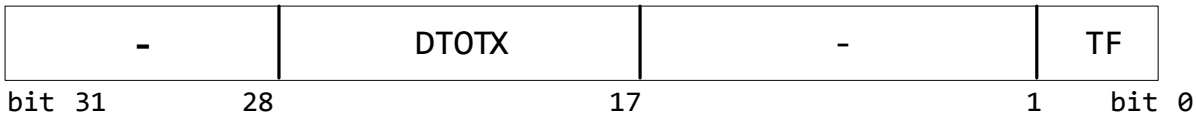


Figura 22: *Registro de configuración del módulo transmisor*

Descripción de los campos del registro:

bit 31-28 Reserved

bit 27-17 DTOTX

Determina el número de datos a leer de la FIFO y transmitir en paralelo.

bit 16-1 Reserved

bit 0 TF: Transfer bit

Este bit a '1' habilita la transmisión del número de datos especificados en DTOTX. En el supuesto de que se quisiese cancelar la transmisión, se debería configurar a '0'.

4.2.2.4. Interfaz de salida de datos en paralelo hacia el sistema de adquisición

La interfaz de transmisión de datos en paralelo hacia el sistema de adquisición receptor está formada por tres señales: CLK_OUT, DATA_OUT, ENABLETX.

Señal	Descripción
CLK_OUT	Señal de reloj saliente que proporciona la tasa a la que deben leerse los datos enviados.
DATA_OUT	Bus de 8 bits de salida de datos en paralelo.
ENABLETX	Señal que indica que se están transmitiendo datos útiles a capturar.

Tabla 6: *Listado descriptivo de la interfaz de salida de datos en paralelo*

El receptor obtendrá los datos proporcionados por el bus 'DATA_OUT' en el flanco de subida de la señal de reloj 'CLK_OUT' y siempre y cuando el nivel lógico de 'ENABLETX' sea alto.

Para conseguir que el flujo de datos hacia el sistema de adquisición cuando ha sido comandada una transferencia sea constante, no se produzcan pérdidas y el receptor siempre tenga un dato nuevo disponible para leer en el siguiente flanco de reloj, se ha diseñado la interfaz de salida de datos que se describe a continuación:

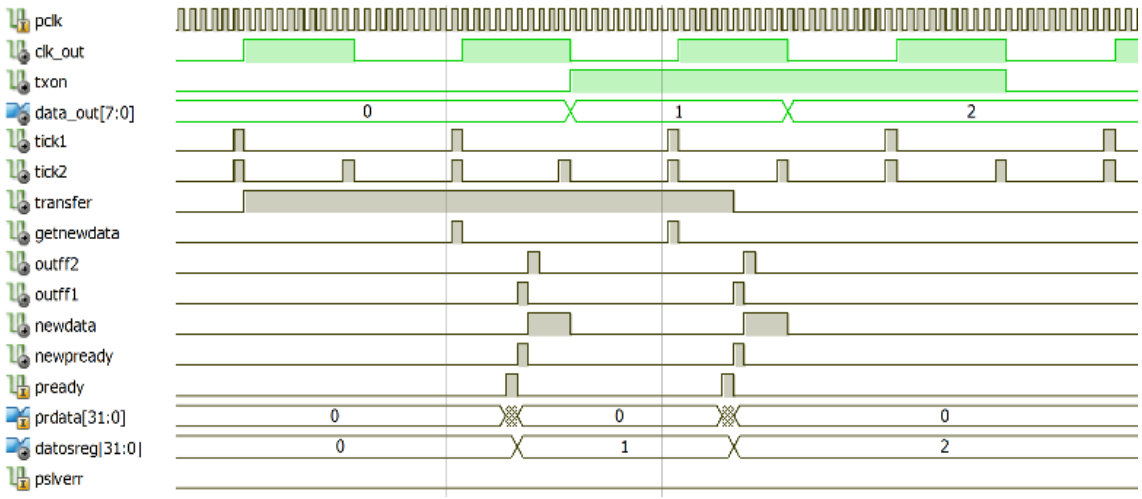


Figura 23: *Cronograma de la interfaz de salida de datos en paralelo*

Comportamiento:

Las señales 'tick1' y 'tick2' son generadas por sendos contadores a partir del reloj del sistema para establecer la velocidad de transferencia de los datos, en nuestro caso 160 Mbits/s y, además, son empleados como elemento de sincronismo. La señal 'tick2' posee el doble de frecuencia que 'tick1' y cada vez que pasa a valer uno, invierte la señal de salida 'CLK_OUT', generando de esta forma una señal de reloj.

Los datos a enviar al sistema receptor se actualizan cuando la señal 'tick1'se encuentra a nivel lógico bajo y la señal 'tick2'se encuentra en nivel alto, siempre que previamente se haya adquirido satisfactoriamente un dato de la FIFO. De esta forma, se garantiza que ante una posible variabilidad temporal en el envío de las señales, el receptor no capture datos erróneos; ya que éstos se actualizan en el flanco de bajada de la señal del reloj de salida. De esta manera, el receptor podrá capturar una señal estable en el flanco de subida de dicha señal de reloj, siempre y cuando 'ENABLETX' se encuentre a nivel lógico alto.

La señal de 'transfer' se emplea para controlar la adquisición de los datos y está activa cuando el bit que ordena la transmisión de datos del TRANSFER Register también lo está. Es decir, no se haya cancelado la transmisión, no exista error en el esclavo y aún existan datos a transmitir.

Para asegurarnos que existe un dato disponible a transmitir en el siguiente flanco ascendente del reloj, se emplea la señal 'getNewData'. Esta señal estará activa cuando ambas señales de 'tick' estén en nivel alto y también lo esté la señal de 'transfer'. Será usada por la máquina de estados del transmisor para dar inicio a la fase 'SETUP', iniciar la transferencia APB hacia el módulo Slave-FIFO y leer un nuevo dato de él. La transferencia APB completa dura seis ciclos de reloj, teniendo en cuenta todos los estados, por lo que dispondremos del dato a trasmitir con suficiente antelación antes de que tenga que ser enviado.

La señal 'ENABLETX' indica al receptor que en la interfaz de entrada procedente del transmisor existen datos a capturar en el flanco de subida del reloj. Esta señal por lo tanto, debe estar activa mientras que el transmisor esté enviando datos al sistema de adquisición.

En el caso de que el usuario decida cancelar la transmisión, el módulo transmisor no solicitará más datos al Slave-FIFO. Pero, no deshabilitará el envío de datos hasta que no haya transmitido todos los datos extraídos de la FIFO; evitando su pérdida. Por este motivo, es necesario que el transmisor sepa cuándo ha recibido un nuevo dato del Slave-FIFO. Esto se consigue gracias a la señal 'newData'. Esta señal

pasa a nivel lógico alto cuando el transmisor ha completado con éxito una nueva transferencia APB. Para ello, se ha empleado un detector de flanco, que se describe a continuación.

El detector de flanco positivo, en este caso, es un circuito que genera un pulso de duración igual a un ciclo de reloj cuando la señal cambia su estado de '0' a '1'. Existen varias formas de implementar esta funcionalidad, como por ejemplo mediante una máquina de estados. Sin embargo, en este caso se ha implementado empleando dos Flip-Flops tipo D y una puerta lógica AND con una de sus entradas negada.

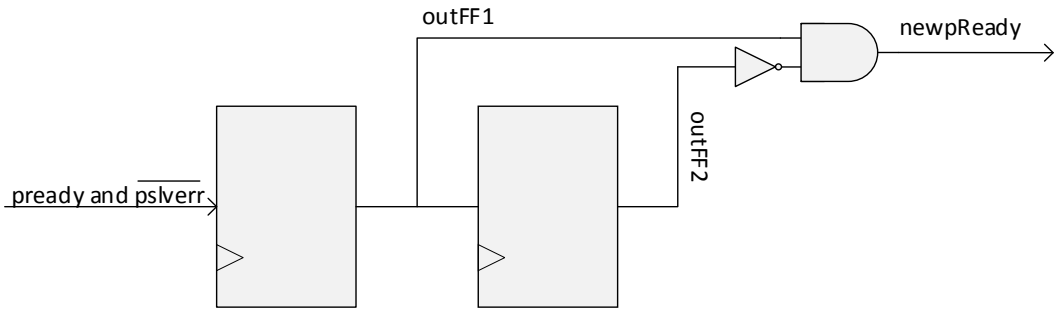


Figura 24: *Detector de flanco basado en dos FF tipo D, un inversor lógico y una puerta lógica AND de dos entradas*

Cuando el transmisor, en la última fase de la transferencia APB, recibe los datos adecuadamente, hace que se cumpla la condición de entrada del primer flip-flop. En el ciclo inmediato posterior, la salida de este elemento es uno. En ese preciso instante, antes de que transcurra ningún ciclo de reloj más, la condición que hace que la señal 'newpReady' pase a nivel lógico alto, se cumple. Cuando 'newPready' esté a nivel alto, la señal 'newData' también pasará a estarlo y se mantendrá en este estado hasta que cambie el dato a transmitir en la interfaz de salida.

De esta forma, se asegura que la señal que habilita la transmisión 'txON' se mantendrá activa hasta que se envíe el último dato que haya recibido el transmisor, aunque la señal de 'transfer' valga cero, condición que ocurre cuando se produce una cancelación.

El diagrama de bloques completo de la interfaz del módulo está recogido en la *figura 25* .

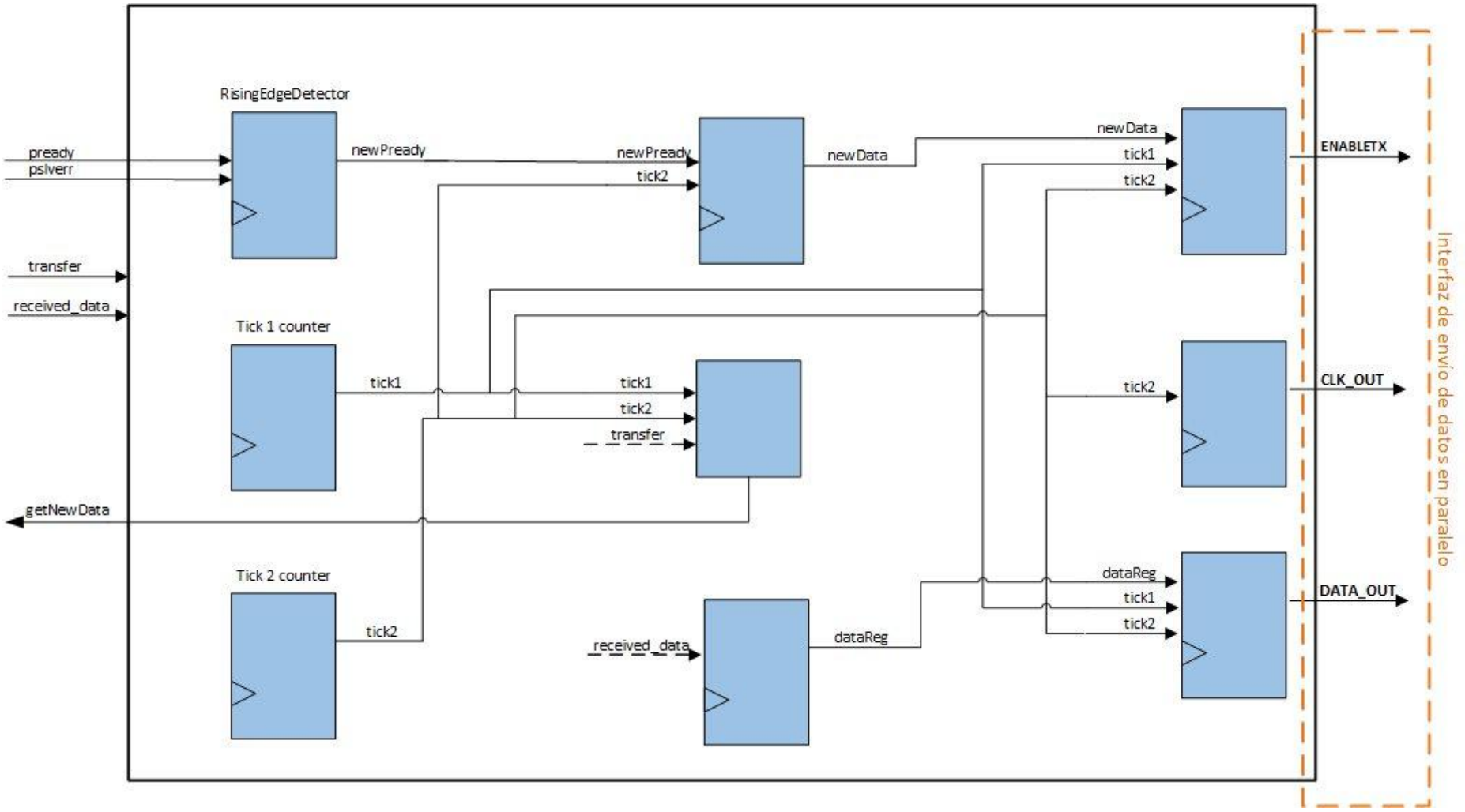


Figura 25: *Interfaz de salida de datos en paralelo del módulo TX*

4.2.3 Módulo Arbiter

El sistema está diseñado empleando un bus APB compartido con más de un maestro. Se puede dar el caso en el que varios maestros intenten acceder al bus simultáneamente, por lo que es necesario un elemento que gestione esta situación y decida a cuál de los módulos maestros solicitantes otorgar el acceso al bus. Esa es la razón de la necesidad de la creación e incorporación del módulo Arbiter al sistema.

4.2.3.1. Características

Este módulo posee tres interfaces esclavo APB, cada una de ellas procedente de los diferentes maestros APB del sistema y una interfaz maestro APB conectada con los dos módulos que poseen interfaces esclavo. Posee dos multiplexores que le permiten seleccionar las señales adecuadas, un registro y una máquina de estados cuya utilidad es replicar el protocolo APB en la interfaz maestro de salida. El diagrama de bloques completo en el que se aprecian todos los elementos del módulo está recogido en la *figura 27*.

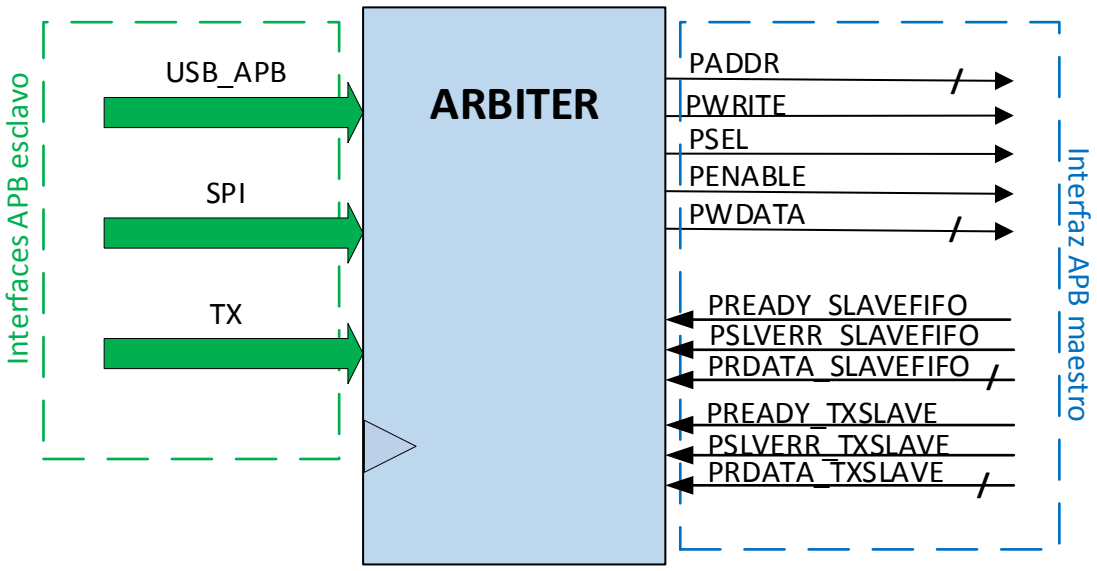


Figura 26: *Interfaces APB del módulo Arbiter*

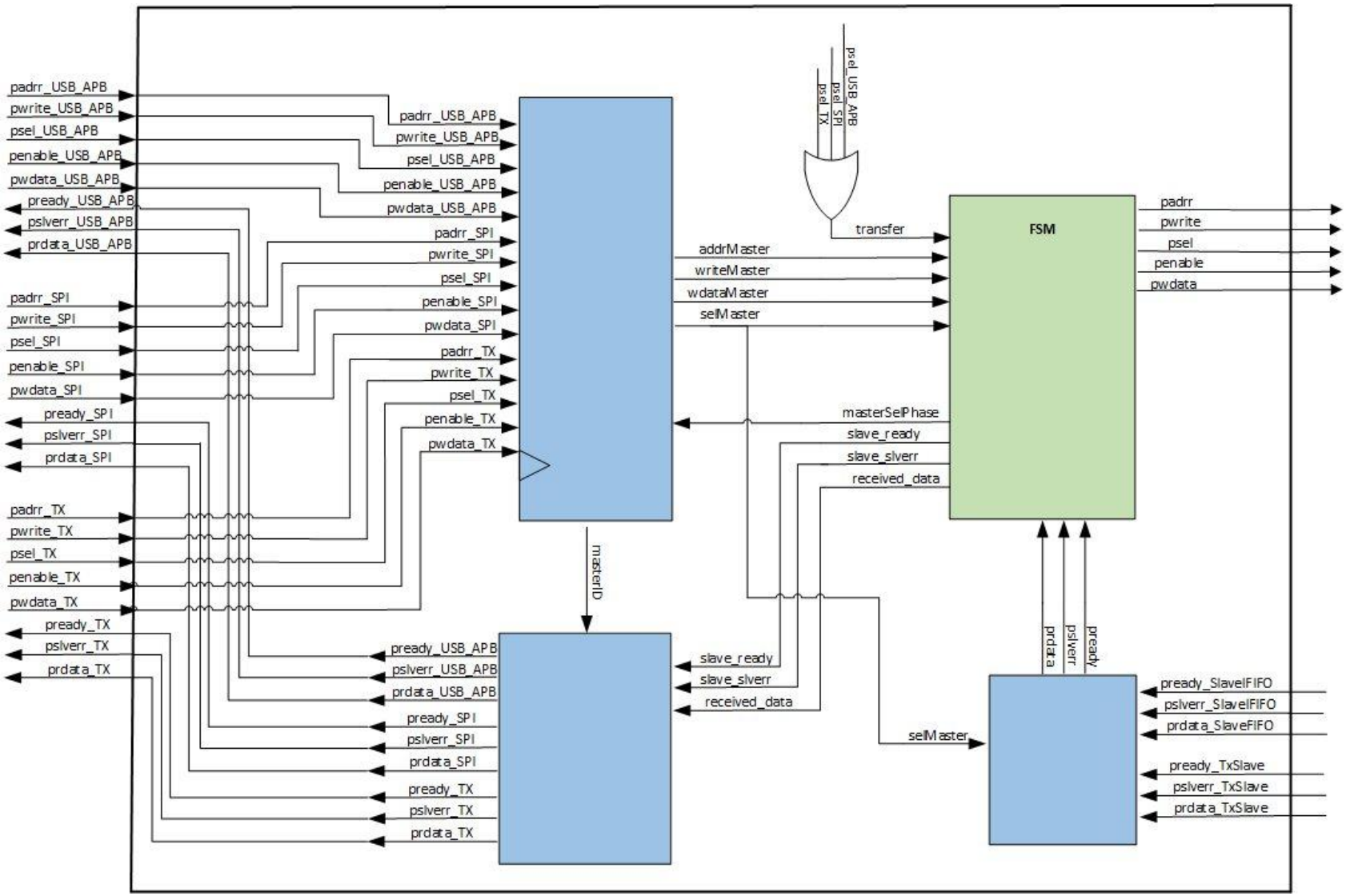


Figura 27: *Diagrama de bloques del módulo Arbiter*

4.2.3.2. Funcionalidad

Este módulo recibirá las solicitudes de los maestros del sistema y dependiendo de la prioridad de cada uno de ellos (*tabla 7*) dará acceso al bus a uno o a otro. De esa manera se garantiza que un único maestro tiene acceso al bus en un momento dado y que los demás a los que no se les ha concedido, deban esperar hasta que éste quede libre y el Arbiter permita su acceso.

Debido a la introducción de éste elemento síncrono en el módulo, todas las transferencias APB del sistema durarán un período de reloj adicional que no estaría presente si no se dispusiese de éste módulo y la comunicación APB fuese directa entre esclavo y maestro.

4.2.3.3. Algoritmo de arbitraje

En el sistema existen dos módulos maestros APB implementados, el USB-APB y el TX. Sin embargo, el sistema está diseñado para poder albergar otro módulo adicional más en una posible mejora futura, el módulo SPI. Por este motivo, se ha decidido incluirlo en el criterio de selección de acceso al bus.

Módulo solicitante	Prioridad
USB-APB	1
SPI	2
TX (interfaz APB maestro)	3

Tabla 7: *Prioridad de acceso al bus APB de los maestros del sistema*

El algoritmo de selección otorga el acceso al bus al maestro solicitante con mayor prioridad, cuando no esté ocupado por ningún otro. Ya que, una transferencia APB no podrá ser interrumpida, aunque en el momento en el que se esté llevando a cabo, un maestro con mayor prioridad intente acceder al bus. El Arbiter solamente permitirá el acceso a este nuevo maestro solicitante cuando haya concluido la transferencia por completo, es decir, el maestro que está ocupando el bus en ese momento haya recibido la señal de 'PREADY' por parte del esclavo comandado.

4.2.3.4. Máquina de estados finitos del módulo Arbiter

Para garantizar que los esclavos, comandados por parte de los maestros APB, reciben por su interfaz las señales del protocolo APB en los instantes temporales precisos correspondientes a la especificación, es necesario que el Arbiter disponga de una máquina de estados que le permita reproducir el protocolo en su interfaz de salida.

A continuación se representa el diagrama de estados simplificado, en el que únicamente aparecen las transiciones entre estados y la condición necesaria para ello.

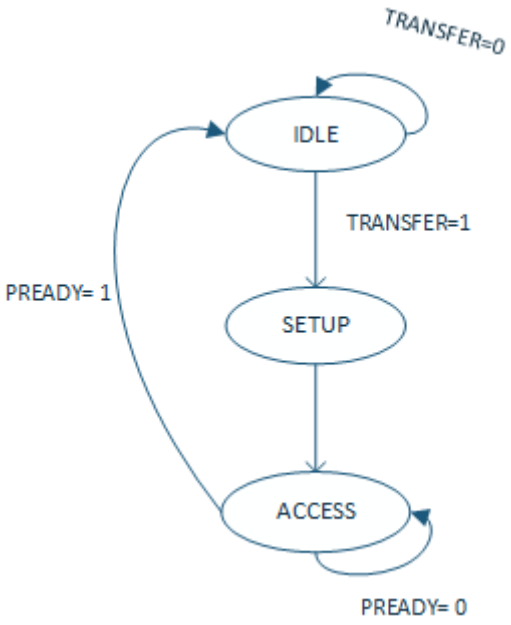


Figura 28: *Diagrama simplificado de estados del módulo Arbiter*

La máquina de estados finitos operará entre los siguientes estados:

- **IDLE**
  Este es el estado inicial por defecto de la máquina de estados. La selección del maestro que puede acceder al bus se lleva a cabo exclusivamente en este estado.

- **SETUP**
  Cuando se cumple la condición de 'TRANSFER', es decir, alguno de los módulos maestros del sistema desean tomar el bus, la máquina de estados cambia al

estado de SETUP. En este estado, las señales salientes de la interfaz maestro APB del Arbiter adquieren los valores de las señales APB del maestro seleccionado para tomar el bus, según el algoritmo de arbitraje. El módulo sólo se mantiene en el estado de SETUP por un ciclo de reloj y siempre cambia al estado de ACCESS en el siguiente flanco de reloj ascendente.

- **ACCESS**

  En este estado se devuelve la señal de PREADY, además de los valores correspondientes para PSLVERR y PRDATA, dependiendo de si ha sucedido un error en la transferencia, o no.

La máquina de estados del módulo Arbiter, además de reproducir el protocolo APB en su interfaz de salida maestro APB, garantiza que las transferencias APB no sean interrumpidas, haciendo uso de la señal 'masterSelPhase'. En el diagrama ASM de la *figura29* se puede apreciar esta funcionalidad con más detalle. La señal 'masterSelPhase' únicamente se encontrará a nivel lógico alto cuando la máquina de estados de encuentre en el estado de IDLE y, por lo tanto, no se haya iniciado aún alguna transferencia APB. Esta señal es empleada por el módulo para actualizar el conjunto de valores de las señales APB procedentes del maestro seleccionado. En el diagrama ASM este conjunto ha sido representado como 'masterValues'. 'APB Outgoing' hace referencia a las señales de la interfaz APB de salida del módulo Arbiter, exceptuando la señal de 'PENABLE'; ya que esta última toma valor gracias a la máquina de estados para que la reproducción del protocolo en la interfaz de salida sea la adecuada.

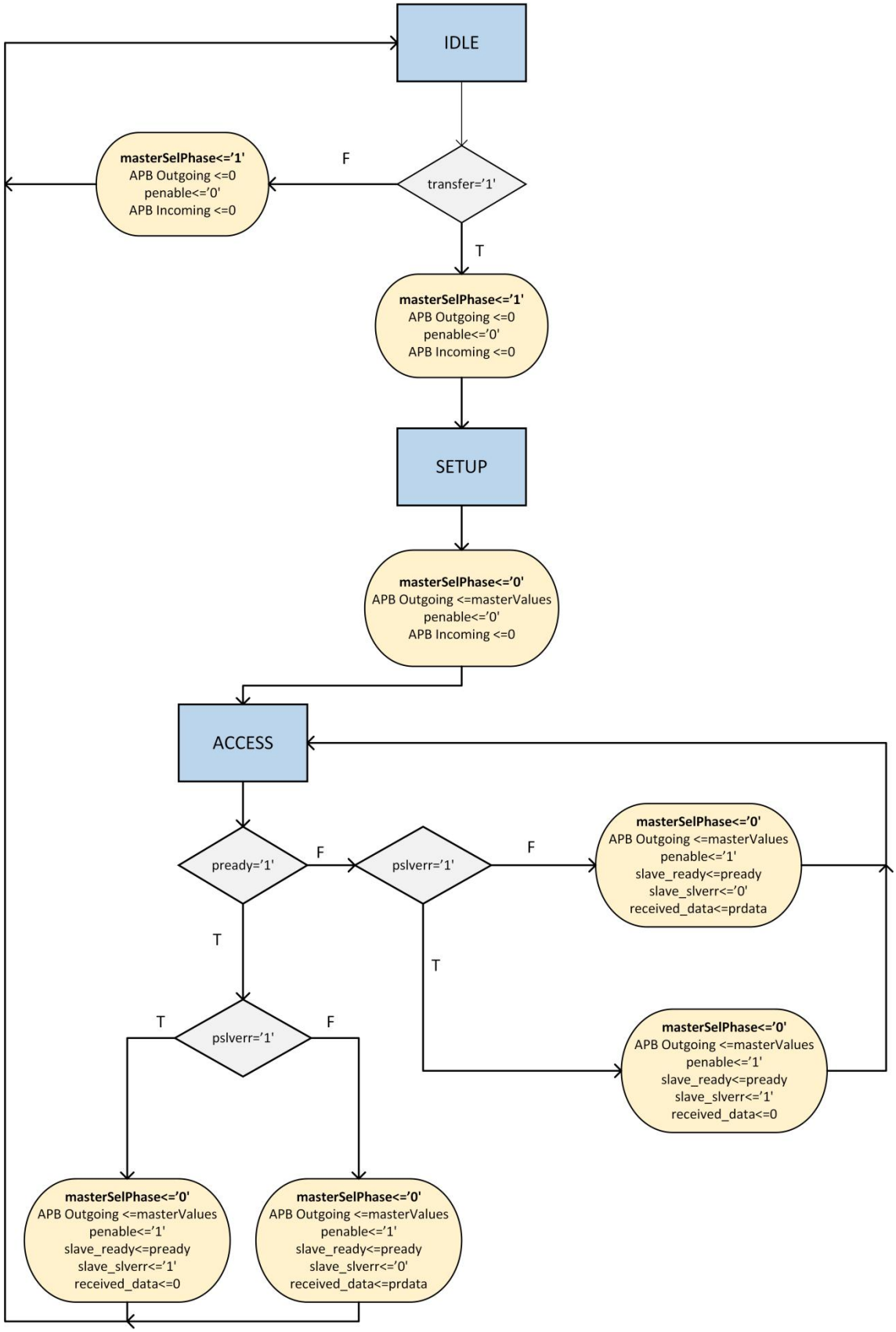


Figura 29: *Diagrama ASM del módulo Arbiter*

Capítulo 5

Verificación del sistema

El proceso de verificación es el primero de la etapa de evaluación del producto, una vez que éste ha sido implementado e integrado.

La verificación tiene como objeto probar que el sistema desarrollado cumple con los requisitos establecidos y las especificaciones técnicas; como por ejemplo, con el protocolo de comunicaciones AMBA APB.

Está basado en un análisis exhaustivo de todo el sistema y se ha realizado empleando el simulador de lenguaje de descripción de hardware 'ISim' integrado dentro de 'ISE'.

5.1. Verificación individual de los módulos del sistema

En los apartados consecutivos, se describirán las condiciones reproducidas y su resultado para la completa verificación de cada módulo. También se referenciarán las capturas del simulador que constatan éstos resultados.

5.1.1. Verificación del módulo Slave-FIFO

1. Comprobación del estado inicial.

Al inicio, se puede apreciar en la *figura 30* cómo la señal 'Empty' procedente del componente FIFO está activa hasta que no finaliza la transferencia APB. Debido a esto, el registro de estado contendrá el código de excepción correspondiente a esta situación hasta que se produzca el cambio.

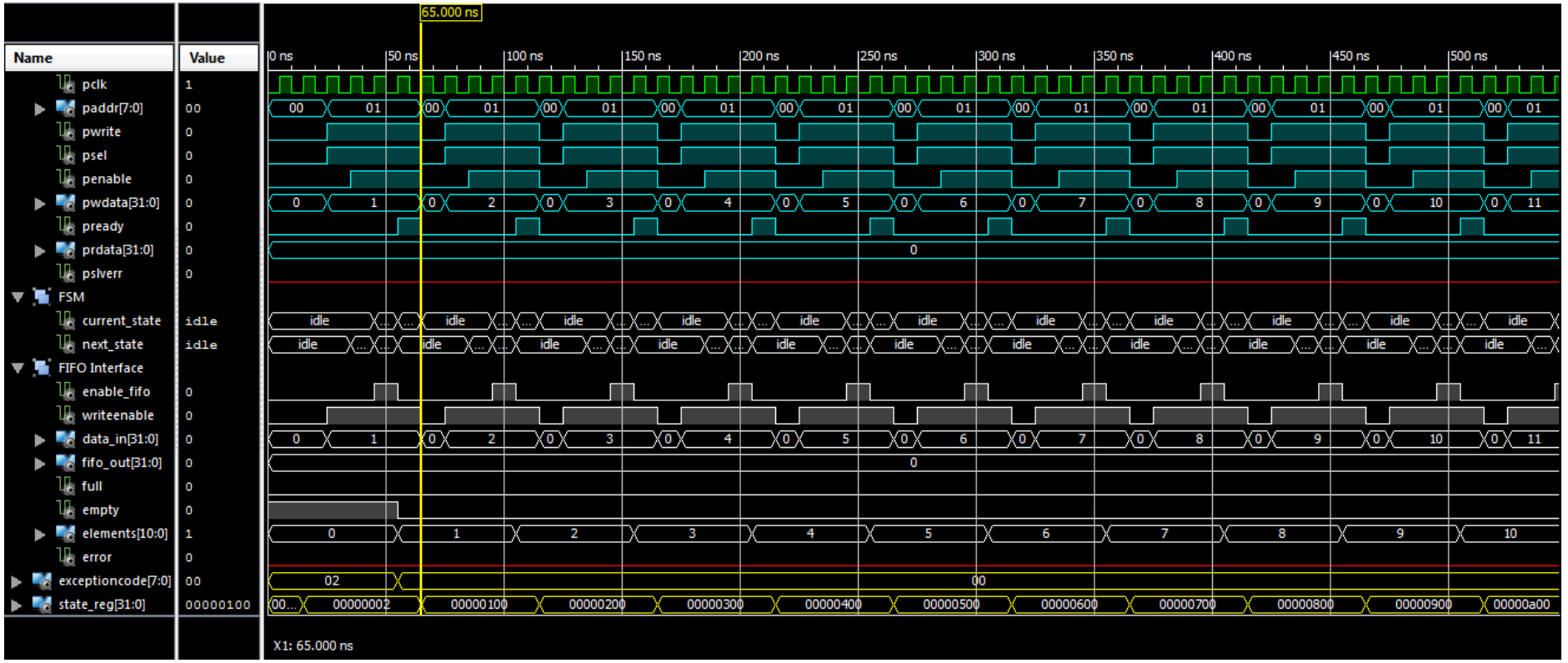


Figura 30: *Captura del resultado de la verificación del módulo Slave-FIFO de los puntos 1y 2 (parte I)*

2. Llenado del búfer FIFO con una secuencia de números consecutivos.

Se escribe en el búfer FIFO una secuencia de números consecutivos del 1 al 1024, hasta ocupar la totalidad del mismo. La máquina de estados controla que el acceso al componente sea el adecuado y ocasiona que la transferencia APB sea realizada con estados de espera, durando ésta un total de cinco ciclos de reloj *(figura 30)*. El índice de escritura que controla el acceso a las diferentes posiciones de memoria del búfer FIFO se actualiza adecuadamente, al igual que el contenido de la memoria *(figura 31).*

Después de que se haya introducido el último dato en el búfer, la señal 'Full' se activa provocando la actualización del código de excepción y del registro de estado *(figura 32).*

3. Intento de escritura en el búfer FIFO lleno.

Al intentar escribir en el búfer FIFO, encontrándose éste lleno, el componente activa la señal de error. Esta señal se mantendrá a nivel alto hasta que se realice un acceso correcto al mismo; en este caso en concreto, hasta que se lea algún dato de él. Al haberse producido un error, la señal saliente 'pslverr' de la interfaz APB se mantiene en alto durante la duración de la última fase del protocolo *(figura 32).*

El código de excepción y el registro de estado actualizan su valor de la forma adecuada y el búfer y los valores de los punteros de acceso mantienen su integridad.

4. Lectura del registro de estado.

Como se puede observar en la *figura 32* la duración de la transferencia APB cuando se comanda la lectura del registro de estado es la misma que cuando se accede al búfer FIFO, por lo que se cumple el requisito de diseño de que el acceso al componente sea determinista. La lectura del registro es la adecuada. El código de excepción se mantiene sin variación hasta que se produzca un acceso apropiado al búfer, o se produzca una situación anómala de otra índole.

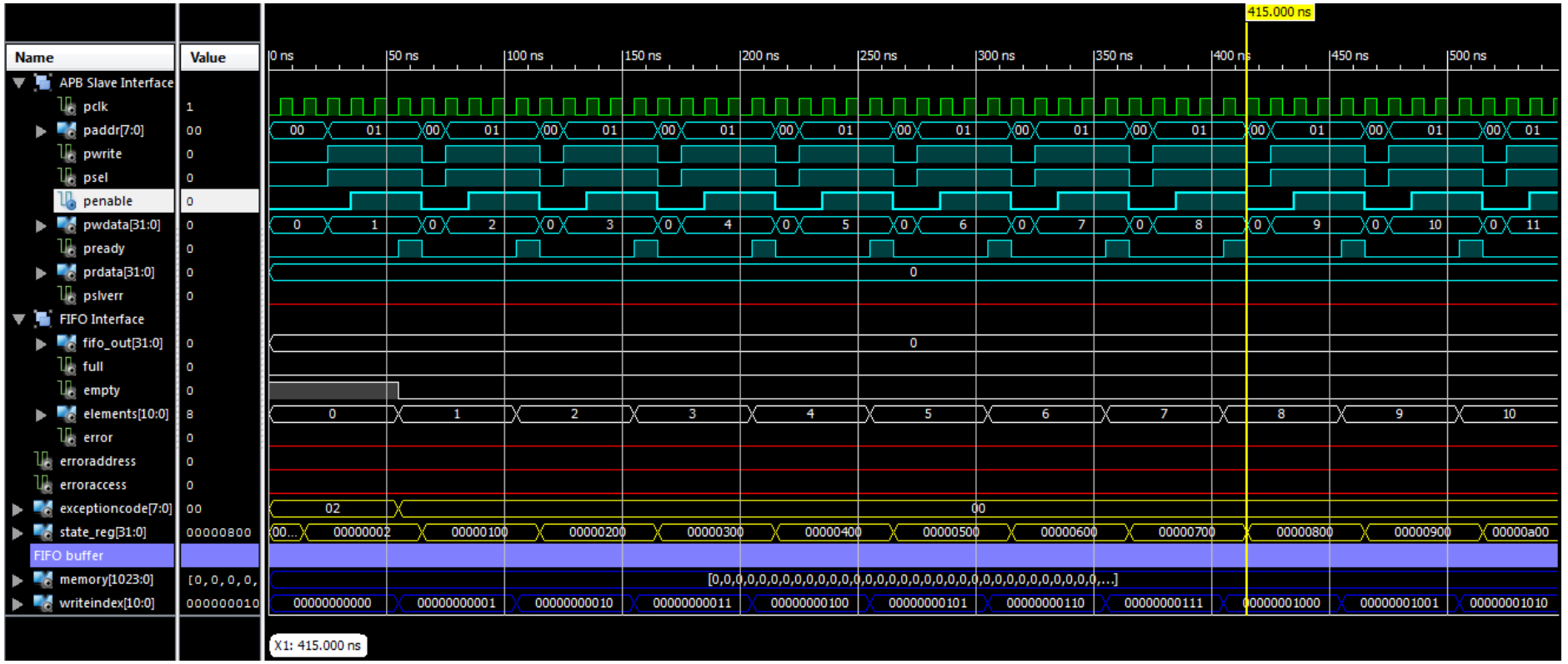


Figura 31: *Captura del resultado de la verificación del módulo Slave-FIFO de los puntos 1 y 2 (parte II)*

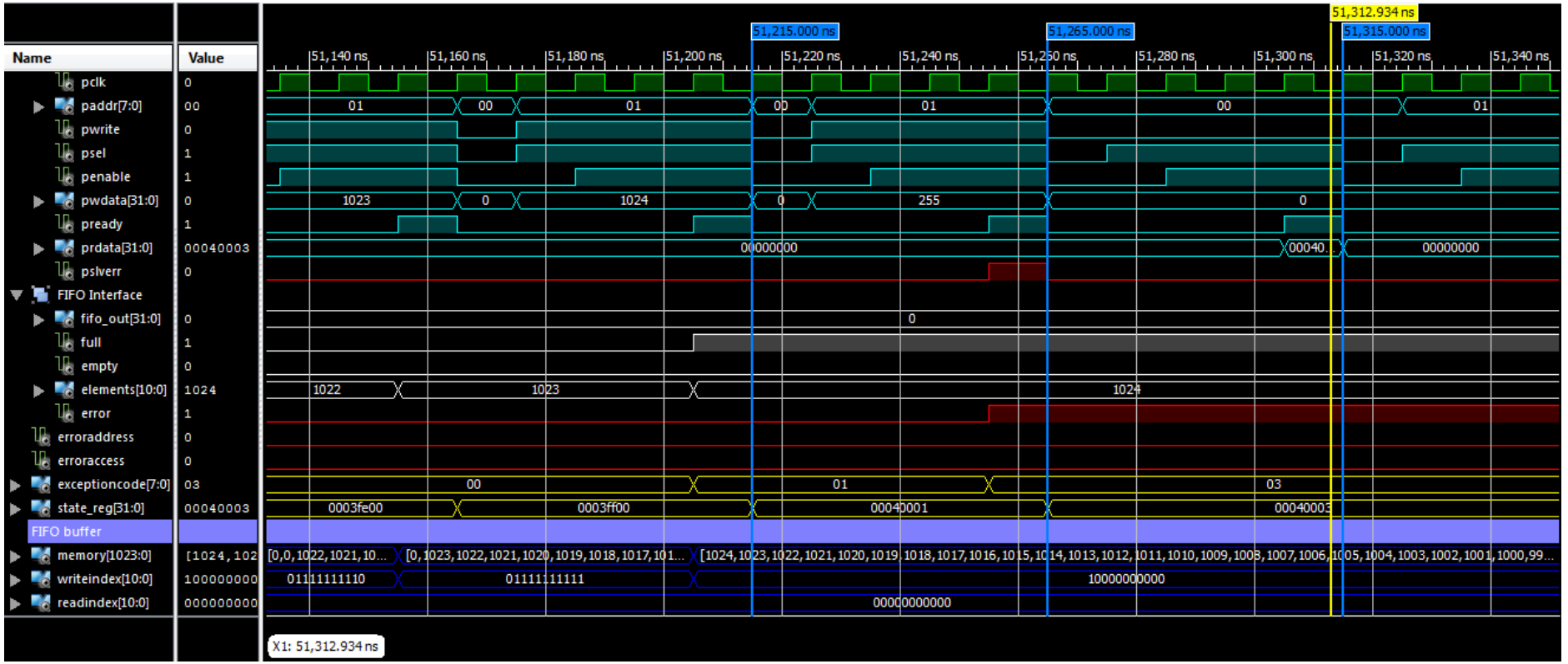


Figura 32: *Captura del resultado de la verificación del módulo Slave-FIFO de los puntos 2, 3 y 4*

5. Lectura de todos los datos del búfer FIFO.

Al obtener el primer dato de la FIFO, tanto la señal de 'Full', como la de 'error' se ponen a cero y el registro de estado se actualiza. Los demás son leídos correctamente y el valor de los punteros de acceso se va actualizando con cada transferencia APB.

Posteriormente a la obtención del último dato disponible en la FIFO, y como se puede apreciar en la *figura 33*, la señal de 'Empty' se activa y el código de excepción y el registro de estado se actualizan.

6. Intento de lectura del búfer FIFO vacío.

Al intentar leer en la FIFO vacía, se activa la señal de 'error' del componente. Como consecuencia, la señal 'pslverr' también se activará en el último ciclo de la transferencia APB.

El código de excepción y el registro de estado actualizan su valor de la forma correcta y los valores de los punteros mantienen su integridad *(figura 33).*

7. Lectura del registro de estado.

La lectura del registro de estado es la apropiada, como se aprecia en la *figura 33,* al igual sucedía en el punto 4, el código de excepción se mantiene sin variación hasta que se produzca un acceso apropiado al búfer, o se produzca una situación anómala de otra índole.

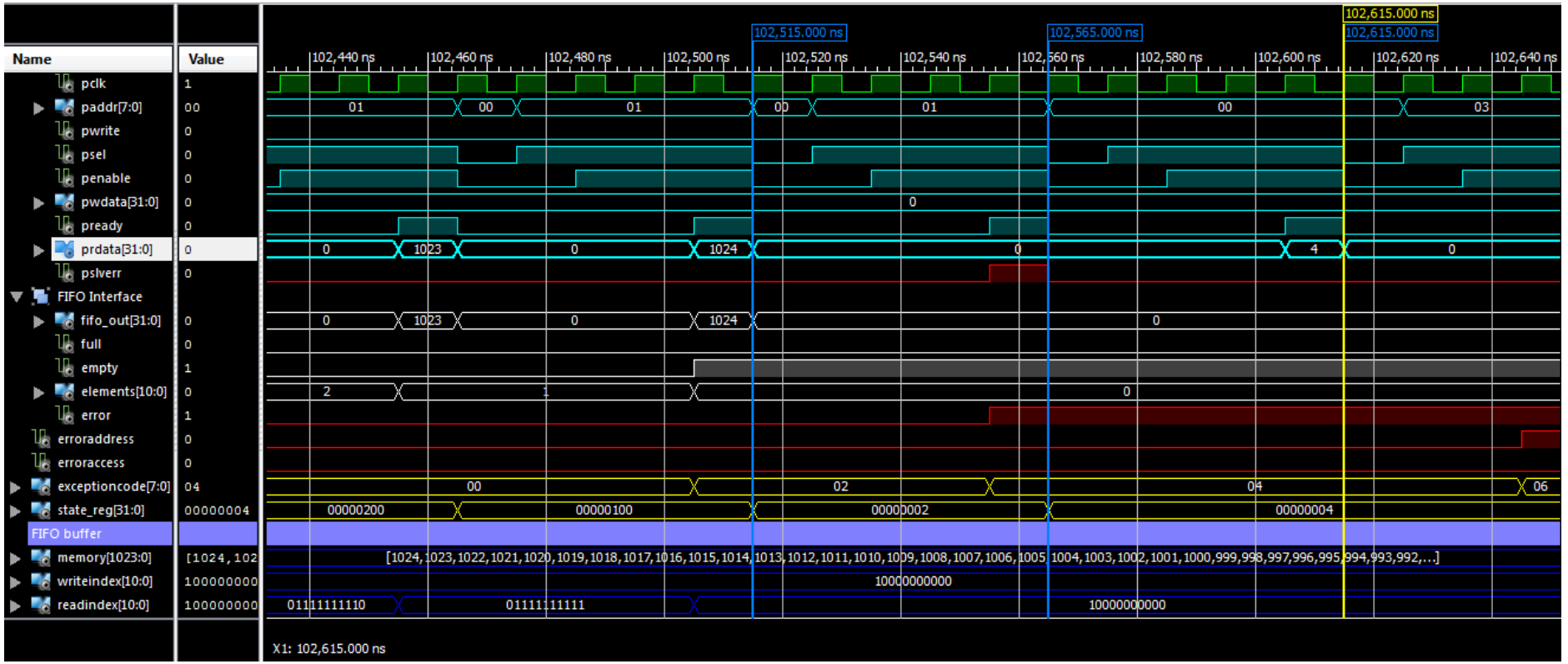


Figura 33: *Captura del resultado de la verificación del módulo Slave-FIFO de los puntos 5, 6 y 7*

8. Intento de lectura de una dirección inexistente.

Como se observa en la *figura 34*, al puerto de entrada 'paddr' se le da un valor que no tiene correspondencia con ningún elemento del módulo. Esto provoca que la señal que avisa de este acceso inadecuado 'erroraddress' se ponga en alto hasta que se produzca un acceso adecuado al registro y tanto el código de excepción, como el registro de estado de actualicen. Además el usuario es informado de este error, activándose la señal de 'pslverr' de la interfaz APB.

9. Lectura del registro de estado.

Se obtiene por el puerto de salida 'prdata' el número de elementos del búfer y el código de excepción, que en este caso en concreto corresponde al 0x06. Como la lectura es correcta, el código de excepción se actualiza a 0x04, valor anterior que indicaba que se ha intentado leer de una FIFO vacía y que hasta que no se produzca un acceso correcto a ella, no *variará (figura 34).*

10. Intento de escritura en registro de estado.

La escritura del registro de estado únicamente está permitida para el propio Slave-FIFO. Cualquier tentativa de hacerlo por otro módulo del sistema, provoca que la señal 'erroraccess' se active y, al hacerlo, también lo esté en el último ciclo de la transferencia APB la señal de 'pslverr'. La integridad del registro de estado se mantiene y el código de excepción y el registro de estado se actualizan correctamente *(figura 34).*

11. Lectura del registro de estado.

Al realizar un acceso adecuado al registro de estado, éste se actualiza adecuadamente.

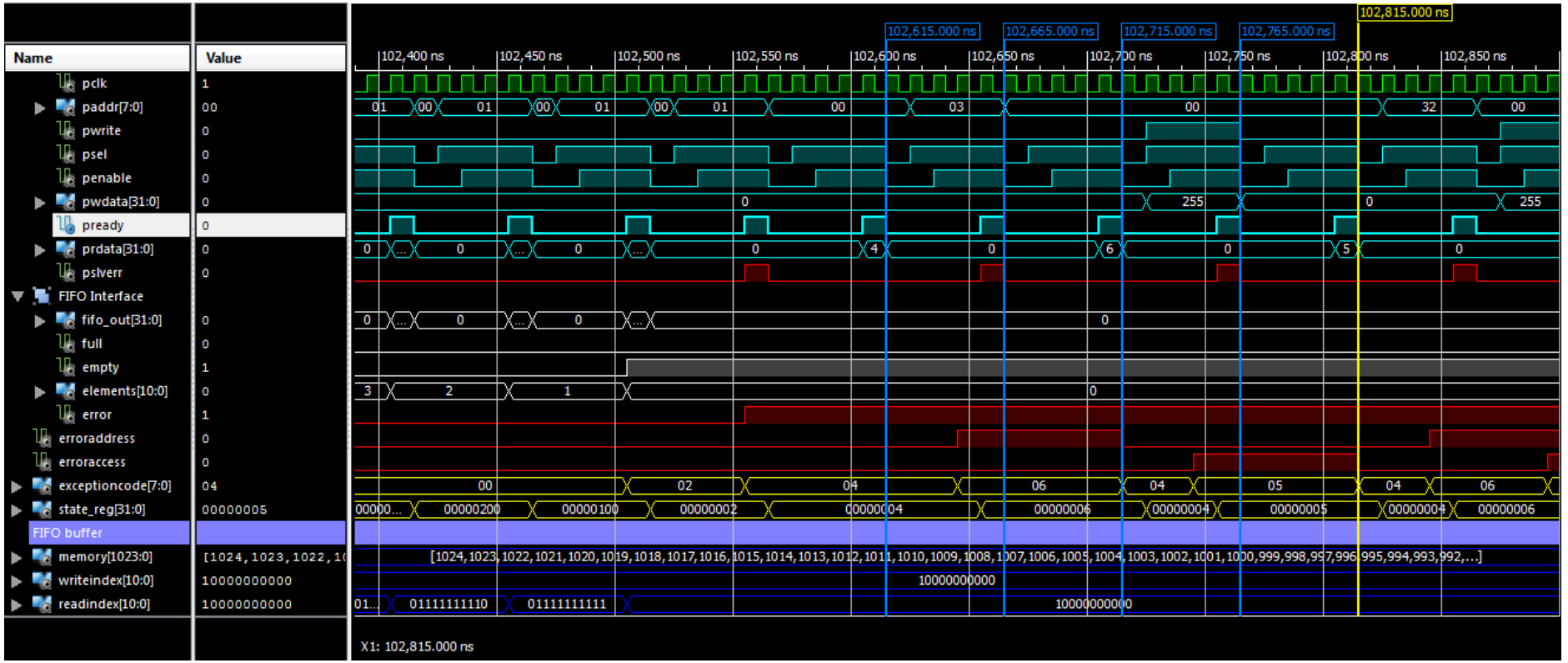


Figura 34: *Captura del resultado de la verificación del módulo Slave-FIFO de los puntos 8, 9, 10 y 11*

12. Intento de lectura de una dirección inexistente, intento de escritura en el registro de estado y lectura del registro de estado.

Estas tres acciones son realizadas en sendas transferencias APB consecutivas como se puede apreciar en la *figura 35.* El inicio de la primera se corresponde con el marcador situado en el instante temporal de la simulación de 102815 ns.

Se aprecia cómo el valor del código de excepción del registro de estado cambia de 0x04, que corresponde a un intento de lectura de la FIFO vacía, a 0x06, indicando el intento de acceso a una dirección no contemplada. Además por tratarse de una situación anómala las señales de 'erroraddress' y 'pslverr' se activarían de igual forma como sucedía en el punto 8. La secuencia esperada es que el usuario al recibir la señal de error, inmediatamente después realice una lectura al registro de estado para ver de qué error se trata. Sin embargo, no se ha presupuesto que esto suceda así y puede darse el caso en el que se comandasen varias transferencias erróneas seguidas al módulo. El criterio establecido es que prevalecerá el código de la que haya sido efectuada más recientemente. Por ejemplo, en este caso, cuando se procede a la lectura del registro de estado en el instante temporal 102915 ns *(figura 35)* de la simulación, el valor del código de excepción obtenido es 0x05, indicando cuál ha sido el último acceso irregular.

13. Intento de escritura en el registro de estado, intento de lectura de una dirección inexistente, introducción de un dato en la FIFO y lectura del registro de estado.

De forma análoga a como sucede en el punto anterior, éste sirve para evaluar qué sucede si se accede a diferentes partes del módulo, ya sea de forma incorrecta, como correctamente.

En el instante 102965 ns *(figura 35)* de la simulación se comanda la primera orden de la secuencia. Se puede apreciar como el criterio establecido de prevalencia de los errores se mantiene como en el punto anterior. En el instante 103065 ns, se comanda la orden de escritura de un dato en la FIFO vacía, se observa cómo es llevada a cabo con normalidad, el código del registro de estado pasa a ser 0x00, indicando normalidad y el número de elementos se actualiza a 1.

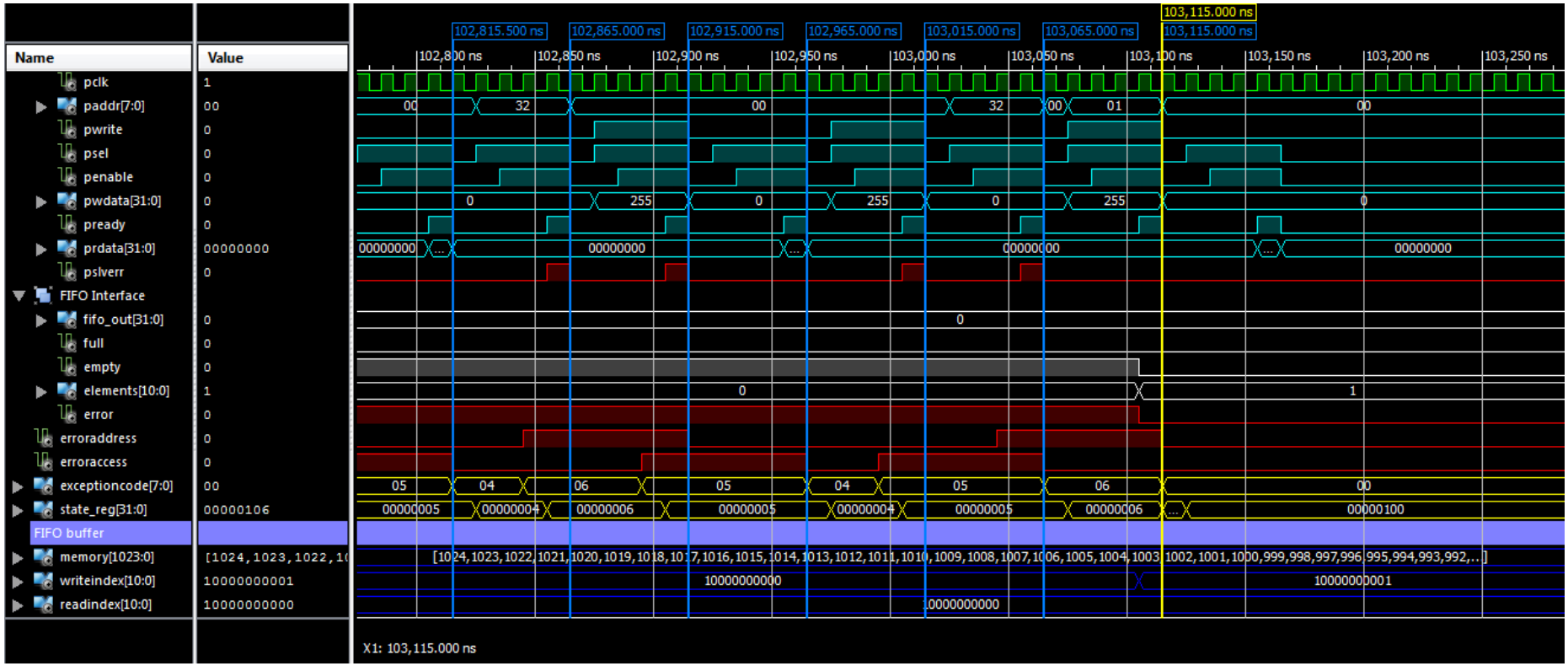


Figura 35: *Captura del resultado de la verificación del módulo Slave-FIFO de los puntos 12 y 13*

5.1.2. Verificación del módulo transmisor

Para poder evaluar este módulo por completo, es preferible hacerlo en conjunto con el módulo Slave-FIFO y una vez que se haya integrado el Arbiter al sistema. Pues de esta forma, se podrá evaluar cómo cambia el valor del registro de estado y de los contadores de números de elementos transmitidos. Aun así, se ha realizado una verificación más sencilla para comprobar que el acceso a los registros es el adecuado.

1. Comprobación del estado inicial

El comportamiento es el esperado. Inicialmente el valor de los registros y de todas las señales es cero.

2. Acceso al registro de transferencia

Se comanda la lectura y transmisión en paralelo de diez datos. Para ello, se modifica el registro de estado de forma adecuada, indicando el número de datos a transmitir y poniendo a '1' el bit de 'Transfer' que habilita la operación.

3. Lectura del registro de estado

Tras comandar la operación, se obtiene el valor del registro de estado por el puerto de salida 'prdata' de la interfaz esclavo. En este caso, únicamente con el bit TXIP activo que indica que al módulo transmisor se le ha ordenado el envío de unos datos, pero aún no ha finalizado el mismo.

4. Cancelación del envío de datos.

De manera similar al punto 1, se accede al registro transmisor; pero esta vez, el bit de 'Transfer' es configurado a '0'. En el instante temporal 105 ns de la simulación, recogida en la *figura 36,* se puede apreciar cómo varía el contenido del registro de transferencia. Como consecuencia de esto, el registro de estado también cambia; ya que, no existe ningún envío de datos en activo, por lo que el bit de TXIP pasa a valer '0'.

5. Intento de acceso a una dirección inexistente del módulo

Al intentar realizar una operación de escritura en una dirección diferente a las contempladas, la señal de salida 'pslverr' de la interfaz esclavo se activa, avisando de este error.

6. Lectura del registro de estado

Se obtiene el valor del registro de estado por el puerto 'prdata', que en este caso es igual a cero. A diferencia del módulo Slave-FIFO, el módulo TX no guarda información alguna que identifique el tipo de error, pero si avisa de él en el momento en el que se produce. Por este motivo, al leer el registro de estado no aparece alusión alguna a la operación anterior. En cambio, como ya se comprobará más adelante en la simulación global del sistema, si se produjese un error en el módulo Slave-FIFO cuando el transmisor está accediendo a él, éste sí aparecería.

7. Intento de escritura en el registro de estado

Esta prueba corresponde con la última transferencia APB de la *figura 36.* De forma semejante a lo sucedido en el punto 4, el transmisor activa la señal APB 'pslverr' de la interfaz esclavo para advertir al maestro la escritura de una dirección errónea.

Todos los puntos anteriores han sido simulados consecutivamente, es decir, sin esperas entre una orden APB y otra. La transferencia APB es llevada a cabo sin el uso de estados de espera, tardando en total tres ciclos de reloj. La *figura 36* recoge la totalidad de las pruebas realizadas.

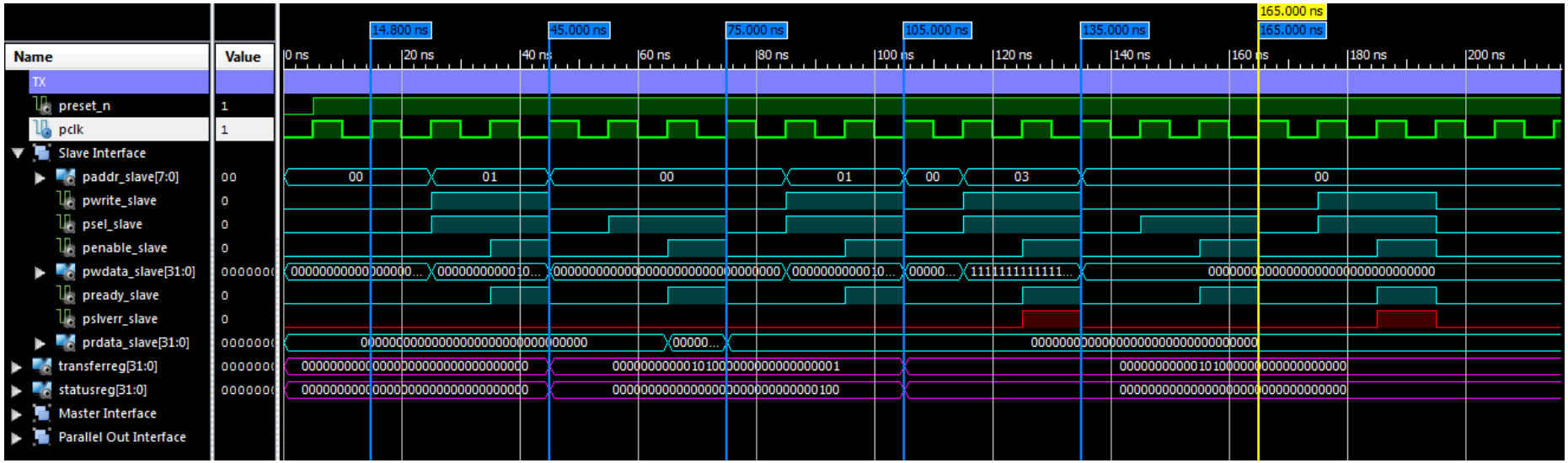


Figura 36: *Captura del resultado de la verificación del módulo TX*

5.1.3. Verificación del módulo Arbiter

Al igual que ocurría con el módulo transmisor, para verificar el correcto funcionamiento de este módulo es preferible hacerlo una vez esté integrado en el sistema para poder apreciar mejor el intercambio de señales entre los maestros y esclavos. Por este motivo, en los siguientes puntos únicamente se evaluará si el algoritmo de arbitraje que determina la prioridad entre los diferentes maestros funciona correctamente y se cumple la premisa de que una transferencia APB, una vez iniciada, no puede ser interrumpida.

Aunque el módulo SPI no está integrado en el sistema, el módulo Arbiter sí dispone de la lógica necesaria para gestionarlo como un maestro más. Por este motivo, se ha decidido evaluar junto con los otros dos maestros APB del sistema, a vistas de una posible ampliación del mismo.

1. Comprobación del estado inicial.

En el estado inicial la señal 'masterSelPhase' tiene un valor de '1', lo cual indica que la máquina de estados se encuentra en el estado de 'IDLE'; por lo que no hay ninguna transferencia APB en curso y puede seleccionarse un nuevo maestro para que acceda al bus. El valor del resto de señales es el esperado a cero.

2. Todos los maestros del sistema intentan comunicarse con el mismo esclavo al mismo tiempo.

Como se puede observar en las *figuras 37 y 38* en el instante temporal de 15 ns, todos los módulos del sistema intentan acceder simultáneamente al módulo Slave-FIFO, teniendo lugar Las fases APB de SETUP y ACCESS en los instantes temporales de 25 y 35 ns, respectivamente. Al encontrarse el módulo Arbiter en el estado de 'IDLE' y ser posible la elección de un nuevo maestro, el de mayor prioridad de los solicitantes es el que finalmente accede al bus. En este caso, el USB-APB, como se puede comprobar por el código de 'masterID'. Se comprueba que la reproducción del protocolo APB para los valores del maestro seleccionado es la adecuada.

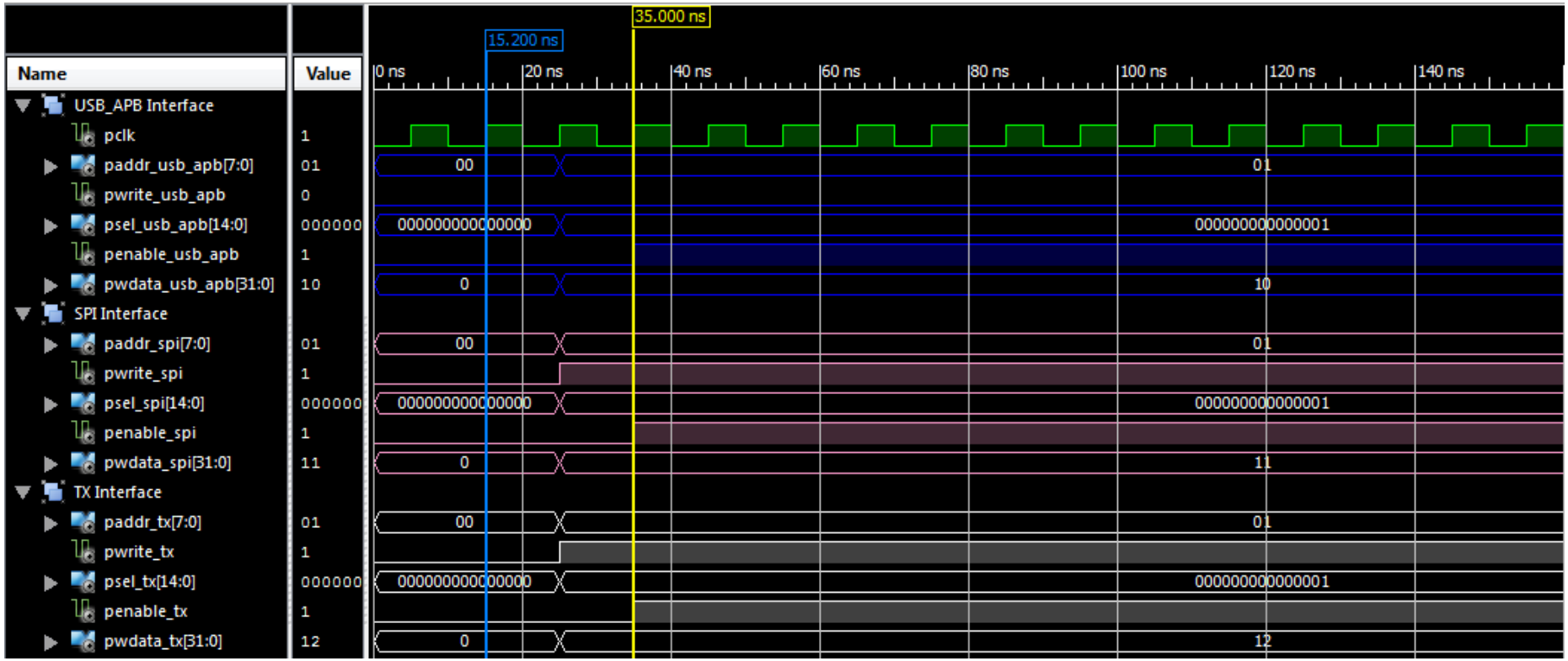


Figura 37: *Captura del resultado de la verificación del módulo Arbiter de los puntos 1 y 2 (parte I)*

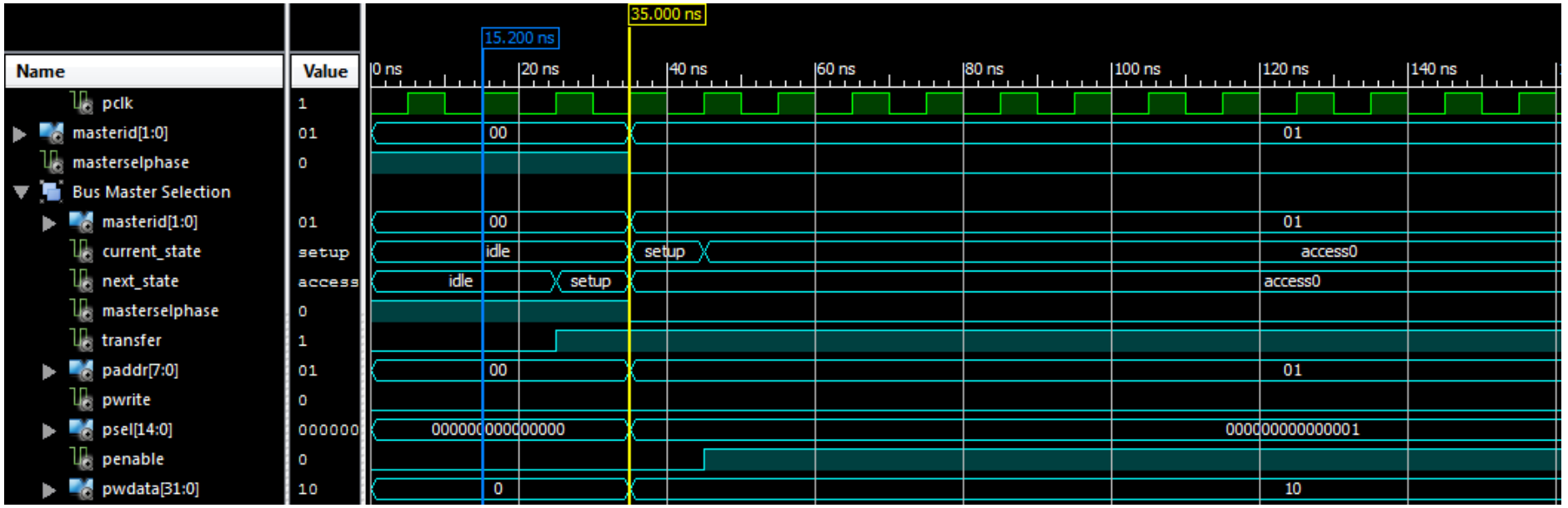


Figura 38: *Captura del resultado de la verificación del módulo Arbiter de los puntos 1 y 2(parte II)*

3. Intento de acceso al bus de forma simultánea por dos maestros para comunicarse con el mismo módulo esclavo.

En este caso el módulo USB-APB y el módulo TX intentan comunicarse con la interfaz esclavo del módulo de TX[1]. Observando la señal 'masterID' *(figura 39)*, se puede apreciar cómo el maestro seleccionado que accede al bus es el de mayor prioridad, identificado con el id 1. Además también se puede ver cómo el protocolo APB es reproducido adecuadamente, dando los valores del maestro a las señales; como, por ejemplo, a la señal 'pwdata' o 'psel'.

4. Intento de interrupción de una transferencia APB en curso.

Estando la transferencia del punto anterior sin finalizar, se intenta comandar una nueva transferencia APB por parte del módulo SPI. Pero una vez que la señal 'masterID' cambia su valor al correspondiente para el módulo USB-APB en el instante temporal de 35 ns , ya no vuelve a cambiar en toda la simulación. Esto se debe a que la transferencia APB no ha finalizado[2]. Lo mismo sucede con la señal 'masterSelPhase'. *(figura 39)*

[1] Como ya se mencionó con anterioridad, el módulo transmisor es imposible que pueda comandarse a sí mismo o a cualquier otro que no sea el módulo Slave-FIFO. Sin embargo, se ha decidido simular esta situación por el mero interés de escoger otro esclavo que no fuese el seleccionado en el punto 1 y evaluar si las señales correspondientes a la selección de esclavo se actualizaban correctamente.

[2] El comportamiento normal es que la transferencia APB finalice transcurridos unos ciclos de reloj, cuando haya recibido la respuesta por parte del esclavo. Sin embargo, como se mencionó al inicio de esta sección, para apreciar este comportamiento es preferible que el módulo esté integrado en el sistema.

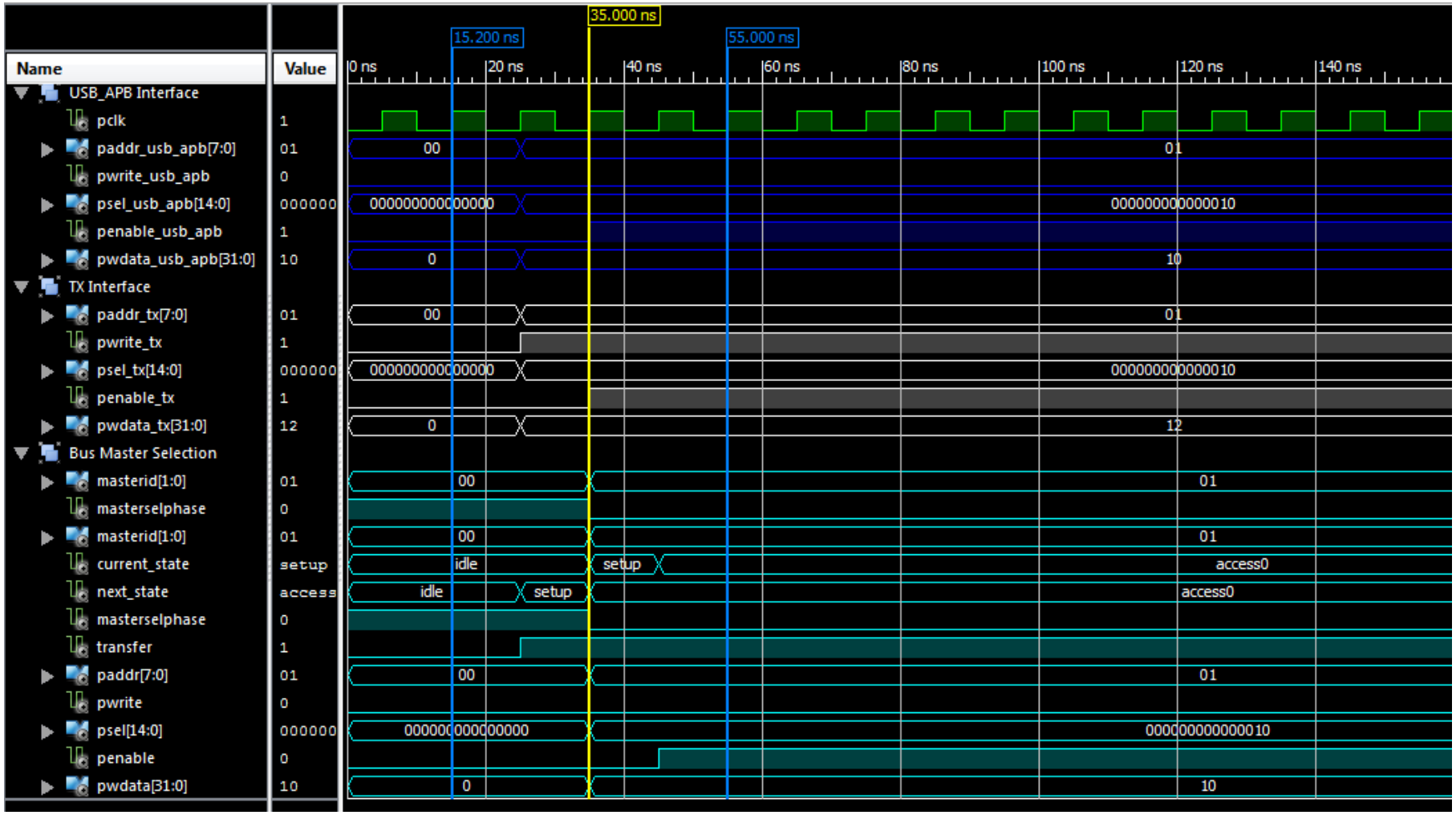


Figura 39: *Captura del resultado de la verificación del módulo Arbiter de los puntos 3 y 4 .*

5.2. Verificación global del sistema

1. Comprobación estado inicial

Se puede apreciar en la *figura 40* cómo el estado inicial es el adecuado. Como ya se constató en la evaluación individual del módulo Slave-FIFO, al encontrarse vacío el búfer, la señal de 'Empty' estará a nivel lógico alto y el registro de estado contendrá el código de excepción correspondiente. Además, el módulo Arbiter se encuentra en la fase de IDLE, la señal 'masterSelPhase' está a nivel lógico alto, lo que permite que un nuevo maestro tome el bus.

2. USB-APB escribe un dato en la FIFO.

En el instante temporal de 15 ns se inicia esta transferencia APB. Se puede apreciar cómo hasta el instante de 75 ns ésta no finaliza, durando un total de 60 ns. Como el período del reloj del sistema es de 10 ns, esto implica que la transferencia APB dure un total de 6 ciclos de reloj en completarse. Si recordamos los resultados obtenidos cuando se simuló el módulo Slave-FIFO de manera individual, un acceso como este tardaba en completarse un ciclo menos del actual. Esto es debido, al período extra que añade éste cuando reproduce el protocolo APB en su interfaz maestro de salida. Por lo tanto ésta y todas las demás situaciones evaluadas en los siguientes puntos durarán un ciclo de reloj más que cuando fueron evaluadas individualmente en la verificación individual de los módulos.*(figura 40 y 41).*

Se comprueba cómo el dato es introducido correctamente en la FIFO. El estado del componente pasa a ser de 'Empty' a contener un elemento. También, observando la *figura 41*, podemos corroborar cómo el Arbiter reproduce el protocolo APB y esto retrasa el fin de la transferencia un ciclo de reloj más.

3. USB-APB lee ese dato de la FIFO.

El módulo recibe el dato introducido anteriormente por su puerto de entrada 'prdata'. Cuando ha finalizado la transferencia APB en el instante de 75 ns, el módulo Slave-FIFO vuelve a su estado inicial de vacío; actualizándose el registro de estado. *(figura 40).*

4. USB-APB intenta leer más datos del búfer vacío.

Al intentar leer de una FIFO vacía, la señal de 'error' de la FIFO se activa y el módulo USB-APB es notificado de ello en la última fase del protocolo cuando la señal 'pslverr' está a nivel alto, junto con la señal de 'pready' que indica la última fase de la transacción. Al finalizar ésta, en el instante de 195 ns, el registro de estado del módulo Slave-FIFO es actualizado con la nueva información.*(figura 40).*

5. USB-APB lee el registro de estado del Slave-FIFO.

La lectura del registro es la adecuada, se obtiene el valor del registro de estado por el puerto de entrada 'prdata'. En este punto se constata el acceso determinista al módulo Slave-FIFO ya mencionado con anterioridad en otros apartados. La transferencia APB tiene la misma duración en este caso, en el que se está accediendo a una dirección diferente a la del búfer FIFO, que si el acceso fuese a este último elemento. *(figura 40).*

6. USB-APB llena el búfer FIFO del módulo Slave-FIFO.

Al final de la captura de la *figura 40* se puede ver el comienzo de este punto en el instante 255 ns, el cuál consta de 1024 transferencias APB. Debido a que cada una de ellas tiene una duración de 60 ns, la última finalizará en el instante 61695 ns, como se puede observar en la *figura 42*. Al realizar el cálculo teórico, se puede apreciar que coincide con el dato obtenido del ISim :

$$t_{fin} = t_{inicio} + (n^{o}_{datos} * t_{undato}) = 255\,ns + (1024 * 60ns) = 61695\,ns$$

En este instante, la señal de 'Full' de la FIFO cambia su nivel lógico a alto y el registro de estado actualiza tanto el número de elementos que contiene el búfer, como el código de excepción.

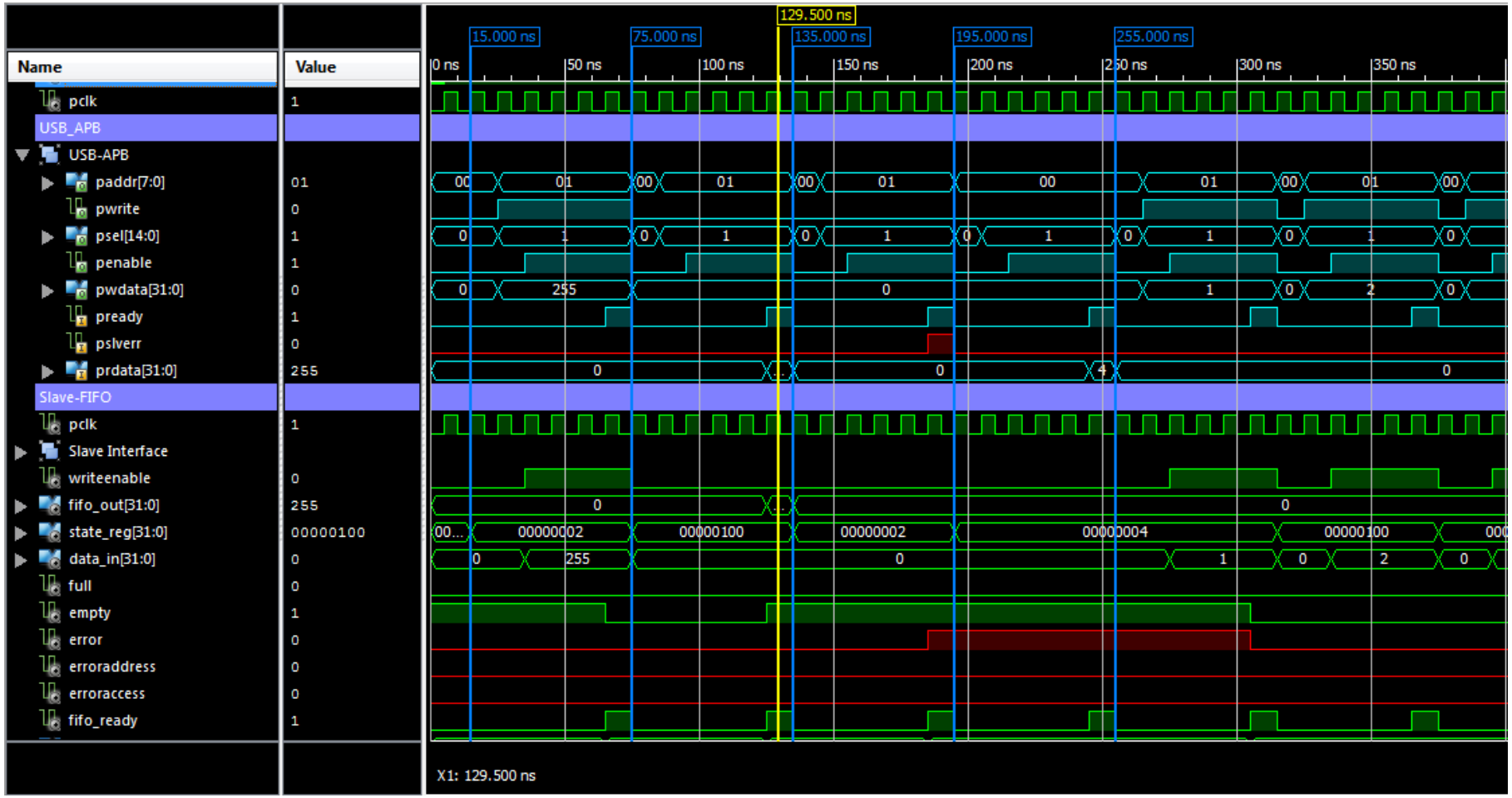


Figura 40: *Captura del resultado de la verificación del sistema de los puntos 1 al 6 (parte I)*

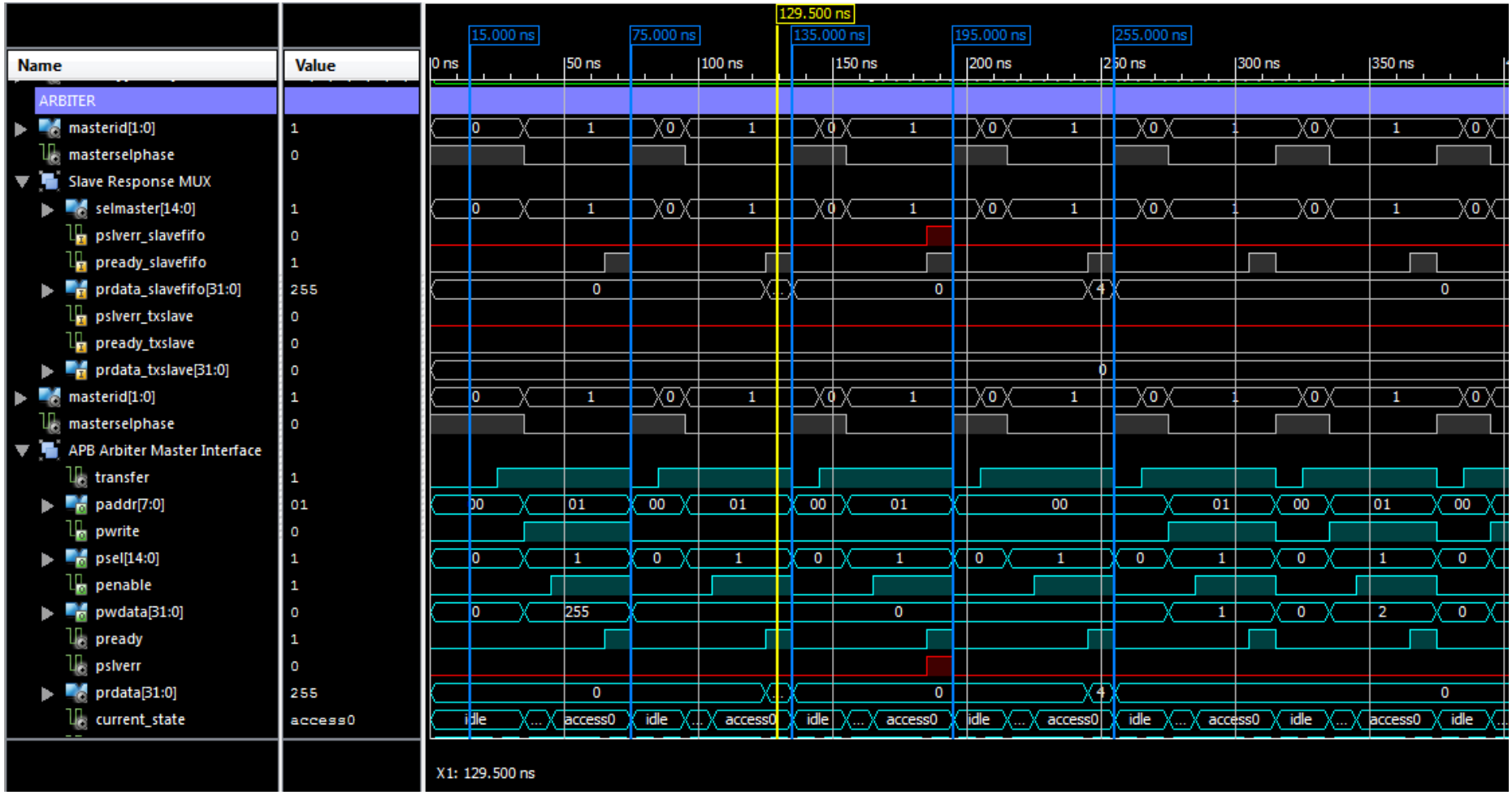


Figura 41: *Captura del resultado de la verificación del sistema de los puntos 1 al 6 (parte II)*

7. USB-APB comanda la lectura del registro estado del Slave-FIFO

La lectura del registro de estado es la adecuada, por el puerto 'prdata' se obtiene el valor del registro de estado del módulo del Slave-FIFO que, en este caso , es 0x00040001. Los 8 LSB , que corresponden al código de excepción hacen referencia al estado actual del búfer, que es lleno ('Full'). Los demás indican el número de elementos de la FIFO, en este caso 1024 datos. *(figura 42)*

8. USB-APB intenta la escritura de varios datos en la FIFO, pese a saber que está llena.

Al realizar el intento de escritura de dos datos en la FIFO, se activa la señal de error en el módulo esclavo y, debido a esto, también lo hace la señal entrante 'pslverr' del módulo USB-APB. El registro de estado se actualiza de forma adecuada.*(figura 42)*

9. USB-APB comanda la lectura del registro estado del Slave-FIFO

El inicio de esta transferencia APB es en el instante de 61.875 ns del *Anexo C, figura 42* y el comportamiento es el esperado. El módulo USB-APB obtiene el valor del registro de estado por el puerto de entrada 'prdata'.

10. USB-APB intenta la escritura en registro estado del módulo Slave-FIFO

Al intentar realizar esta acción no permitida, la señal 'pslverr' se activa avisando de ello. El código de excepción del módulo Slave-FIFO se actualiza correctamente y con ello el registro de estado.*(figura 42)*

11. USB-APB intenta acceder a dirección inexistente del módulo Slave-FIFO.

De forma similar a lo que sucedía en el punto anterior, ésta es una situación irregular que debe producir una respuesta de error. El comportamiento es el esperado, la señal de 'pslverr' se activa y se actualiza el registro de estado del módulo Slave-FIFO. En el instante de 61995 ns en la *figura 42* se puede observar que en el módulo esclavo la señal de error coexiste con las demás. Esto se debe a que, hasta que no se produzca un acceso adecuado al búfer, ésta no pasará a valer '0'. El código correspondiente a este error permanecerá en el registro de estado hasta que no se acceda de forma correcta y sólo variará si se produce algún error de otra índole más reciente y hasta que éste último sea leído.

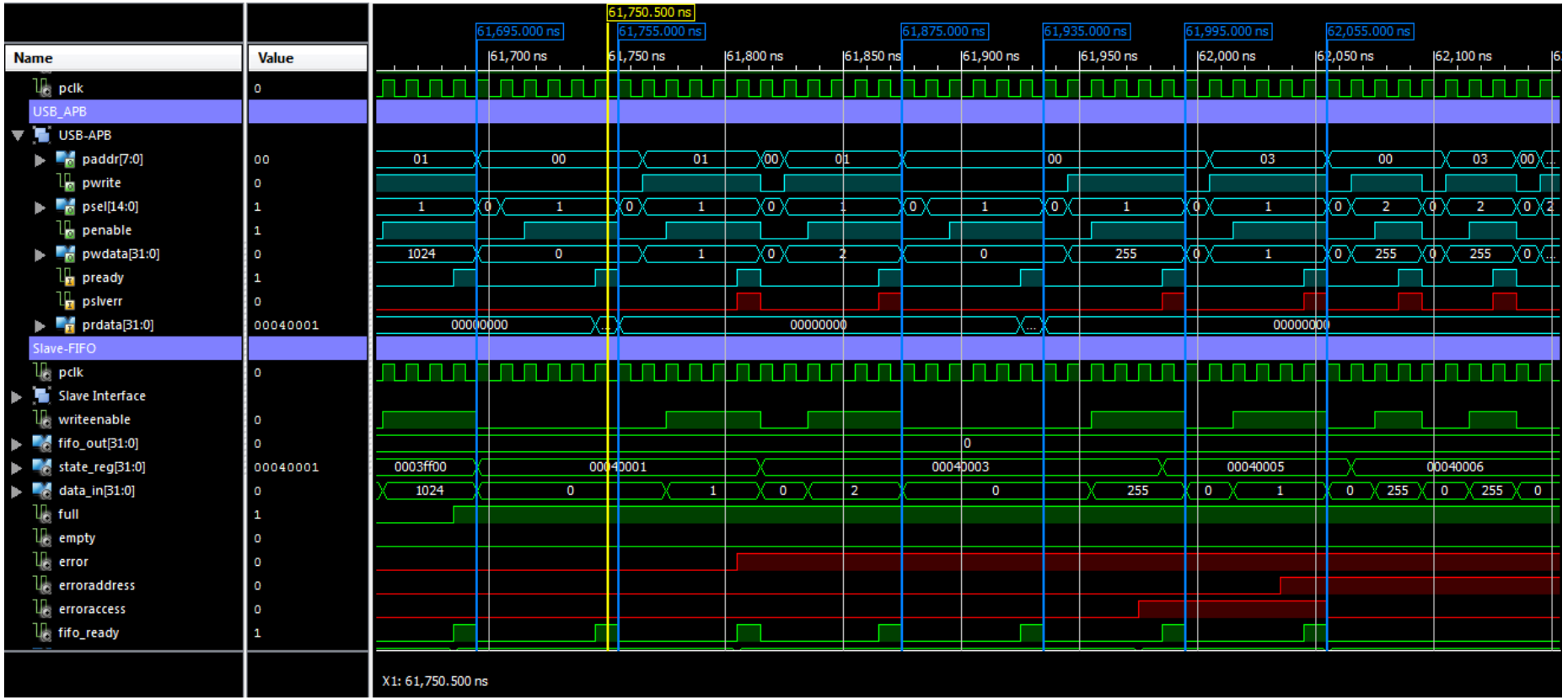


Figura 42: *Captura del resultado de la verificación del sistema de los puntos 6 al 11*

12. USB-APB intenta escribir registro estado del TX.

Se aprecia que cuando el módulo USB-APB se comunica con el módulo TX la transferencia APB tarda en completarse 40 ns, 20 ns menos que cuando se comunicaba con el Slave-FIFO. Esto se debe a que el trasmisor ha sido diseñado de tal manera que no existan estados de espera en la comunicación APB. Como se puede constatar observando la interfaz esclavo del transmisor. Sin embargo, y como se ha comentado en el punto uno de esta simulación, el módulo Arbiter provocará que toda comunicación APB del sistema se prolongue un ciclo más de reloj y se mantenga un ciclo más en el estado de ACCESS. Por este motivo la transferencia APB se extiende hasta los 40 ns ,en vez de durar 30 ns, que sería lo correspondiente a una transferencia sin estados de espera.*(figura 43)*

En cuando a la orden comandada en el instante temporal de 62,055 ns, el comportamiento es el esperado. Se activa la señal de 'pslverr' en la última fase de la transferencia APB y es salvaguardada la integridad del registro de estado.

13. USB-APB intenta acceder a dirección no existente del TX.

Este es un caso similar al anterior. La señal 'pslverr' se activa, indicando que ha sucedido una situación de fallo en el módulo transmisor. Como ya se ha mencionado en otras ocasiones, el módulo transmisor no almacena información alguna que identifique el origen del error producido, a diferencia del módulo Slave-FIFO.

(figura 43)

14. USB-APB comanda lectura y TX datos en paralelo de 1024 datos

Nuevamente en el instante temporal de 62135 ns de la *figura 43* se aprecia cómo el módulo USB-APB inicia transferencia APB para comandar al módulo transmisor la lectura y envío en paralelo hacia el sistema de adquisición de 1024. Para lograrlo, accede al registro de transferencia y modifica su contenido satisfactoriamente. Como consecuencia, el registro de estado también se actualiza poniéndose a '1' el bit TXIP, que indica el envío de datos en curso.

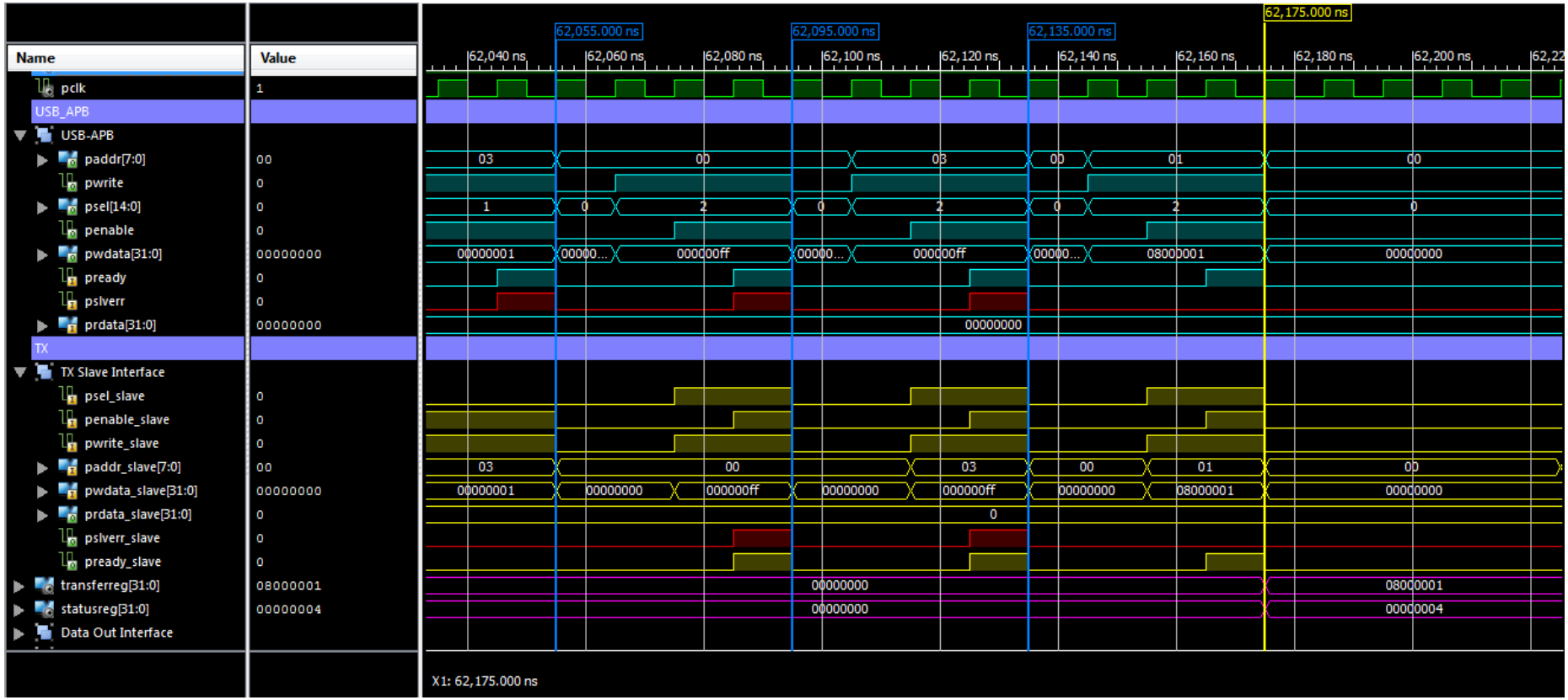


Figura 43: *Captura del resultado de la verificación del sistema de los puntos 12,13 y 14*

5.2.1. Análisis de la extracción de datos de la FIFO y transmisión en paralelo hacia el sistema de adquisición.

La *figura 44* muestra el inicio de la transmisión de datos en paralelo hacia el sistema receptor. El primer cursor, situado en el instante de 62.175 ns corresponde al fin de la operación APB de escritura que modifica el registro de transferencia del transmisor, comenzando así el envío de datos por el bus paralelo. En este instante la señal de 'transfer' pasa a estar activa. Según la lógica de sincronismo diseñada , cuando los pulsos 'tick1', 'tick2' y la propia señal de 'transfer' valgan '1', se activará a la señal 'getNewData' y el módulo TX solicitará el primer dato al Slave-FIFO.

Se puede observar que la transferencia APB entre la interfaz maestro del módulo TX y el Slave-FIFO conlleva 60 ns, o 6 períodos de reloj, al igual que sucedía en el punto 1 de esta sección. Este hecho constata, aún más si cabe, la propiedad determinista del módulo Slave-FIFO.

Cuando el transmisor recibe el dato, lo registra y la señal 'newData' se mantiene activa hasta que el dato registrado es puesto en la interfaz de salida en el momento adecuado.

Se aprecia en la *figura 45* cómo el transmisor completa la orden de transmisión comandada en el instante de 267.115 ns, causando que el registro de estado cambie de valor. El bit TXIP pasa a valer '0' y el bit de DONE que indica que la orden de transmisión comandada ha sido llevada a cabo con éxito, pasa a valer '1'. En la parte inferior de la captura del simulador se puede apreciar observando la señal de 32 bits 'statusreg'.

Además también se confirma que la señal 'ENABLETX' se mantiene activa hasta que se ha enviado el último dato, en este caso corresponde al número 0.

15. USB-APB lee el registro de estado del transmisor

El resultado es el esperado. El módulo obtiene por su puerto de entrada 'prdata' el valor del registro de estado del transmisor.*(figura 45).*

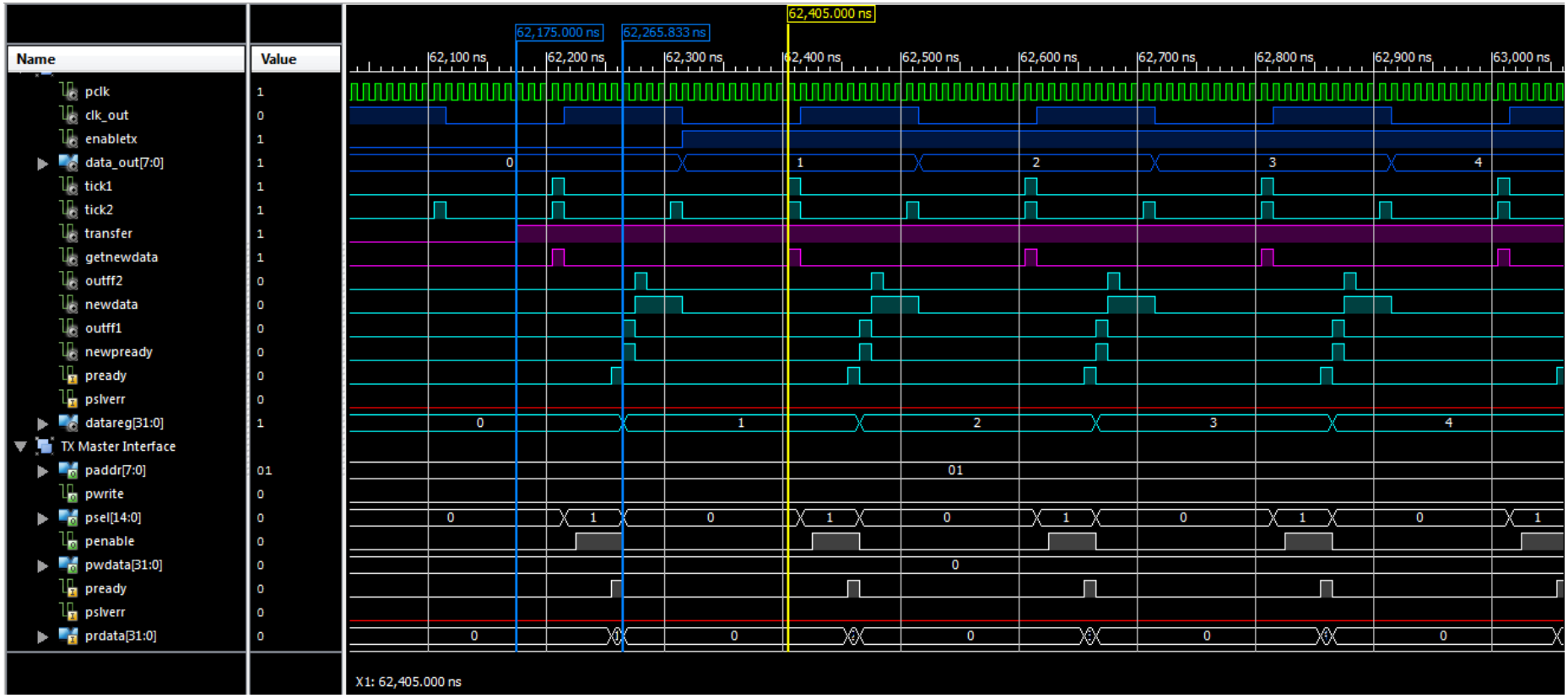


Figura 44: *Captura del resultado de la verificación de la interfaz de salida de datos en paralelo del módulo TX*

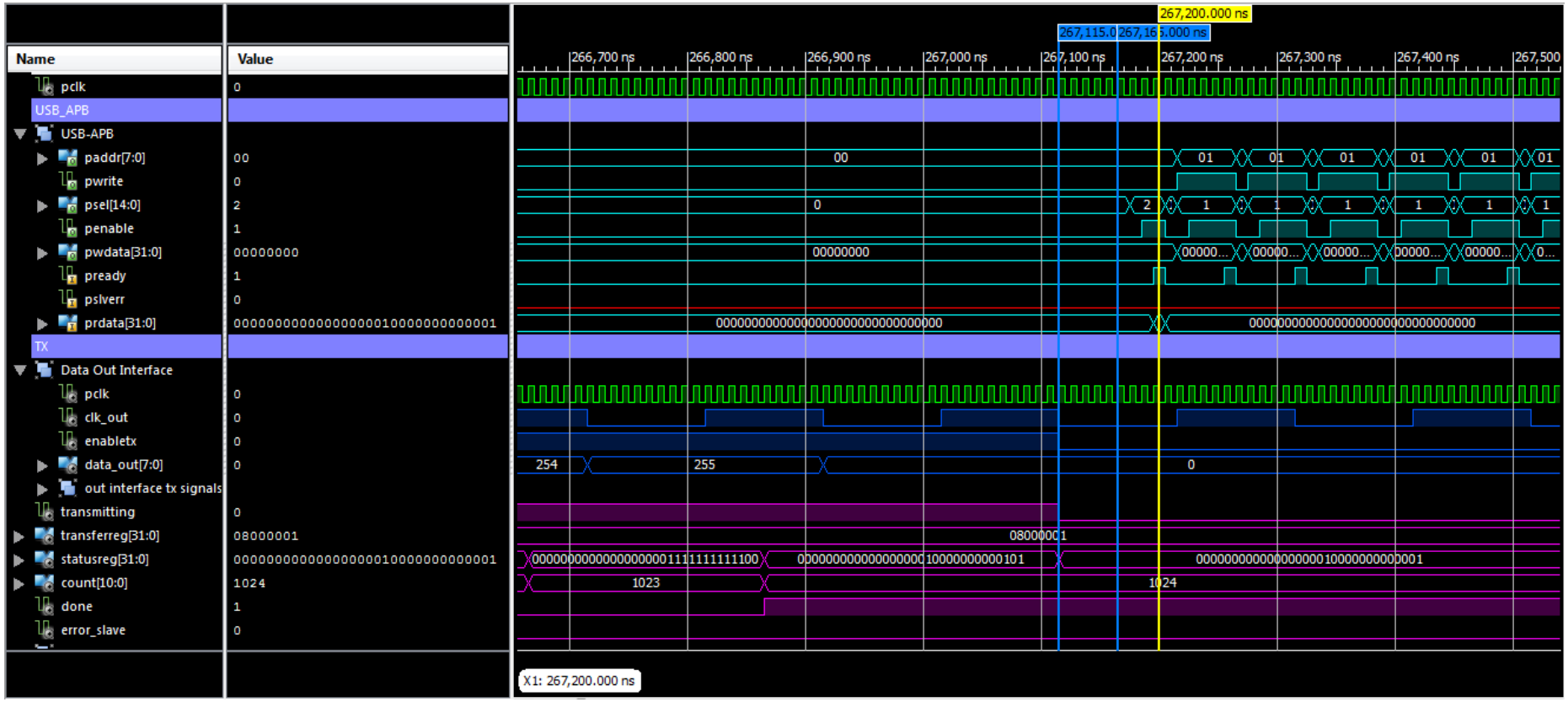


Figura 45: *Captura del resultado de la verificación del fin del envío de datos hacia el sistema de adquisición*

16. USB-APB vuelve a llenar la FIFO, ordena al módulo TX el envío de 256 datos y cancela la transmisión antes que finalice.

En el instante temporal de 328685 ns*(figura 46)*, al finalizar el último ciclo de la transferencia APB, el registro de transferencia cambia de valor adecuándose a esta nueva orden. A consecuencia de ello, también lo hace el registro de estado y el módulo transmisor comienza la sincronización para pedir el primer dato al Slave-FIFO. Pasados 260 ns se decide cancelar la transmisión, modificando de nuevo el registro de transferencia. Se aprecia que aunque la señal de 'transfer' pase a valer cero al comandar esta orden, la señal de 'ENABLETX' no se deshabilitará con el flanco de bajada de la señal de reloj hasta que no se envíe el último dato extraído de la FIFO.

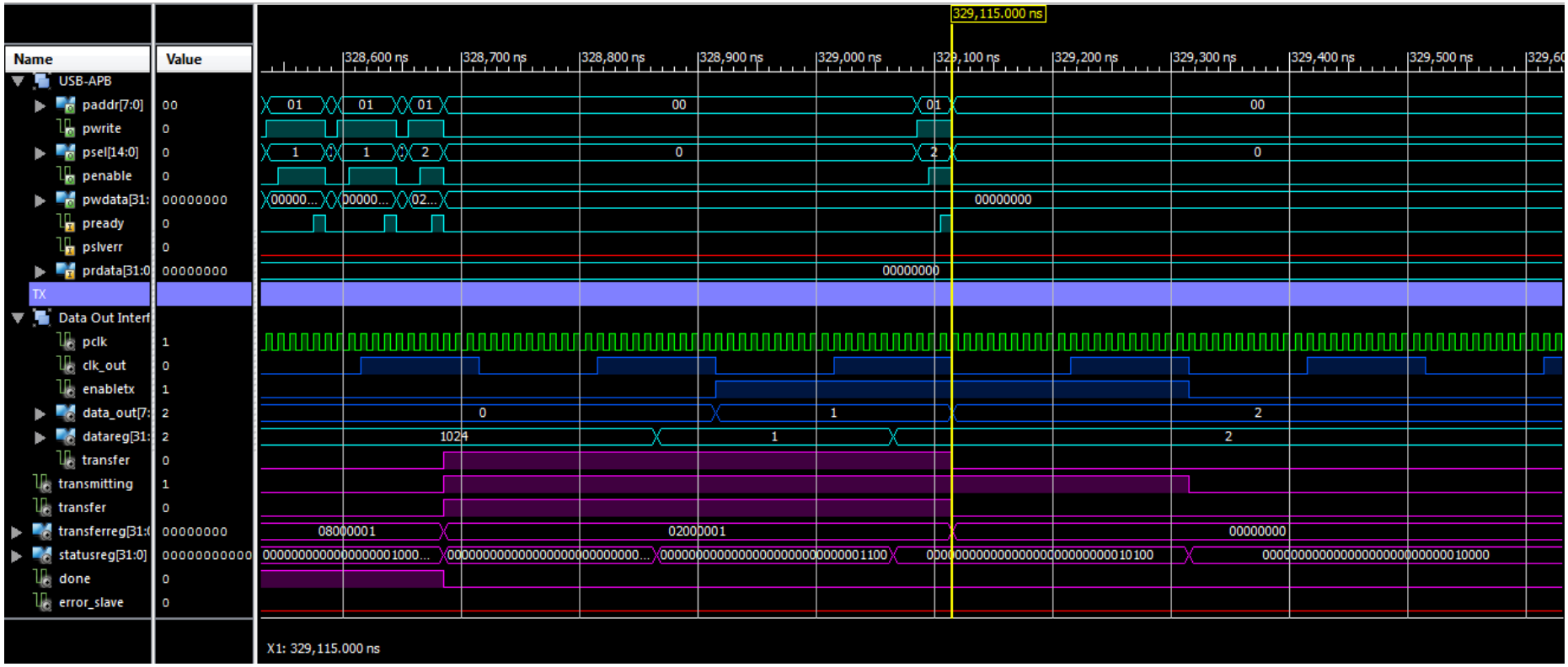


Figura 46: *Captura del resultado de la verificación del sistema del punto 16*

Capítulo 6

Validación del sistema

El proceso de validación forma parte del segundo de los procesos de la etapa de evaluación del sistema antes de la entrega de cualquier producto.

Consiste en un conjunto de pruebas al sistema, llevadas a cabo bajo un entorno representativo del posible modelo a emplear en un CubeSat para determinar si éste cumple tanto el propósito, como las expectativas para el cuál fue diseñado.

También garantiza su correcto funcionamiento en este entorno y que cualquier anomalía pueda ser descubierta y resuelta antes de la entrega del producto.

Primero se hará una descripción breve del set-up de pruebas a usar y, posteriormente, se describirán las pruebas llevadas a cabo y sus resultados.

El set-up de validación creado consta de dos fases:

- Validación a nivel lógico del sistema transmisor.
- Validación completa del sistema formado por el transmisor y un módulo receptor.

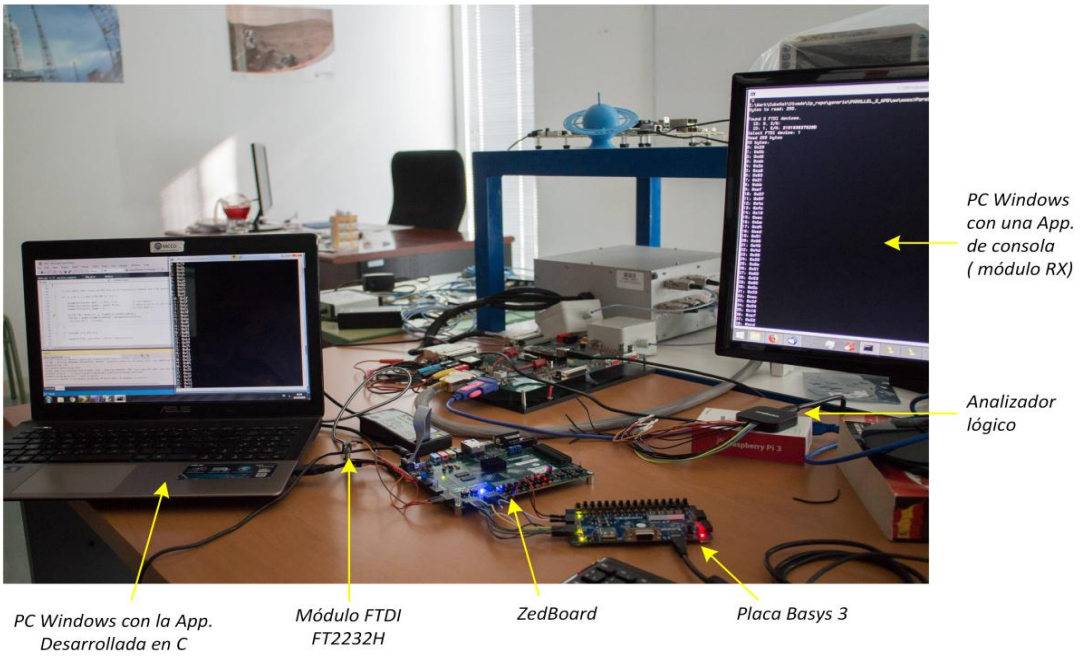


Figura 47: *Set-up de pruebas completo*

6.1. Aplicación desarrollada en lenguaje C

La aplicación desarrollada en lenguaje C tiene como cometido principal permitir que el usuario sea capaz de comunicarse mediante USB con el sistema implementado. Para lograrlo, se ha creado una aplicación de consola usando el entorno de desarrollo integrado de Microsoft Visual Studio.

Los archivos de los que consta y su funcionalidad son descritos brevemente a continuación.

Header Files

- apb_ftdi_wrapper.h

  Archivo proporcionado que contiene la declaración de las funciones necesarias para manejar el dispositivo FTDI y realizar las operaciones APB básicas.

- apb_functions.h

  Contiene la declaración de las funciones específicas diseñadas para el control del sistema; como, por ejemplo la función 'writeLoopFIFO'.

- ftd2xx.h

  Controlador de dispositivo USB para los modelos FTDI FT232x, FT245x, FT2232x y FT4232x

Resource Files

- uart_apb_ssel.lib

  Librería proporcionada con las funciones y declaraciones necesarias para la comunicación con el módulo USB-APB.

Source Files

- apb_functions.cpp

  Contiene la implementación de las funciones específicas para el sistema declaradas en el archivo 'apb_functions.h'.

- **test1.cpp**

  Programa principal de la aplicación de consola desde la que se instanciarán las funciones a usar para el control del sistema y el resto de código de la aplicación. Desde aquí se llamará a las diversas funciones implementadas en 'apb_functions.cpp' de forma secuencial hasta evaluar todos los puntos descritos en el plan de test. Además, se comprobará si el código devuelto que indica el resultado de la operación, es el esperado o no.

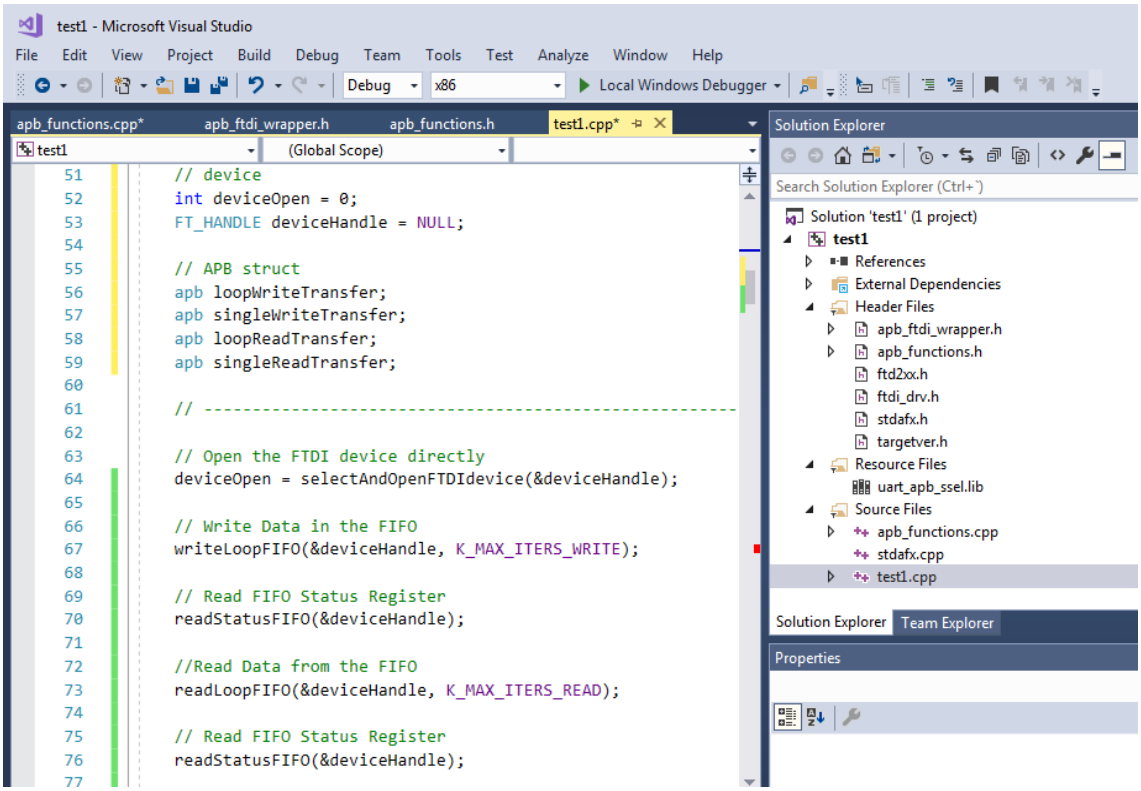


Figura 48: *Captura del programa principal de la App. desarrollada en lenguaje C*

6.2. Equipo

6.2.1. ZedBoard (Zynq Evaluation and Development Board)

Es una plataforma de evaluación y desarrollo basada en el AP SoC Zynq-7000 de Xilinx, en concreto incluye el dispositivo XC7Z020. Éste es uno de los dispositivos más pequeños de esta gama y está basado en la familia de FPGAs Artix-7 de Xilinx. Es considerada como una de las plataformas de desarrollo más populares y alberga un gran número de periféricos integrados y capacidades de expansión.

Esta placa contiene el sistema generador de datos basado en FPGA del presente proyecto. Es decir, alberga los módulos: USB-APB proporcionado, Slave-FIFO, TX y Arbiter.

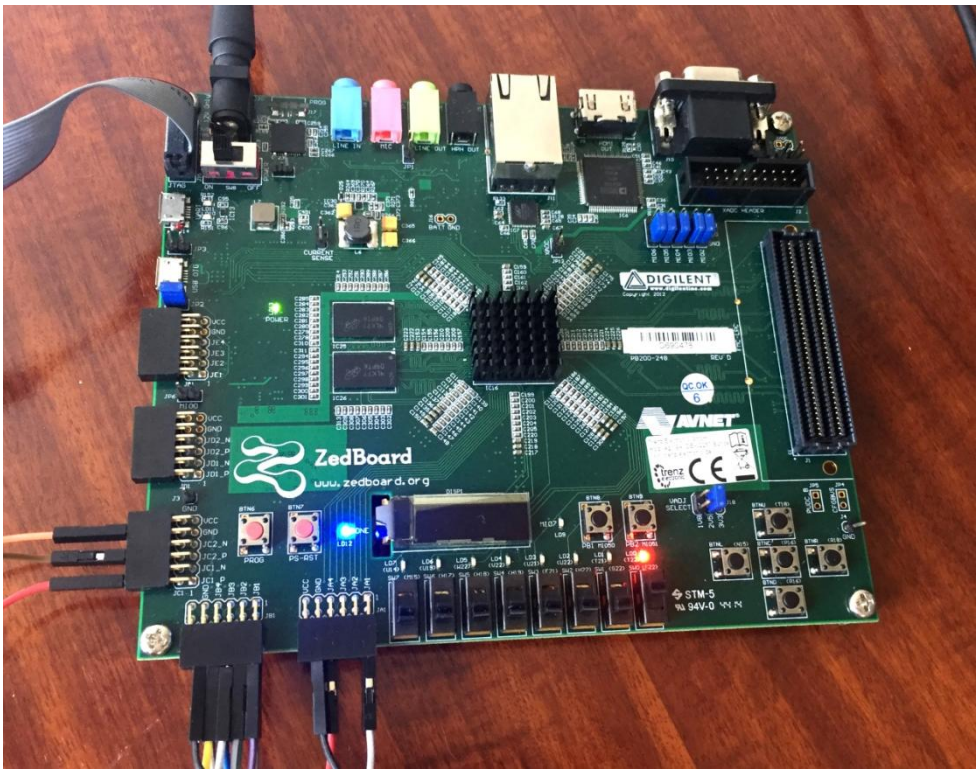


Figura 47: *ZedBoard*

6.2.2. Placa Basys 3

Es una plataforma de desarrollo sencilla basada en la FPGA Artix-7 de Xilinx y diseñada para ser programada empleando el entorno de ' Vivado Design Suite'. Cuenta con una FPGA de alto rendimiento, una interfaz USB, una pantalla de 7 segmentos, un gran número de dispositivos de entrada/salida incorporados y cuatro conectores de expansión PMOD, entre otros elementos hardware.

En este proyecto ha sido empleada para albergar el receptor proporcionado y, de este modo, realizar la fase 2 de validación completa del sistema.

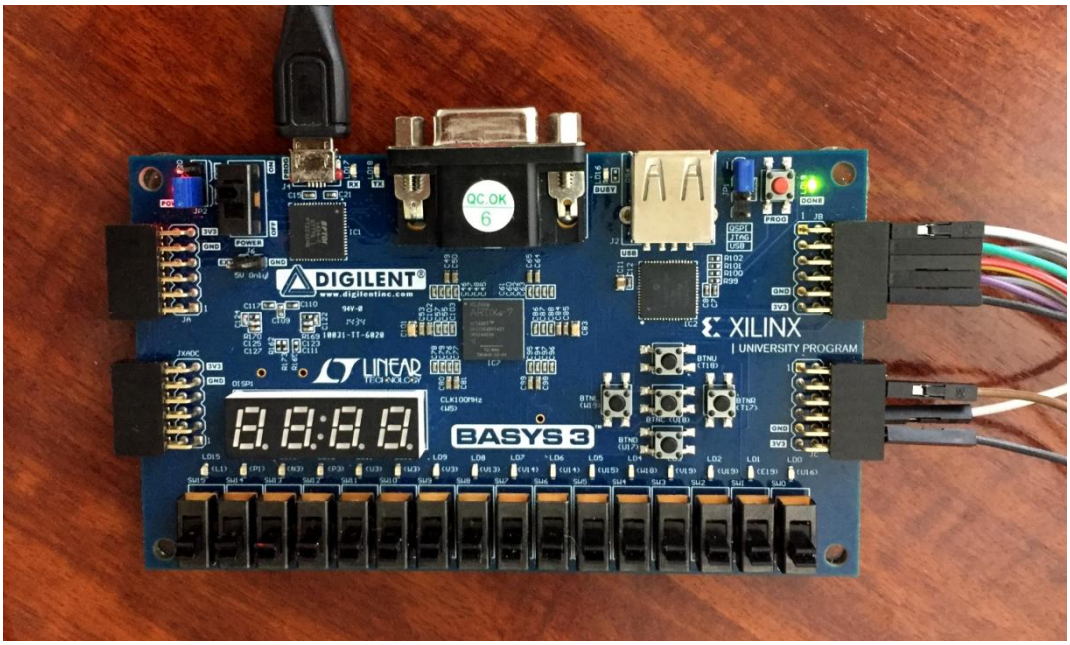


Figura 48: *Placa Basys 3*

6.2.3. Analizador lógico de 8 entradas de la marca 'Saleae'.

Un analizador lógico es un dispositivo cuyo objetivo es visualizar los valores digitales de un circuito durante un período de tiempo de adquisición. Para ello, reúne varias muestras y las almacena para su posterior visualización en pantalla, haciendo uso del software apropiado.

Para el sistema evaluado se ha empleado un analizador lógico de 8 entradas de la marca 'Saleae' conectado a las señales de la interfaz de salida de datos del módulo transmisor. Cada entrada está conectada a cada una de las salidas[3].

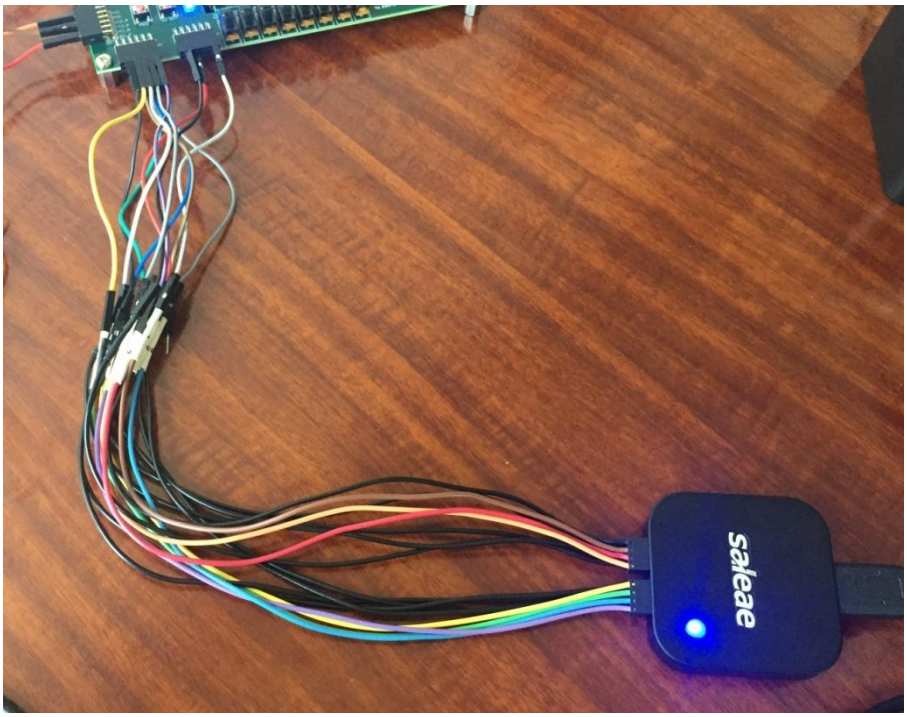


Figura 49: *Analizador lógico*

[3] El bus del sistema diseñado opera con palabras de 32 bits; sin embargo, existe una limitación física debido a que no se dispone de los recursos suficientes en ambas placas para conectar todas las señales necesarias. Por este motivo, el módulo TX envía por la interfaz de salida de datos palabras de 8 bits, correspondientes a los LSB de los datos de 32 bits extraídos de la FIFO. Por lo tanto, para la interfaz de salida de datos del transmisor son necesarias las 8 líneas de datos del conector Pmod JB1 y dos del conector JA1. Para poder visualizar la totalidad de las señales, sería necesario que el analizador lógico constase de 10 entradas, en vez de 8. Ante esta nueva limitación, se decidió conectar al analizador la señal de reloj, la señal habilitante y los 6 LSB del bus de dato

6.2.4. Módulo FTDI FT2232H

Es un módulo que permite la conversión de comunicación USB a serie. Idóneo para fines de desarrollo; ya que permite añadir la funcionalidad USB a un diseño de forma rápida.

Este dispositivo permite que el usuario acceda, a través de la aplicación en lenguaje C, al módulo USB-APB del sistema desarrollado para comandar las diferentes órdenes al resto de componentes. En conjunto con la librería proporcionada en lenguaje C para su desarrollo, se puede obtener información relativa a la comunicación y tener conocimiento de si ha ocurrido algún error.

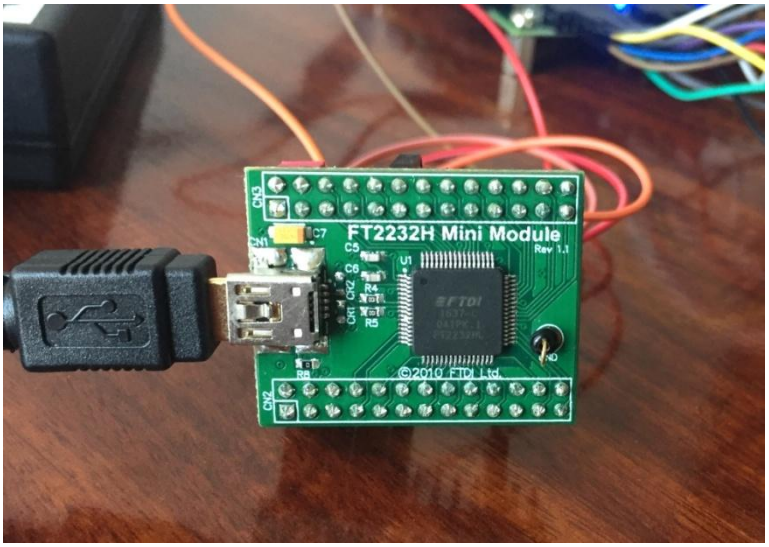


Figura 50: *Módulo FTDI FT2232H*

6.3. Validación del sistema

6.3.1. FASE I:

Validación a nivel lógico de señal del sistema transmisor

En esta primera etapa de validación, únicamente se evaluará la interfaz de salida de datos en paralelo del módulo transmisor. No se entrará en detalle en la secuencia necesaria de comunicación del usuario con los diferentes módulos del sistema para poder llevar a cabo esta operación. Toda esta secuencia será evaluada, más adelante en la fase 2 de la validación del sistema.

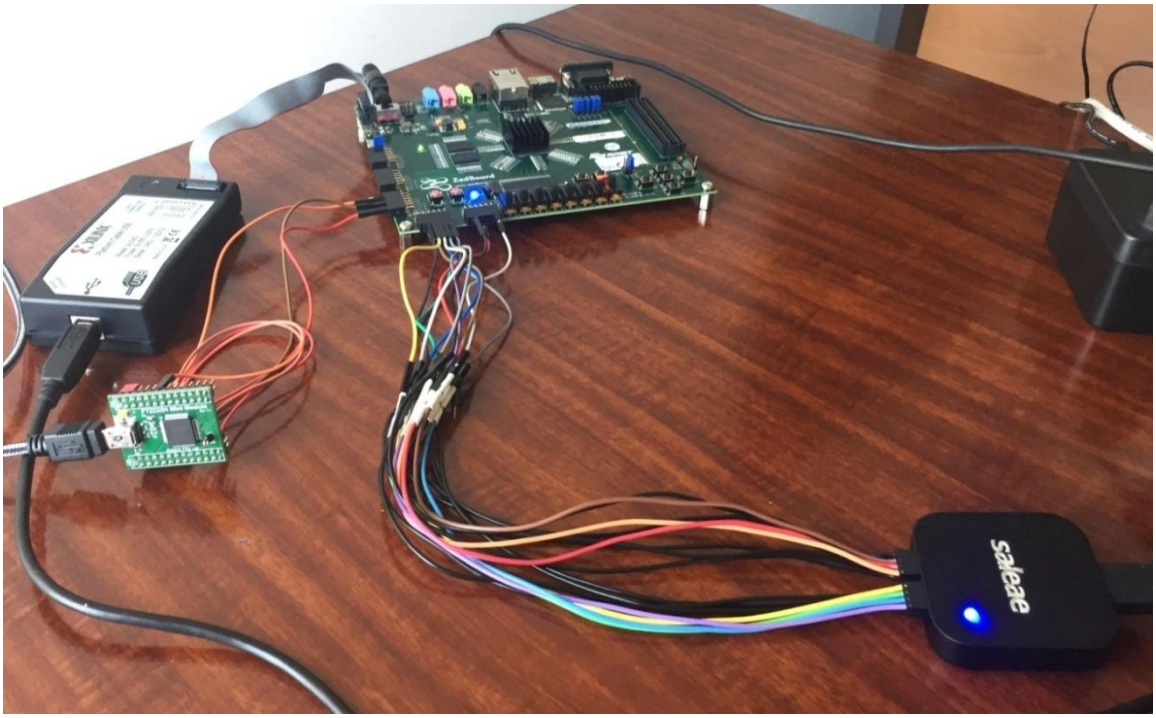


Figura 51: *Montaje de la Fase I de la validación del sistema*

6.3.1.1. Pruebas realizadas y resultados obtenidos

1. Envío de un conjunto de datos de forma adecuada y siguiendo las especificaciones de diseño.

El módulo transmisor envía adecuadamente un bloque de datos obtenidos de la FIFO. Para lograrlo, se hace uso de la aplicación C, la cual introduce en la FIFO un conjunto de datos y posteriormente comanda su transferencia en paralelo. La *figura 54* recoge la captura tomada del analizador lógico de los primeros datos transmitidos *(parte izquierda de la imagen)* y el valor de éstos primeros datos al ser introducidos en la FIFO *(parte derecha)* .Al comparar ambas capturas, se observa que el resultado obtenido es coherente.

También se puede apreciar cómo las señales cambian de estado lógico en los instantes adecuados. El bus de datos actualiza su valor con el nuevo dato en el flanco de bajada del reloj de salida, para que el receptor pueda leerlos en el de subida.

2. Envío de un conjunto de 1024 datos de forma adecuada y siguiendo las especificaciones de diseño

El bus del sistema diseñado opera con palabras de 32 bits; sin embargo, debido a las características del set-up, este valor se reduce a 8 bits. Por este motivo, la tasa de transmisión también cambiará su valor de 160 Mbits/s a 40 Mbits/s. Teniendo en cuenta esta consideración, se comprueba que el tiempo que se tarda en enviar todos los datos de la FIFO por la interfaz de salida del transmisor *(figura 55)* coincide con el valor esperado $\left(\frac{1}{40Mbits\ /s} * 1024_{palabras} * 8_{bits\ /palabra} = 0{,}2048\ ms\right)$.

3. Cancelación de una transmisión en curso.

Las señales cuando se cancela la transmisión cambian de valor en el momento adecuado, enviándose un total de 28 datos. Cifra que coincide con el número de elementos transmitidos que contiene el registro de estado del transmisor. *(figura 56).*

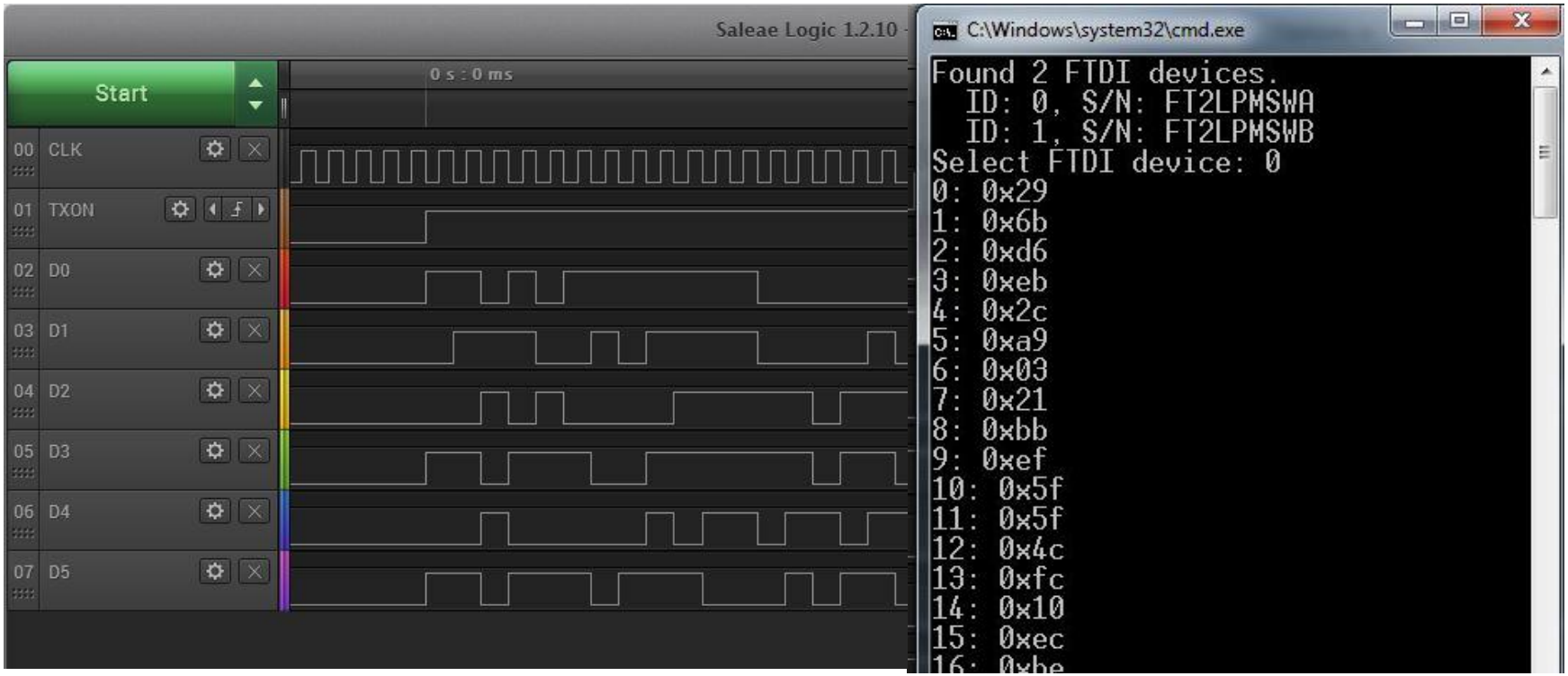


Figura 52: *Resultado prueba 1, fase I de la validación del sistema*

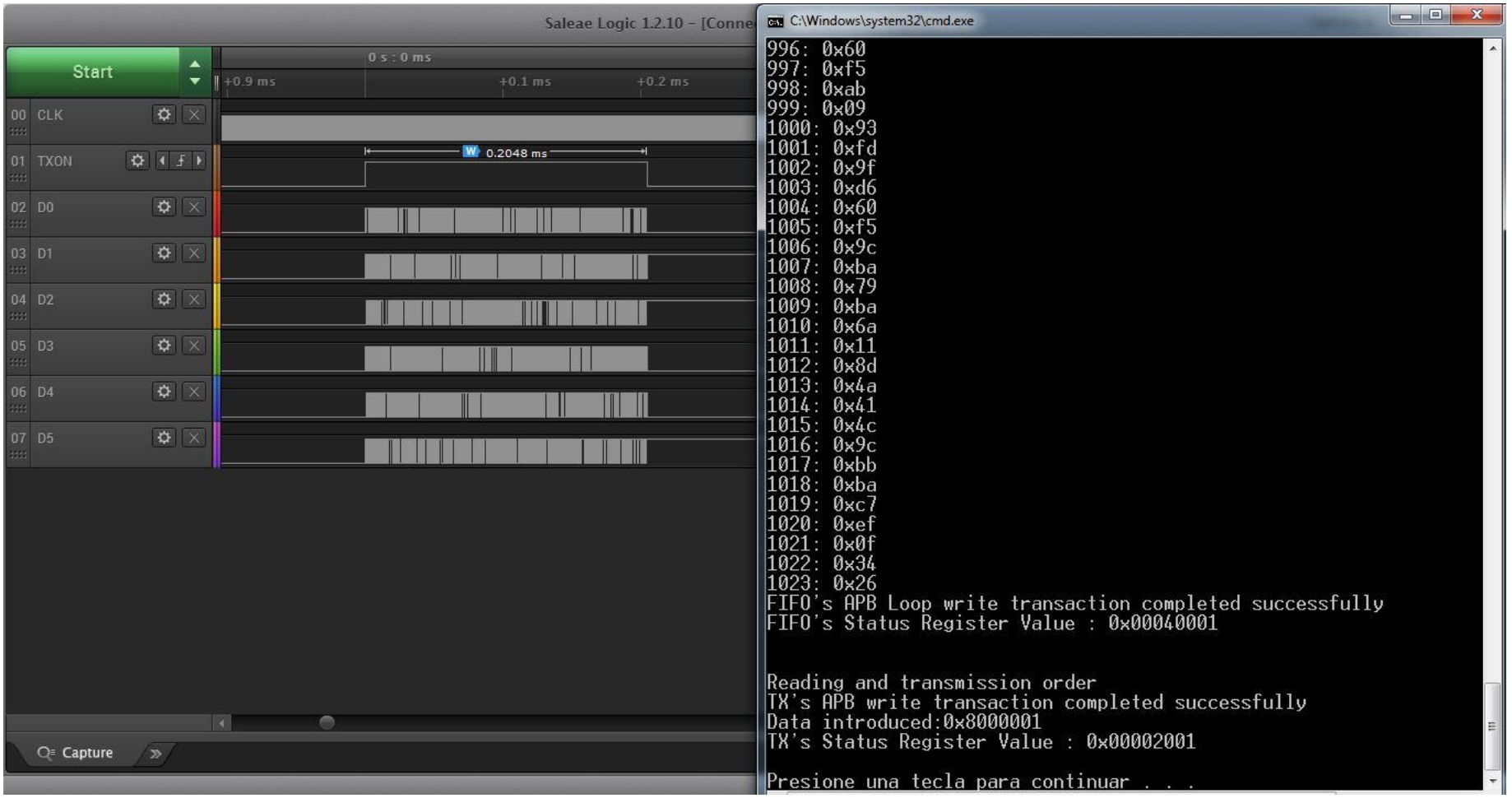


Figura 53: *Resultado prueba 2, fase I de la validación del sistema*

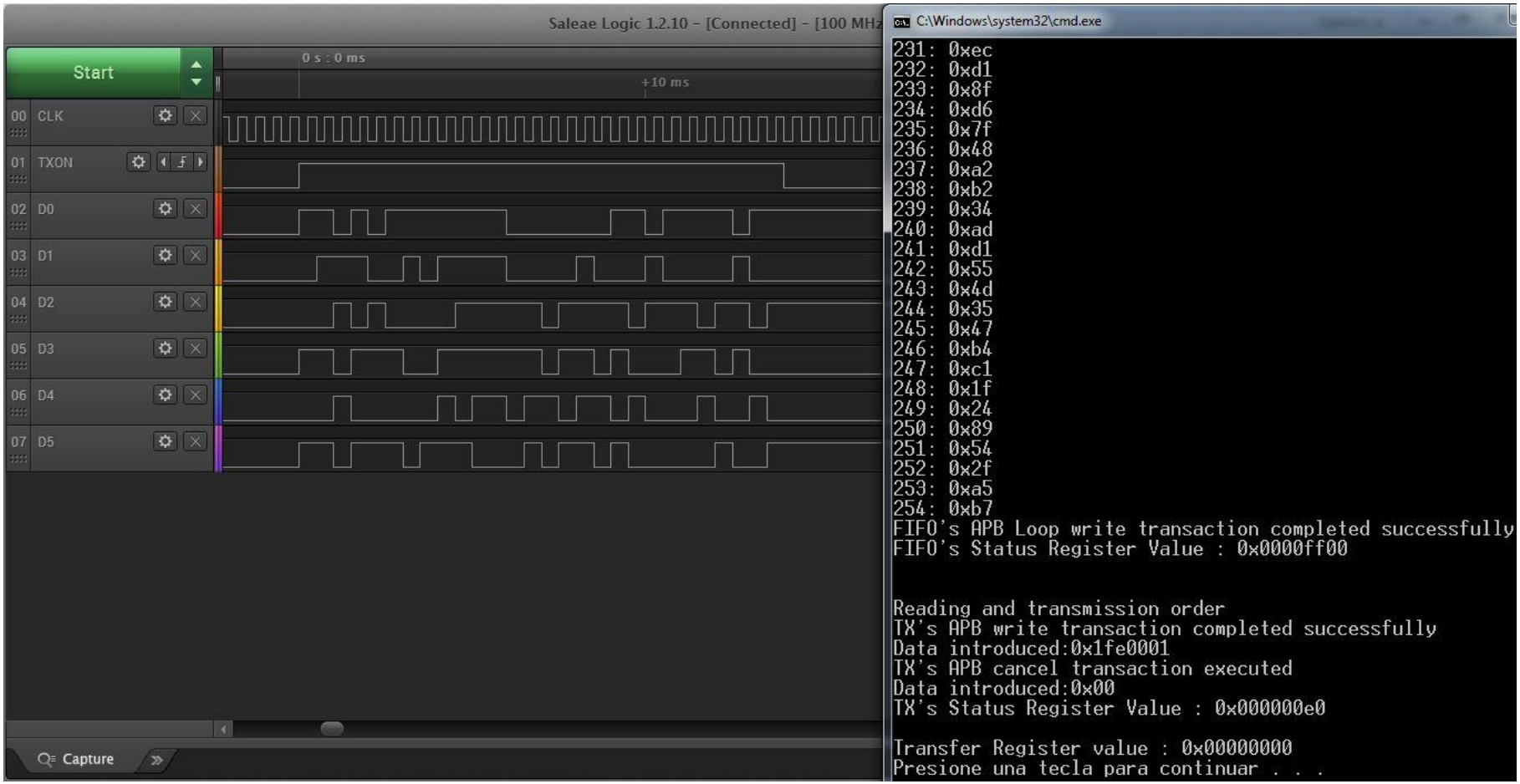


Figura 54: *Resultado prueba 3, fase I de la validación del sistema*

6.3.2. FASE II:

Validación completa del sistema formado por el transmisor y el módulo receptor.

En esta segunda fase, y última del proceso de validación, se evaluará en profundidad el conjunto de todo el sistema, incluyendo la comunicación con el módulo receptor proporcionado. Con este fin, se ha elaborado un completo plan de test para garantizar que el sistema desarrollado cumple los objetivos para los cuáles fue diseñado.

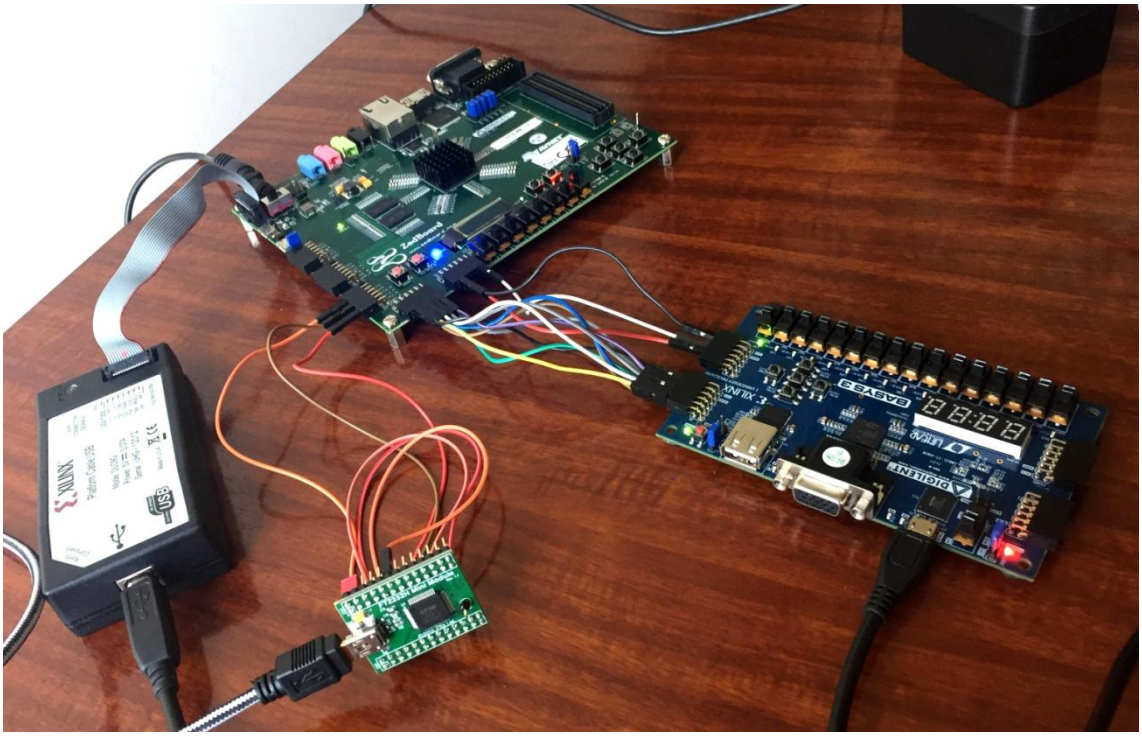


Figura 55: *Montaje de la Fase II de la validación del sistema*

6.3.2.1. Plan de test

ID	DESCRIPCIÓN	P/F*
Grupo 0: Comunicación PC-USBAPB		
0	El usuario puede seleccionar y abrir el dispositivo FTDI.	P
1	El usuario puede cerrar el dispositivo FTDI.	P
Grupo 1: Comunicación con el módulo SlaveFIFO		
2	El usuario es capaz de escribir un dato en la FIFO	P
3	El usuario puede leer el registro de estado de la FIFO	P
4	El usuario es capaz de leer un dato de la FIFO	P
5	El usuario no debe poder obtener ningún dato de una FIFO vacía (error)	P
6	El usuario no debe poder acceder a una dirección inexistente (error)	P
7	El usuario no debe poder escribir en registro de estado (error)	P
8	El usuario es capaz de almacenar un bloque pequeño de datos en la FIFO	P
9	El usuario es capaz de leer un bloque pequeño de datos de la FIFO	P
10	El usuario es capaz de llenar la FIFO	P
11	El usuario no debe ser capaz de escribir en FIFO llena (error)	P
12	Si en el búfer FIFO ha ocurrido un error, el usuario debe poder ser conocedor de él, hasta que se produzca un acceso adecuado al mismo	P

** P =Pass ; F= Fail*

Grupo 2: Comunicación con el módulo TX y envío de datos al sistema de adquisición		
13	El usuario es capaz de comandar la lectura y transmisión de un bloque pequeño de datos de la FIFO	P
14	El sistema de adquisición recibe satisfactoriamente un pequeño bloque de datos enviados por el sistema transmisor	P
15	El usuario es capaz de consultar el registro de estado del módulo TX	P
16	El usuario es capaz de cancelar una transmisión de datos antes de que ésta finalice	P
18	El usuario no debe ser capaz de escribir en el registro de estado del módulo TX (error)	P
19	El usuario no debe poder acceder a una dirección inexistente del módulo TX (error)	P
20	El usuario es capaz de comandar la lectura y transmisión de todos los datos almacenados en la FIFO	P
21	El sistema de adquisición recibe satisfactoriamente todos los datos almacenados en la FIFO enviados por el sistema transmisor	P

Tabla 8:*Plan de Test*

Los resultados obtenidos de las pruebas de validación del plan de test del grupo 0 y del grupo 1 están recogidos en las *figuras 58 y 59* ; y los del grupo 2 en las *figuras 60-64*.

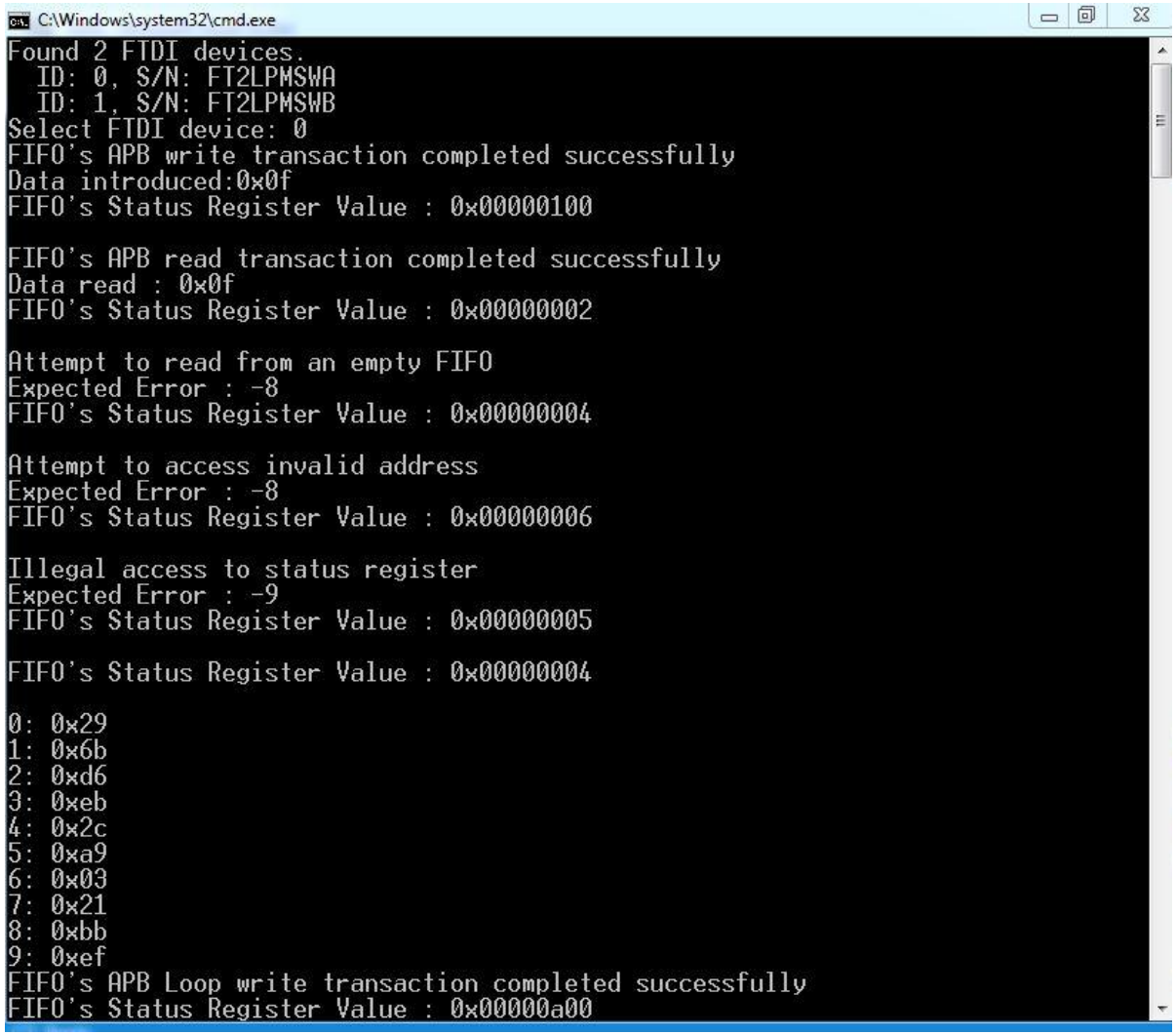


Figura 56: *Resultado de las pruebas con id 0-8 de la fase II de la validación del sistema*

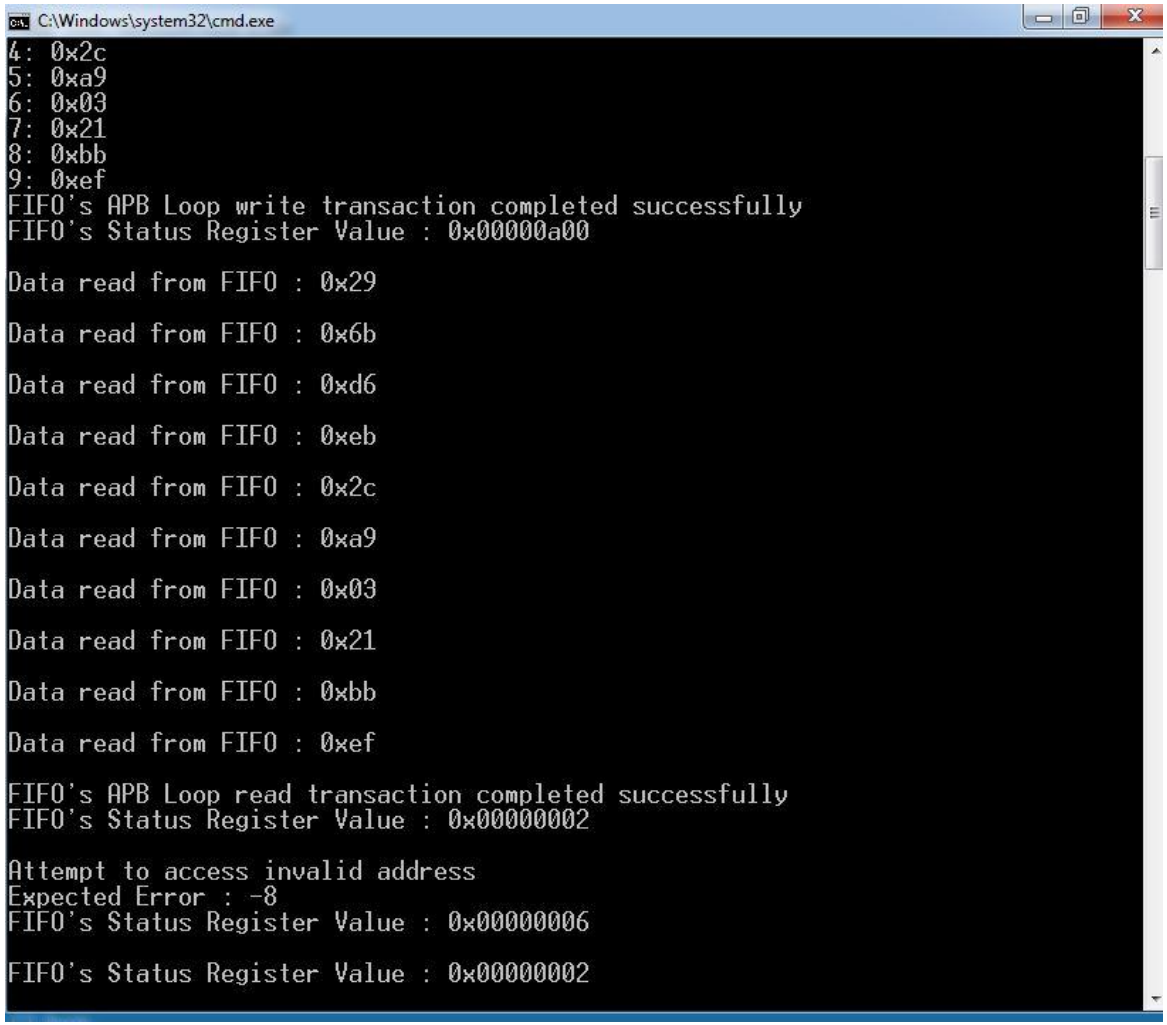


Figura 57: *Resultado de las pruebas con id 9-12 de la fase II de la validación del sistema*

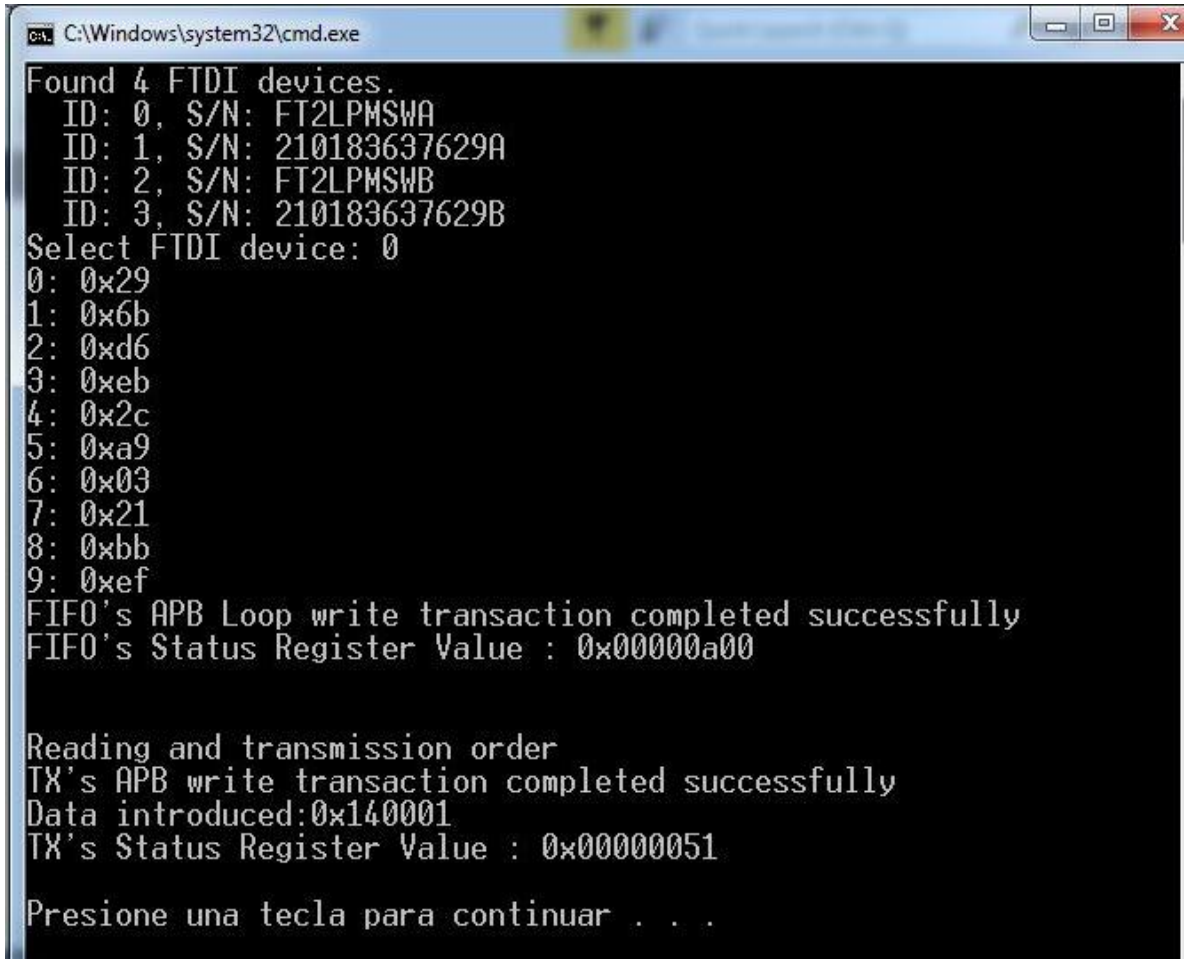


Figura 58: *Resultado de las pruebas con id 13 y 15 de la fase II de la validación del sistema (sistema transmisor)*

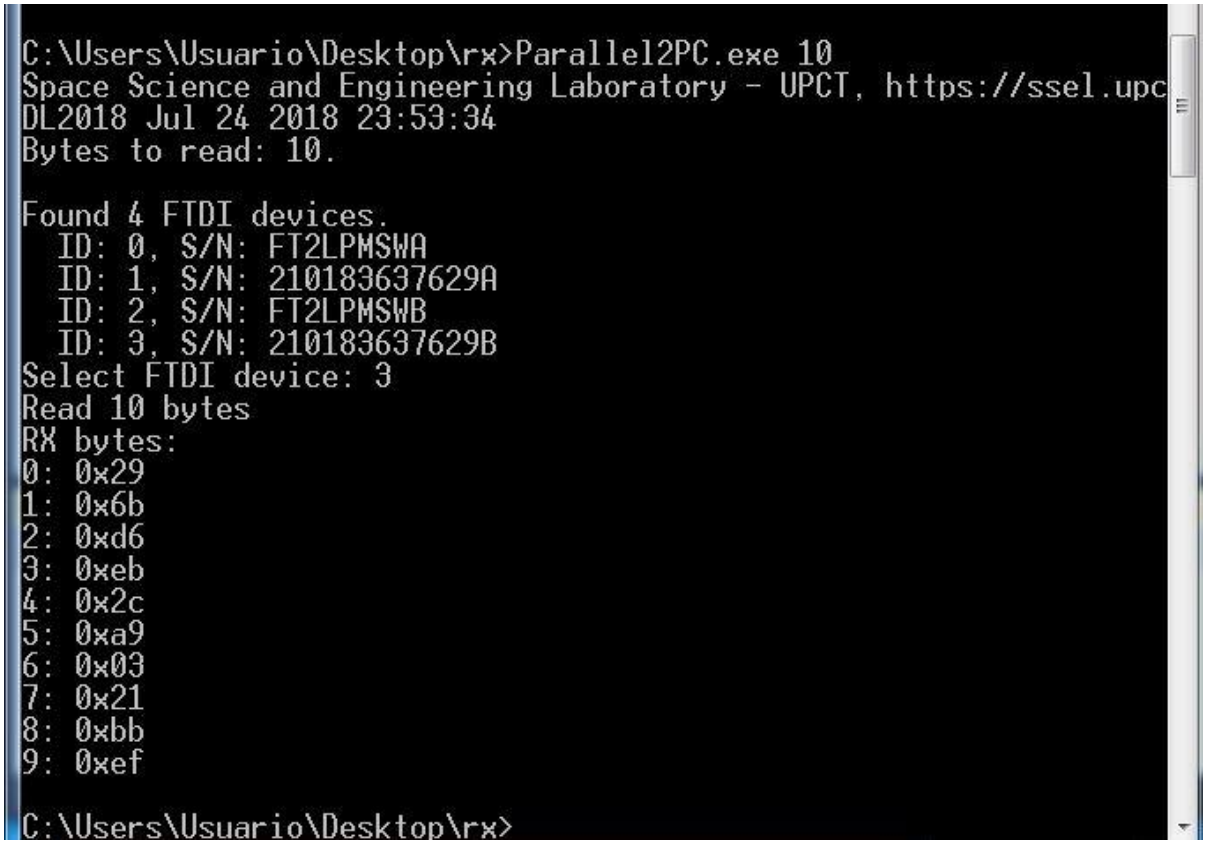


Figura 59: *Resultado de la prueba con id 14 de la fase II de la validación del sistema (sistema receptor)*

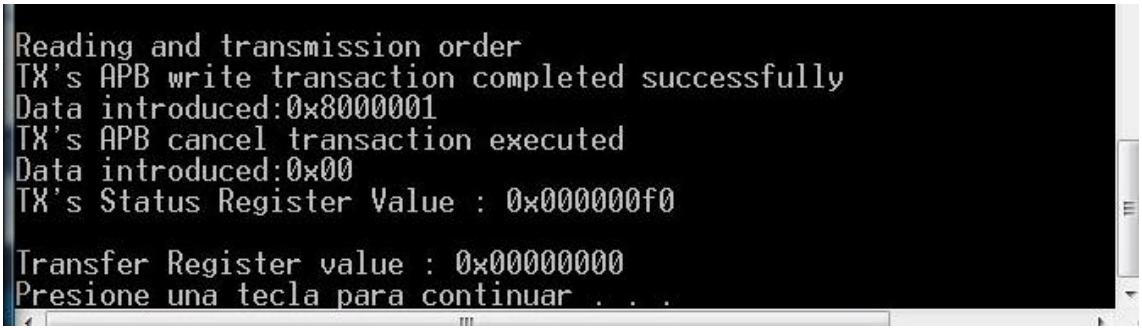


Figura 60: *Resultado de las pruebas con id 15, 16 y 20 de la fase II de la validación del sistema (sistema transmisor)*

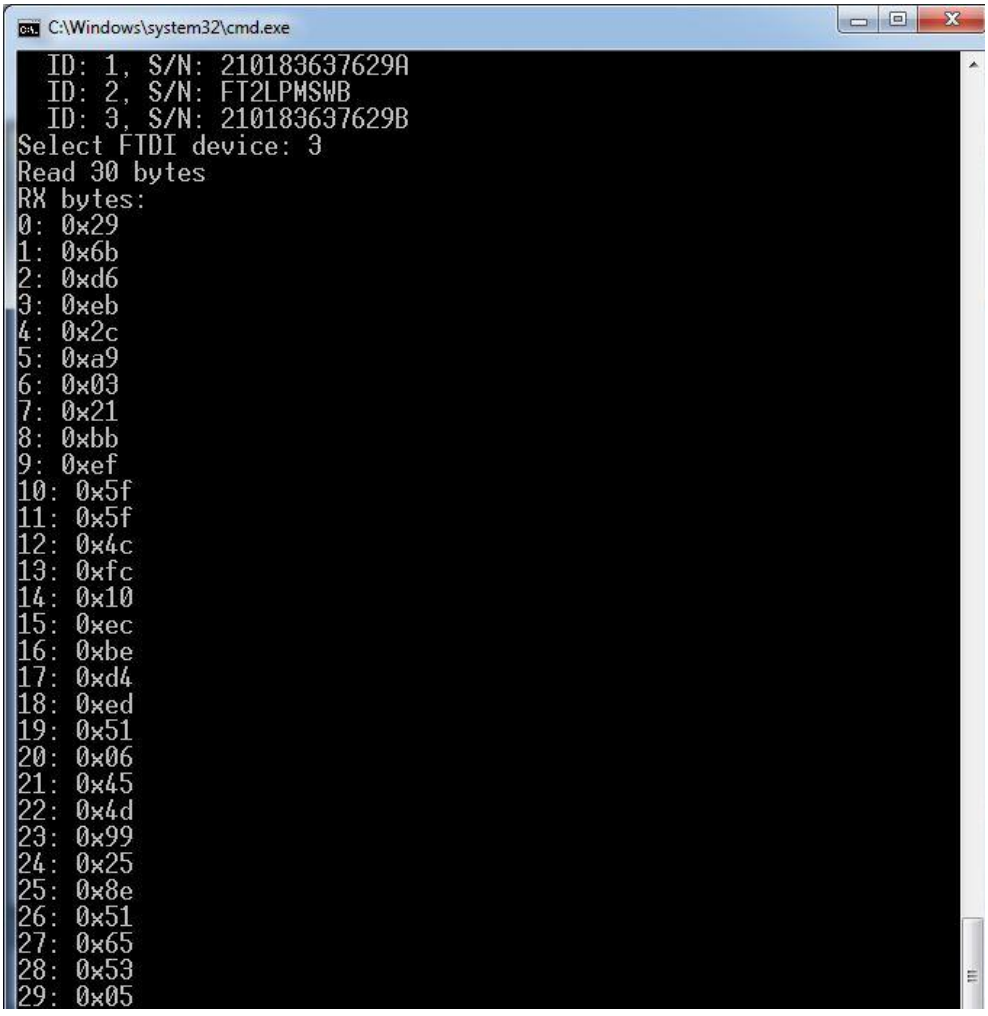


Figura 61: *Resultado de la prueba con id 16 de la fase II de la validación del sistema (sistema receptor)*

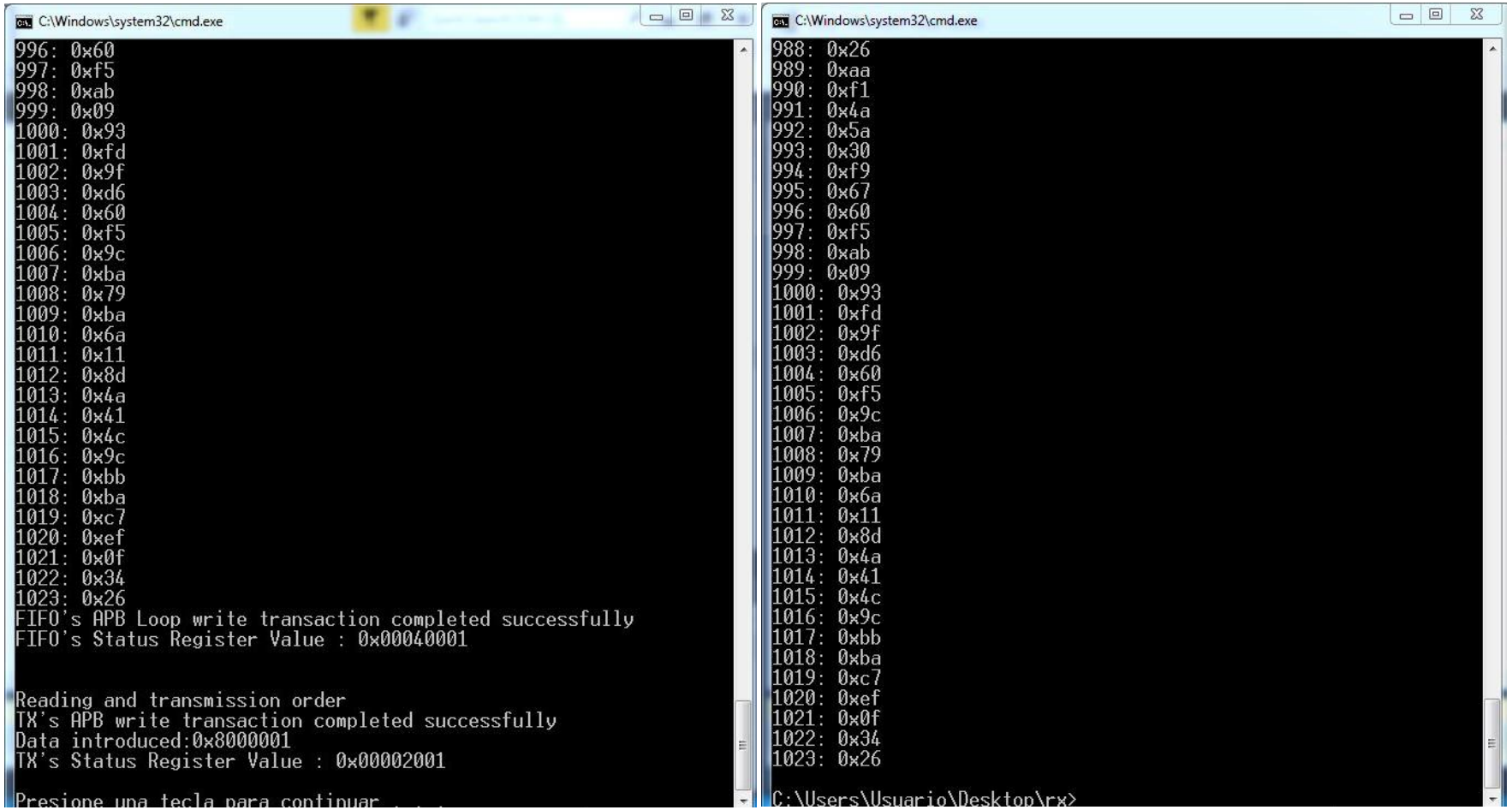


Figura 62: *Resultado de las pruebas con id 20 y 21 de la fase II de la validación del sistema (sistema transmisor izq., sistema receptor dcha.)*

Capítulo 7

Conclusiones

7.1. Conclusiones

El presente proyecto me ha servido para descubrir el gran potencial y auge de los CubeSats en el conjunto de las misiones espaciales y el abanico de nuevas posibilidades que brindan tecnologías como Zynq UltraScale+ para el desarrollo de las mismas. Otro de los principales conocimientos adquiridos ha sido la importancia que conlleva seguir una metodología de trabajo adecuada. En el caso del presente proyecto, ésta se ha aproximado a la primera etapa de desarrollo de un sistema completamente operativo para su uso en el espacio.

Previamente al inicio del diseño del sistema, se profundizó en la arquitectura de bus que se iba a emplear. En esta fase se estudió la especificación en profundidad y se crearon pequeñas arquitecturas maestro-esclavo que recogiesen todas las posibles situaciones y que sirviesen para asimilar adecuadamente los conceptos teóricos. Una vez aprendido esto, se plantearon las expectativas y requisitos a cumplir; además de los medios y herramientas a usar. Es entonces cuando se inicia la fase de diseño del sistema, en la cual se estuvo constantemente evaluando los pros y contras de cada idea aportada, generando nuevas alternativas para lograr la funcionalidad requerida y seleccionando las que se consideraron más apropiadas. Un ejemplo de esto es que, en un principio se optó por diseñar el sistema con dos buses APB, en vez de uno. El primero serviría exclusivamente para que un futuro módulo SPI se comunicase con la interfaz esclavo del módulo TX y le comandase a éste el envío de los datos. Mientras tanto, el otro bus se emplearía para el resto de operaciones del sistema. Aunque en un principió pareció una buena alternativa separar la comunicación entre ambos módulos, más tarde fue desechada la idea, debido a su ineficiencia, y finalmente se optó por un único bus de sistema.

Esta etapa de análisis, identificación y transformación de unos requisitos a un diseño en concreto ha sido el núcleo y sostén de todo el proyecto. Ha abarcado gran

parte del mismo y ha servido para descubrir la relevancia que tiene la realización de un buen diseño teórico, que se ajuste a unos requisitos específicos, antes de empezar a escribir el código en VHDL.

Posteriormente a la etapa de diseño del sistema se inició la fase de realización de mismo. Se implementó cada uno de los módulos empleando el lenguaje VHDL y se realizó una pequeña verificación de cada uno de ellos. Esta fase del proyecto fue de gran utilidad para afianzar y ampliar los conocimientos del lenguaje VHDL y del uso de la herramienta ISE Design Suite de Xilinx.

La última fase llevada a cabo fue la evaluación del sistema creado. Para ello se integraron todos los módulos diseñados y se realizó la verificación y la validación, que a continuación se describen, del sistema completo.

La verificación tuvo como finalidad probar que el sistema desarrollado cumple con los requisitos y las especificaciones técnicas establecidas. Para ello se empleó el simulador integrado en el programa ISE, ISim. Todas las pruebas a las que fue sometido el sistema permitieron comprobar la funcionalidad y también corregir determinados aspectos del mismo.

La etapa de validación estuvo compuesta por un conjunto de pruebas realizadas en un entorno lo más representativo posible al modelo a emplear. Sirvió para determinar si éste cumplía con el propósito y las expectativas, por las cuáles fue diseñado, y garantizar su correcto funcionamiento. Para llevar a cabo esta fase fue necesaria la creación de una aplicación en lenguaje C para un entorno Windows, que permitiese la comunicación con el sistema desarrollado. Debido a esto, fue necesaria la adquisición de nuevos conocimientos del lenguaje C de programación.

Pese a que el objetivo principal de las dos fases es la evaluación del sistema, ambas no cubren los mismo aspectos. La verificación comprueba que el sistema diseñado ha sido implementado correctamente; mientras que, la validación sirve para conocer si se ha construido el sistema adecuado para el entorno requerido.

A la vista de los resultados obtenidos, se determina que el conjunto del sistema implementado cumple de manera satisfactoria los objetivos planteados inicialmente y que, la comunicación entre éste y el sistema de adquisición es la adecuada. Finalmente, se llega a la conclusión de que el uso de tecnologías como ZynqUltraScale+ de Xilinx para misiones espaciales realizadas por CubeSats, es una alternativa muy interesante a tener en cuenta; ya que abre un abanico de posibilidades a nuevas aplicaciones al proporcionar un gran número de recursos y flexibilidad.

7.2. Posibles mejoras del sistema

A la vista de los resultados obtenidos, se plantean un listado de las posibles mejoras a realizar en el sistema desarrollado.

La primera de ellas sería la adición de un módulo SPI esclavo al conjunto del sistema, pero maestro respecto del bus APB. De esta forma, al implementar en el sistema de adquisición el módulo SPI maestro correspondiente, éste podría comandar directamente el envío de los datos. Esto permitiría al sistema de adquisición indicar al transmisor la cantidad de datos que quiere recibir y cancelar el envío si así lo desease. Además, permitiría al sistema receptor conocer el estado tanto del módulo transmisor, como del módulo Slave-FIFO, mediante la consulta de sendos registros de estado.

Otra posible mejora sería poder variar la tasa de transmisión de datos. Esta mejora, si se añadiese a la anterior, permitiría al sistema de adquisición indicar al módulo transmisor la tasa a la que quiere recibir los datos. Una forma de llevarlo a cabo podría ser añadir un tercer registro al módulo de transmisión en el que se especificase la tasa a la que se quiere transmitir.

También se podría evaluar el aumento de la tasa de transferencia de datos en el caso en el que continuase siendo un valor fijo, modificando la interfaz de transmisión de datos en paralelo del módulo TX.

Una alternativa más en cuanto al envío de los datos, sería que una vez comandada la orden de transmisión, el flujo de envío fuese continuado hasta que se decidiese la interrupción del mismo. Es decir, que el módulo transmisor estuviese mandando datos al sistema de adquisición hasta que la operación fuese cancelada. Quizás esta mejora sea la más difícil de llevar a cabo, pues requeriría una modificación mayor del sistema.

Finalmente e independientemente de las mejoras presentadas con anterioridad, podría aumentarse la capacidad del búfer FIFO o plantear otra forma de diseño del mismo.

Bibliografía

[1] Louise H. Crokett, Ross A. Elliot, Martin A. Enderwitz, Robert W.Stewart, " The Zynq Book", Ed. Strathclyde Academic Media (2014).

[2] ARM, Specification AMBA 3 APB Protocol v1.0, ARM Limited (2004)

[3] Garrett Shea, NASA, "NASA Systems Engineering Handbook", (2017)

[4] Bruce Yost,NASA, "State of the Art of Small Spacecraft Technology", (2018)

[5] Henry Helvajian, Siegfried W. Janson,"Small Satellites: Past, Present and Future", (2008)

[6] Hank Heidt, Jordi Puig-Suari, Augustus S.Moore, Shinichi Nakasuka, Robert J. Twiggs, " CubeSat: A new Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation", (2000)

[7] Pong P.Chu, "FPGA Prototyping by VHDL Examples", (2008)

[8] Jacques Verly, TED talk " From Sputnik to Beanie Babies & CubeSats to OUFTI", (2014)

[9] Xilinx, "Zynq UltraScale+ MPSoC Product Brief", (2016)

[10] https://www.nasa.gov/

[11] https://www.xilinx.com/

[12] http://www.cubesat.org/

[13] https://www.arm.com/