Trabajo Fin de Máster
Máster en Ingeniería Electrónica, Automática y Robótica

Diseño e Implementación de Interfaz SPI de muy bajo consumo

Autor: José Julián Rodríguez Murcia

Tutor: Fernando Muñoz Chavero

José María Hinojo Montero

Dpto. de Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Master
Master en Ingeniería Electrónica, Robótica y Automática

Diseño e Implementación de Interfaz SPI de muy bajo consumo

Autor:

José Julián Rodríguez Murcia

Tutor:

Fernando Muñoz Chavero

José María Hinojo Montero

Dpto. de Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Master: Diseño e Implementación de Interfaz SPI de muy bajo consumo

Autor: José Julián Rodríguez Murcia

Tutor: Fernando Muñoz Chavero

José María Hinojo Montero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

Agradecimientos

Me gustaría comenzar la presente memoria agradeciendo a Fernando Muñoz Chavero por brindarme la posibilidad de realizar este proyecto conjuntamente con su gurpo de investigación. Gracias a ello, he tenido la suerte de conocer a grandes personas y profesionales dentro del departamento, y estar en contacto continuo con las últimas tecnologías y una gran área de conocimiento.

Seguidamente, me gustaría dar las gracias a Jose María Hinojo por ser un fantástico tutor, manager y compañero. Su afán por transferir todo su conocimiento lo convierte en un genial profesor, su profesionalidad y sacrificio en un inmesurable ingeniero y su voluntad y empatía en una magnifica persona. Le estaré eternamente agradecido por todo lo aprendido, y por los buenos ratos pasados durante estos siete meses de trabajo.

Mencionar a mi compañero: Jorge Jimenez Sanchez, y darle las gracias por su ayuda, dedicación y aportación en el desarrollo de este proyecto. Es de detascar su amabilidad, compañerismo y paciencia infinta. Por todo esto, me llevo un gran amigo de este paso por la Universidad de Sevilla.

Gracias a mi gran amigo Alberto Garcia Salguero, desde que empezamos esta andadura de la ingeniería, siempre hemos podido contar el uno con el otro.

Por último, a mis padres por el tiempo que siempre han puesto en hacer de mi una mejor persona, y el esfuerzo e inversión para que yo solo tuviese que procuparme de mis estudios y mi trabajo. Gracias por su confianza y sus ánimos.

Resumen

Palabras Clave: SPI, QSPI, interfaz, protocolo, comunicación, consumo, velocidad, bits.

El continuo escalado de los procesos de fabricación de semiconductores supone un enorme impacto en el diseño e integración de la electrónica integrada actual. Esta evolución ha provocado el aumento de la funcionalidad dentro de los CIs, y un incremento de la complejidad; a su vez es necesario una rápida comunicación en serie que permita comunicar en bajo consumo los elementos que componen los circuitos integrados. Por esta razón, se propone el diseño, implementación y verificación de un interfaz de comunicación en serie de bajo consumo y alta velocidad (QSPI) mediante el lenguaje de descripción hardware (VHDL).

En primer lugar, se diseñará el sistema de comunicación modular que permitirá integrar la doble modalidad de comunicación en serie de baja velocidad (SPI) y de alta velocidad (QSPI), usando técnicas de reducción de consumo en circuitos digitales tales como Clock Gatin y Power Gating. Tras el diseño y la implementación, se desarrollará una serie de tests con varias técnicas de validación que nos permitirán cuantificar los resultados obtenidos en las diversas simulaciones.

Por último, centraremos nuestro trabajo en la síntesis de nuestro sistema y en la creación del esquemático resultante, y una comprobación de la funcionalidad obtenida sobre este.

Abstract

The continuous escalation of semiconductor manufacturing processes has a huge impact on the design and integration of current integrated circuits. This evolution has caused an increase in functionality within ICs, and an increase in complexity; At the same time, fast serial communication is necessary to communicate the elements which make up the integrated circuits in low consumption. For this reason, the design, implementation and verification of a low-power and high-speed serial communication interface (QSPI) through the hardware description language (VHDL) is proposed.

In the first place, the modular communication system will be designed that will allow the integration of the double low speed and high-speed serial communication mode: SPI y QSPI, using consumption reduction techniques in digital circuits such as Clock Gatin and Power Gating After the design and implementation, a series of tests will be developed with several validation techniques that will allow us to quantify the results obtained in various simulations.

Finally, we will focus our work on the synthesis of our system and the creation of the resulting schematic, and a check of the functionality obtained on it too.

Índice

INDICE DE FIGURAS

INDICE DE TABLAS

1 INTRODUCCIÓN

En el panorama actual, la necesidad de transmitir información entre dos dispositivos, ya sea por un medio físico o inalámbrico, es vital. Un claro de ejemplo de esta necesidad o tendencia se muestra en los actuales sistemas emportrados, donde se conectan diferentes dispositivos como, por ejemplo, memorias no volátiles para almacenar los datos obtenidos, sensores digitales como acelerómetros, wake-up radios o transceptores de comunicaciones a un único microcontrolador. Aunque es posible seleccionar opciones donde todos estos elementos hagan uso de una misma interfaz de comunicaciones, en muchas ocasiones no es posible. Esto provoca que en un mismo sistema coexistan varias interfaces de comunicación, aumentando la complejidad del sistema (cada interfaz debe ser configurada correctamente y procesada de forma diferente) y el área ocupada. A cada dispositivo debe ser conectado a su correspondiente interfaz, por lo que es necesario disponer de espacio suficiente como para coenctar todas las líneas necesarias, así como evitar posibles cruces entre diferentes interfaces con el objetivo de minimizar las interferencias. Esto último es vital cuando se tratan de interfaces de comunicación de alta velocidad.

Por su parte, las interfaces de comunicación toman aún más importancia si nos introducimos en el ámbito de la microelectrónica, donde la tranmisión de datos tienen un papel crucial entre los distintos módulos que forman los dispositivos. En este nuevo escenario, además es importante considerar que el área es un criterio de diseño de gran importancia. Por este motivo, es importante seleccionar una interfaz de comunicación versátil, que permita la comunicación de múltiples dispositivos de forma sencilla y que presenta una alta tasa de transmisión de datos, de forma que pueda afrontar los nuevos sistemas que están por desarrollar.

1.1 Motivación

Durante la elaboración de un chip de señal mixta para uno de los proyectos de investigación que lleva a cabo el Grupo de Ingeniería Electrónicca (GIE) en el Departamento de Electrónica de la Universidad, surge la necesidad de incorporar una interfaz de comunicación serie de alta velocidad y bajo consumo a dicho chip. Para solventar esta problemática, se propone el desarrollo e implementación de una interfaz de comunicación serie o *Serial Peripheral Interface (SPI)*. De esta forma, se establece así un protocolo de comunicación bien definido entre dos dispositivos siguiendo un esquema de conexión tipo maestro-esclavo.

En relación a la creación de una interfaz de comunicación de alta velocidad, existen dos posibles alternativas desde el punto de vista del diseño. Por una parte, se puede incrementar la frecuencia de reloj del módulo de forma que el tiempo de bit se reduzca. De esta forma, conseguimos elevar la tasa de transferencia al ser capaces de transmitir datos en un menor tiempo. Sin embargo, esta opción presenta dos problemas:

- No todas las interfaces soportan frecuencias de trabajo variables, por lo que la solución no sería compatible con todos los dispositivos existentes en el mercado.
- Al incrementar la frecuencia de reloj, la potencia dinámica disipada por el circuito se ve incrementada,

ya que esta es directamente proporcinal a dicha frecuencia.

Por tanto, el aumento de la frecuencia de reloj compromete uno de los requisitos de diseño, el bajo consumo.

Por otro lado, la tasa de transmisión de datos se puede incrementar aumentando el número de canales que se pueden transmitir de forma paralela, es decir, si aumentamos el número de canales, para un mismo instante de tiempo, se puede transmitir un mayor número de datos. De esta forma, no es necesario incrementar la frecuencia de operación.

Este enfoque alternativo es el utlizado por la interfaz conocida como *Quad Serie Peripheral Interface* o QSPI. Una versión del protocolo SPI adaptada a comunicaciones que requieren una alta tasa de transferencia como son las memorias no volátiles. Como desventaja, la interfaz de comunicación ve incrementada su complejidad ya que ahora requiere procesar varías líneas de comunicación de forma paralela.

Además, el diseño de esta interfaz debe corresponderse con un diseño orientado al bajo consumo. Esta restricción es una consecuencia directa de la tendencia de la electrónica de consumo hacia el diseño de dispositivos portables (aquellos que operan a baterías y permiten ser transportados fácilmente) o, más recientemente, sistemas capaces de obtener energía del medio que les rodea, eliminando el uso de las baterías. Por tanto, la reducción en el consumo de potencia se ha convertido en un objetivo prioritario en el diseño microelectrónico.

Durante muchos años la velocidad y el área fueron las restricciones dominantes, y el problema del consumo estaba reservado a nichos de aplicaciones muy específicas como los relojes pulsera, marcapasos, audífonos, calculadoras portátiles, circuitos de aplicación militar y algunos sistemas alimentados a baterías. El crecimiento explosivo de los dispositivos portátiles: PDAs, laptops, GPSs, y sobre todo del gran crecimiento de la telefonía móvil y los sensores inalámbricos han convertido el diseño de circuitos de bajo consumo en una de las principales exigencias en el diseño.

Por esta razón, uno de los puntos que nos motiva a desarrollar este proyecto es la introducción de técnicas que permitan alcanzar una reducción del consumo, permitiendo el desarrollo de circuitos integrados de gran eficiencia y la prolongación de la vida útil de las baterías lo máximo posible. Además, cabe destacar que el menor consumo en dispositivos electrónicos puede ser clave en la viabilidad del prototipo a desarrollar en el marco del proyecto de investigación.

Por todo esto, se propone una interfaz que reduzca el consumo de nuestro dispositivo, utilizando técnicas conocidas como *clock gating* o arquitecturas orientadas a bajo cosnumo. Para este último caso, el diseño de la interfaz se ha concebido como un sistema modular, es decir, se ha dividido nuestro sistema en bloques según su funcionalidad, con el fin de habilitar únicamente aquellos módulos que deben funcionar. De esta forma, se busca minimizar la contribución de la potencia dinámica consumida por el sistema.

Este sistema pone solución a la última vertiente de innovación y desarrollo de la tecnología: velocidad; la nueva interfaz a desarrollar no solo reduce el consumo si no que también incrementa la velocidad de

transmisiónde datos, al permitir una comunicación bidireccional por sus cuatros canales. Por lo tanto, tal y como se ha comentado, este diseño de interfaz engloba todas las inquietudes de la electrónica actual y de las últimas décadas.

1.2 Objetivos

Tras haber presentado lo que nos motiva a realizar este trabajo, los hitos que se quieren alcanzar con el desarrollo de este sistema son los siguientes:

- **Diseño de un módulo esclavo que cumpla con las restricciones indicadas en el punto anterior.** Este módulo debe ser capaz de soportar tanto una comunicación serie de baja velocidad (modo SPI) como de alta velocidad (Quad SPI).
    - **Desarrollo SPI.** Se plantea el diseño de una primera interfaz modular, que nos sirva como base para el desarrollo del sistema QSPI. La interfaz de comunicación se dividirá en varios modulos y será integrado en un modulo superior.
    - **Desarrollo QSPI.** Implementación de la segunda interfaz de comunicación a partir de los módulos ya desarrollados en el primer sistema. Será necesario una adaptación del SPI al sistema de transmision de cuatro líneas.
- **Implementación física de dicho módulo.** Tras la descripción hardware del sistema, se da paso a la síntesis y creación del esquemático de la interfaz.
- **Verificación.** Ambos módulos deben ser comprobados y verificados. Se tendrá en cuenta la correcta comunicación entre master y slave, así como los diversos errores que se puedan producir durante la transmision. Creación de un sistema de pruebas y de tests que chequen el correcto funcionamiento de la interfaz.

1.3 Organización

Este documento se dividirá en cuatro grandes partes que nos permitirán describir el trabajo realizado y la tecnología sobre la que nos basamos para el desarrollo de este sistema.

En primer lugar, se describirá otros sistemas de comunicación en serie, teniendo en cuenta su protocolo y la manera que tienen singularmente de transmitir la información. Además, se realizará una comparación obteniendo así sus ventajas y desventajas; y concluyendo porque se elige el sistema empleado. También, se explicarán las técnicas de bajo consumo, la importancia que tienen y las ventajas que permiten conseguir en la electrónica digital. De esta manera, se cierra el primer capítulo donde se presenta el estado del arte.

A continuación, se detallará la arquitectura de nuestro sistema de comunicación, incluyendo una descripción de la funcionalidad de cada bloque y el bloque top que lo integrá.

Tras este capítulo, se dará paso a la verificación y validacíón del sistema; donde se mostrarán diversos cronogramas y métricas de la secuencia de comunicación; así como los tiempos y polaridad del protocolo. Se compararán los resultados obtenidos con las métricas que nos impone el propio protocolo.

Por último, se darán las conclusiones del trabajo implementado y de los resultados que se han obtenido. Se propondrán nuevas líneas en las que trabajar y continuar con lo ya conseguido.

2 INTERFACES DE COMUNICACIÓN

Dentro de los sitemas empotrados, así como en el desarrollo de sistemas microelectrónicos, existen multitud de interfaces y protocolos de comunicación. Se entiende por una interfaz de comunicación, aquel circuito físico que, a través de uno o varios puertos, envía o recibe señales desde o hacia un sistema o subsistema, respectivamente. Como consecuencia de la gran hetereogenidad de sistemas y dispositivos de los que disponemos en la actualidad, no existe una única interfaz universal. Cada una de ellas establecen una capa física y de enlace concretas que imposibilitan la interoperabilidad entre diferentes interfaces. Esto provoca que los dipositivos deban presentar la misma interfaz de comunicación tanto en origen como en destino.

Si la interfaz es el sistema que permite la transmisión de información, el protocolo es el lenguaje con que se comunican los dispositivos en la red. En otras palabras, la manera en la que los dispositivos de una red intercambian información. Un claro ejemplo sería el intento de comunicación fallido de dos computadores conectados en la misma red, pero que intentan comunicarse por dos protocolos. Por esta razón, es necesario que ambas partes "hablen" el mismo idioma, es decir, tengan el mismo protocolo de comunicación.

Sin ir más lejos, con la finalidad de que cualquier computador conectado a Internet pueda transmitir y recibir información con otro, se creó el protocolo TCP/IP habilitando la comunicación entre ordenadores.

La transmisión de información a través de una interfaz de comunicación entre dos sistemas se realiza transmitiendo de manera binaria palabras de 8, 16 o 32 bits. Existen dos métodos principales para realizar la transmisión:

- **Método Serie**. Transmisión secuencial de todos los bits que conforman la transmisión haciendo uso del mismo puerto. Cabe destacar que cada interfaz establece por qué bit comienza, el más o menos significativo.
- **Método Paralelo**. Transmisión simultánea de todos los bits que componen la transmisión haciendo uso de diferentes puertos. En este método, la transmisión va acompañada de una señal de sincronización que indica el momento en el que receptor puede capturar los datos. El objetivo es evitar la desincronización en la transmisión, evitando que se produzcan errores.

Séra en el primer método donde nos centremos, debido al alto grado de estandarización que ha alcanzado desde hace tiempo, y a que nuestro sistema a desarrollar se comunica de esta manera.

Por otro lado, es un método con aspectos lógicos de codificación y sincronización muy bien definidos, lo que facilita la implementación y descripción de los circuitos digitales.

2.1 Interfaces de Comunicación en Serie

Centrándonos en el sistema que vamos a desarrollar, sería de gran utilidad describir los principales sistemas que destacan dentro de este grupo. Dejando atrás las interfaces de computación más conocidas en la actualidad,

como son el sistema USB o Ethernet; nos centraremos en los protocolos serie que se han creado para satisfacer las necesidades de los sistemas empotrados y, en concreto, en los sistemas integrados.

Dentro de los protocolos más extendidos a día de hoy, encontramos el protocolo *Inter-Integrated Circuit* (I2C) o *Serial Peripheral Interface* (SPI). En adelante, ambos protocolos serán identificados en el texto mediante el uso de sus siglas inglesas. Ambos protocolos fueron diseñados para establecer una comunicación serie entre sistemas integrados, tales como sensores o sistemas de almacenamiento, y sistemas microcontroladores. Cabe destacar que ambos protocolos no son excluyentes, por este motivo todos los sistemas microcontraladores/microprocesadores presentan interfaces de comunicación que soportan ambos protocolos. Esto provoca que gocen de una gran penetración en el sector de los sistemas empotrados hoy, debido a sus bondades. Estas serán descritas posteriormente en este capítulo.

Durante las fases inciales de la definición de estos protocolos, encontramos dos compañías importantes: Motorola, responsable del desarrllo de la interfaz SPI, y Philips para I2C, [1], y dos historias diferentes sobre por qué, cuándo y cómo se crearon los protocolos. La primera, narra como el SPI consiguió introducirse en el primer controlador del Motorola 68000 en 1979 y se empleó en el ordenador Apple Macintosh en 1984. Para ambos casos, el SPI se definió como un bus compuesto de cuatro líneas que permitiese la interconexión de los diferentes periféricos externos al microcontrolador, garantizando un enlace de alta velocidad, robusto y simple.

Por otro lado, el sistema I2C fue desarrollado en 1982 y su principal función consistía en establecer una conexión simple entre la CPU y los peiféricos externos, al igual que en el SPI. Sin embargo, con el objetivo de reducir el número de líneas necesarias para establecer la comunicación, se empleó un canal bidireccional. De esta forma, se recicla la misma línea para transmitir y recibir. Como consecuencia, el protocolo I2C no permite una comunicación *full-duplex.*

Los laboratorios de Philips en Eindhoven inventaron el "circuito inter-integrado", IIC o I2C consiguiendo así que únicamente con dos cables se puedan conectar todos los periféricos al microcontrolador. Incialemnte, se consiguión una una velocidad de transmisión de 100 kb/s. Tras las continuas renovaciones de este sistema, se consigue alcanzar una velocidad máxima de 3.4 Mb/, alcanzando así una transmisión de datos con los periféricos mucho más alta.

Como curiosidad en 2016, aunque la implementación del protocolo I2C no requería de honorarios, y tenía una política abierta, sí era necesario el abono de unas tasas para distribuir las direcciones de este protocolo. Actualmente, NXP y Freescale están a cargo de las especificaciones de estos protocolos, aunque estos se hayan convertido tan populares que habría que preguntarse si han pasado a ser públicos.

2.1.1 *Serial Peripheral Interface* (SPI)

La intefaz SPI es un protocolo de comunicación basado en la existencia de único maestro. Esto supone que sólo un dispositivo es el responsable de iniciar todas las comunicaciones con el resto de periféricos y/o sensores, denominados esclavos. Además, esta interfaz tiene bien definidas sus características para lograr una

comunicación síncrona, de alta velocidad y simple. La interfaz de comunicación está compuesta de las siguientes cuatro líneas:

- ***Serial Clock* (SCLK).** Señal de reloj enviada desde el maestro a todos los esclavos con el fin de generar la señal de sincronismo que permita capturar los datos transmitidos a la vez que la generación de los datos de recepción.
- ***Chip Select* (CS).** Señal que permite seleccionar el dispositivo esclavo con el que se comunicará el maestro.
- **Master Out -Slave In (MOSI).** Línea de transmisión de datos del dispositivo maestro al esclavo. Sirve para enviar comandos.
- **Master In – Slave Out (MISO).** Línea utilizada por el dispositivo maestro para recibir los datos enviados por el esclavo.

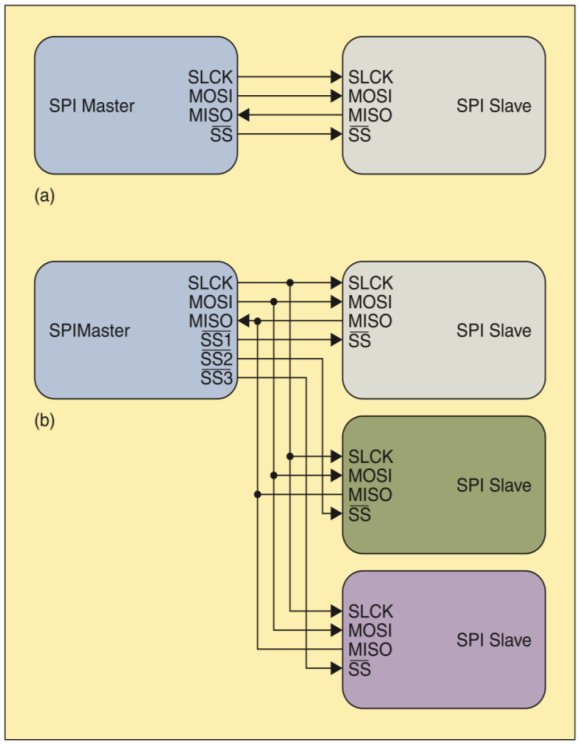


Figura 1: Ejemplo de conexión de un sistema basado en el protocolo SPI. [1].

A la hora de establecer una comunicación, el dispositivo maestro activa la señal CS del esclavo, generalmente activa a nivel bajo, con el que desea intercambiar información. Este intercambio puede corresponderse con una orden de escritura (se desea modificar el contenido de algún registro del esclavo) o de lectura (el maestro necesita acceder a la información que alberga el esclavo).

La comunicación entre maestro y esclavo puede realizarse de cuatro modos diferentes. Cada modo viene determinado por unos parámetros: polaridad del reloj (CPOL) y la fase del reloj (CPHA). Por un lado, la polaridad del reloj hace referencia al nivel lógico al que permanece la línea SCLK para indicar que el enlace no está activo. Por otra parte, la fase del reloj hace mención al flanco de la línea SCLK en el que el maestro escribe

en la lína MOSI y recibe información por la línea MISO. Los cuatros modos de comunicación se definen a continuación:

- **Modo 0 (Polaridad = 0, Fase = 0)**. Los datos se muestrean en los flancos de reloj ascendente.
- **Modo 1 (Polaridad = 0, Fase = 1).** A diferencia del modo anterior, la polaridad del reloj sigue igual, mientras que la fase cambia, por lo que el muestreo de datos se produce medio ciclo de reloj después.
- **Modo 2 (Polaridad = 1, Fase = 0).** Al cambiar la polaridad, el muestreo se realiza en los flancos de reloj descendentes.
- **Modo 3 (Polaridad = 1, Fase = 1).** El muestreo de datos se produce medio ciclo de reloj después de cada flanco descendente.

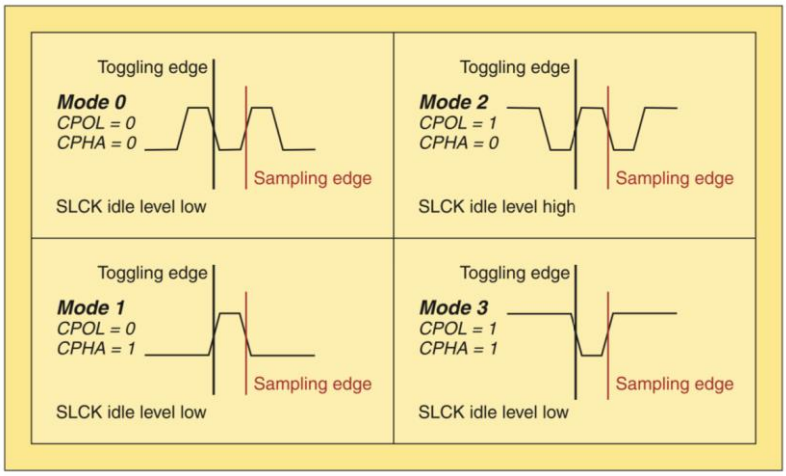


Figura 2: Modos de funcionamiento de la interfaz SPI. [1]

Es importante resaltar que la dupla maestro - esclavo debe usar la misma configuración de parámetros para que la comunicación se lleve a cabo. Si múltiples esclavos se usan en la comunicación y presentan diferentes configuraciones, el maestro tendrá que reconfigurarse cada vez que necesite comunicarse con uno de ellos.

Por otro lado, el protocolo del SPI es muy sencillo y no define ninguna velocidad de datos máxima ni ningún esquema de direccionamiento particular; no tiene un mecanismo para confirmar la correcta recepción de datos. Asimismo, no dispone de mecanismos para el control de flujo de datos. Esto último, recae en el usuario. En realidad, el maestro SPI no tiene conocimiento de si existe un esclavo, a menos que se haga "algo" adicional fuera del protocolo SPI. Además, no tiene en cuenta las características de la interfaz física como los voltajes de E/S y el estándar utilizado entre los dispositivos. Inicialmente, la mayoría de las implementaciones de SPI usaban un reloj no continuo y un esquema byte a byte, aunque muchas de las variantes que ahora existen usan una señal de reloj continua y una longitud de transferencia arbitraria.

2.1.2 QSPI

A partir del protocolo SPI, aparece un sistema que mejora con creces la velocidad de transmisión de datos entre un dispositivo maestro y un esclavo. Para ello, las líneas de datos MISO y MOSI de la interfaz SPI se transforman en líneas bidireccionales, a la vez que se añaden dos nuevas líneas; también bidireccionales. Esto permite que maestro y esclavo queden conectados por 4 líneas de datos que pueden usarse para transmitir o recibir información. Además, también supone un cambio en la funcionalidad del sistema, y la necesidad de tener nuevas señales de control y configuración. Las principales diferencias son:

- **Cuatro señales de transmisión**. Se requieren de cuatro nuevas señales de transmisión de bits bidireccionales. A diferencia del modo simplifcado SPI, con dos únicas líneas unidireccionales, el QSPI tiene 4 líneas que permiten transmitir datos en ambas direcciones. Esto permite transmitir datos en
- **Señales de control**. Para constituir en un mismo sistema las dos interfaces, es necesario que se añadan nuevas señales que permitan configurar ambos métodos correctamente:
    - Señal RWN. La señal indica si la operación es de lectura o escritura, con el fin de elegir la dirección de las líneas de transmisión. En el sistema SPI no era necesario la incorporación de esta señal, ya que las líneas de transmisión tenían una única dirección.
    - Señal Modo. Permite cambiar entre los dos modos de funcionamiento, debe quedar explícito el modo en el que el sistema está trabajando, ya que dos de las líneas quedan en alta impedancia.
- **Periodo Dummy**. Existe un periodo en el cual las líneas cambian de dirección. Se configuran una serie de ciclos que habilitan el cambío de dirección y detienen la transmisión de bits. Es habitual que se efectue tras la escritura del master de lo bits de instrucción o CMD.

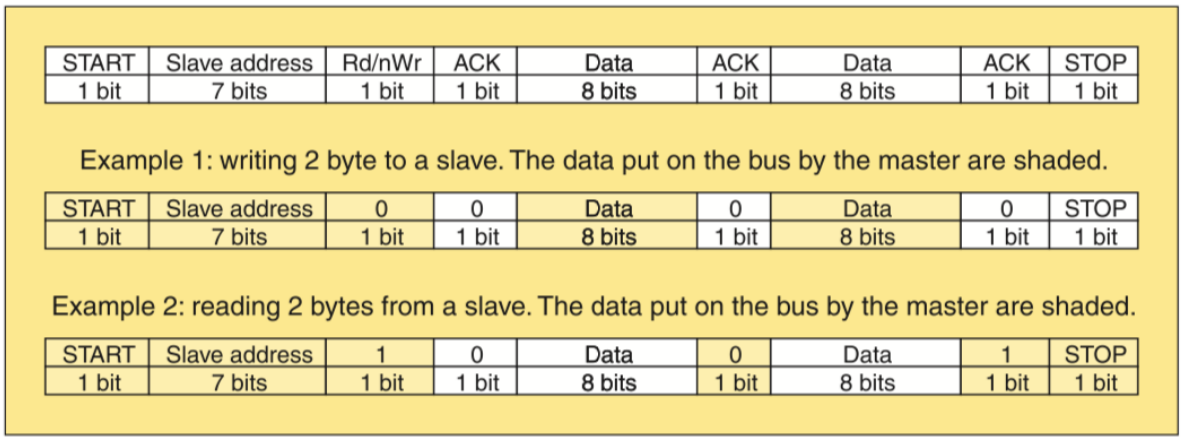
Salvando estas diferencias, la arquitectura del protocolo QSPI es similar al SPI. Por tanto, sigue trantándose de un protocolo sencillo y robusto. Cuenta con el aumento de velocidad de transferencia ya que permite enviar un byte en dos ciclos del reloj SCK, mientras que el SPI necesitaría 8 ciclos de este reloj. La elección de los esclavos se produce de igual manera con la bajada de la señal SS de cada uno de ellos.

Finalizando, el sistema QSPI no supone grandes cambios en la funcionalidad general de la arquitectura del sistema, ya que el protocolo continúa siendo el mismo, y el funcionamiento en la comunicación entre maestro y esclavo es idéntica. Solo supone un extra en los recursos al añadir dos nuevas líneas de transmisión, y la necesidad de dos nuevas señales para el control de la transmisión.

2.1.3 I2C

El I2C es un protocolo multi-maestro que utiliza únicamente dos líneas para su comunicación. Estas se denominan *Serial Data*, SDA, y *Serial Clock*, SCL. Ambas líneas de transmisión son bidireccionales. Este sistema se caracteriza por no necesitar seleccionar al chip esclavo o la lógica arbitraria. Además, prácticamente cualquier número de esclavos y cualquier número de maestros pueden conectarse a estas dos líneas de señal y comunicarse entre sí mediante la transmisión de tramas. La Figura 3 representa el formato de la trama. En ella, se distinguen los siguientes elementos:

- Un bit de comienzo denominado "*Start*".
- Siete bits de direccion asociado al esclavo. Cada dispositivo conectado al bus tiene una dirección única.
- Un bit para indicar el tipo de operación, si se corresponde con una lectura o escritura.
- Un bit de asentimiento.
- Datos divididos en palabras de 8 bits.
- Un bit destinado a indicar el fin de la comunicación.



START	Slave address	Rd/nWr	ACK	Data	ACK	Data	ACK	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 1: writing 2 byte to a slave. The data put on the bus by the master are shaded.

START	Slave address	0	0	Data	0	Data	0	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 2: reading 2 bytes from a slave. The data put on the bus by the master are shaded.

START	Slave address	1	0	Data	0	Data	1	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Figura 3: Protocolo de escritura/lectura I2C [1]

La especifiación del protocolo I2C establece que el circuito integrado que transfiere datos por el bus es considerado el Maestro. En consecuencia, todos los demás circuitos integrados se consideran dispositivos esclavos.

Primero, el maestro emite una condición de "inicio" que actuará como una señal de "atención" a todos los dispositivos conectados. Todos los dispositivos conectados al bus escucharán los datos escritos. A continuación, el maestro envía la "dirección" del dispositivo al que desea acceder, junto con una indicación de si se corresponde con una operación de "lectura" o "escritura". Tras la recepción de la dirección, cada dispositivo comprueba si la dirección recibida corresponde con la suya. Si no es así, los esclavos esperan hasta la condición de parada. Si, en caso contrario, la dirección coincide, emitirá una respuesta de asentimiento, ACK.

Una vez que el maestro recibe esta señal, puede empezar a transmitir o recibir datos. En el caso que mostramos en la Figura 4 el maestro transmitirá datos, ya que es una operación de escritura. Cuando todo termine, el maestro emitirá la condición de parada, STOP. Esta señal avisa a los dispositivos que la comunicación anterior ha finalizado y por lo tanto dispositivos se mantienen a la espera de otra transmisión.

Cuando un maestro desea recibir datos de un esclavo, operación de lectura, se procede de forma similar a excecpción del bit RD/nWR. Este debe fijarse a 1. Una vez que el esclavo ha reconocido la dirección, comienza a enviar los datos solicitados, byte a byte. Después de cada byte de datos, corresponde al maestro asentir los datos recibidos.

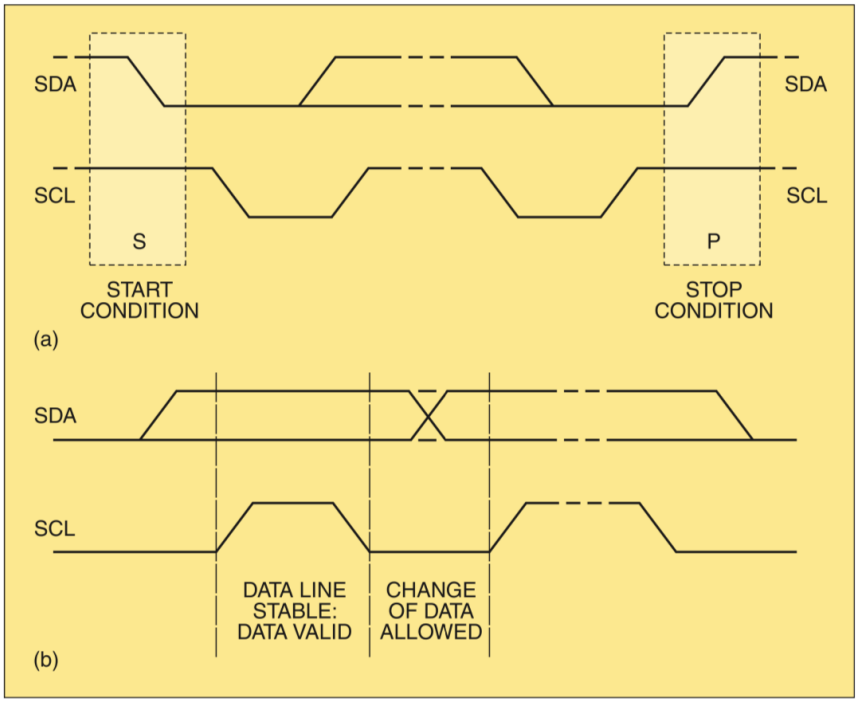


Figura 4: Descripción de trama I2C [1]

La especificación I2C establece que los datos solo pueden cambiar en la línea SDA si la señal de reloj SCL está en un nivel bajo; por el contrario, los datos en la línea SDA se consideran estables cuando SCL está en estado alto, como muestra la siguiente figura.

Donde este estándar es verdaderamente inteligente es en la asociación de la capa física y el protocolo anteriormente descrito que permite la comunicación entre cualquier número de dispositivos con tan solo dos líneas de transmisión. Un claro ejemplo, puede venir dado si dos dispositivos están simultáneamente intentando poner información en SDA y/o SLC. Electricamente, no habría conflicto si múltiples dispositivos intentan poner cualquier nivel lógico en las líneas de transmisión simultaneamente. Si uno de los drivers intenta escribir un valor lógico 0 y el otro 1, entonces la estructura “open-drain” y “pull-up” asegura que no habrá cortocircuito y el bus realmente verá un CERO lógico en tránsito en el bus. En otras palabras, en cualquier conflicto, la lógica CERO siempre "gana".

La implementación físca del bus también permite que los dispositivos maestros escriban y escuchen simultáneamente las líneas del bus. De esta manera, cualquier dispositivo puede detectar colisiones. En caso de conflicto entre dos maestros (uno de ellos tratando de escribir un CERO y el otro un UNO), el maestro que gana el arbitraje en el bus ni siquiera sabrá que ha habido un conflicto: solo el maestro que pierde sabrá, porque tiene la intención de escribir un lógico UNO y lee un CERO lógico. Como resultado, un maestro que pierde el arbitraje en esta interfaz parará de intentar acceder al bus. En la mayoría de casos, solo retrasará su acceso e intentará el mismo acceso más tarde.

Otra ventaja que presenta el protocolo I2C es la ayuda al lidiar con problemas de comunicación. Cualquier dipositivo de esta interfaz permenece escuchando constantemente. Los maestros en I2C que detecten una condición de START esperarán hasts que se detecte un STOP para intentar un nuevo acceso. Los esclavos en

I2C decodificarán la dirección del dispositivo que sigue la condición START y comprobarán si coincide con la suya. Todos los esclavos que no estén direccionados esperarán hasta que se emita una condición de STOP antes de volver a escuchar el bus.

Por último, si algo no funciona correctamente, el dispositivo que estuviese hablando con el bus, lo sabría comparando lo que envía con lo que se ve en el bus. Si se detecta una diferencia, se debe emitir una señal de STOP que libere el bus.

2.1.4 Otras Interfaces

Dejando atrás los dos principales sistemas de comunicación en sistemas empotrados, cabe destacar la relevancia de otras interfaces que, asociadas a los avances en tecnología, han permitido la conexión de una gran cantidad de sensores para la medición y el control distribuidos en diversas aplicaciones.

2.1.4.1 LVDS Interface

El reciente auge de procesadores de alton nivel, realidad virtual y redes ha demandado mucha más banda ancha que antes; imponiendo de esa manera grandes desafíos como la capacidad de transferir datos rápidamente en sistemas de bajo consumo y mediante soluciones que superan el cuello de botella que existe actualmente en la capa física de otros estándares de comunicación.

El sistema de señal diferencial de bajo voltlaje (LVDS) se caracteriza por ser un sistema de transmisión de señales diferenciales de alta velocidad (155.5 Mbps). El esquema diferencial tiene una tremenda ventaja sobre esquemas de un solo extremo ya que es menos susceptible a los comunes ruido de modo El ruido acoplado a la interconexión se ve como modulaciones de modo común por los receptores y se rechaza.

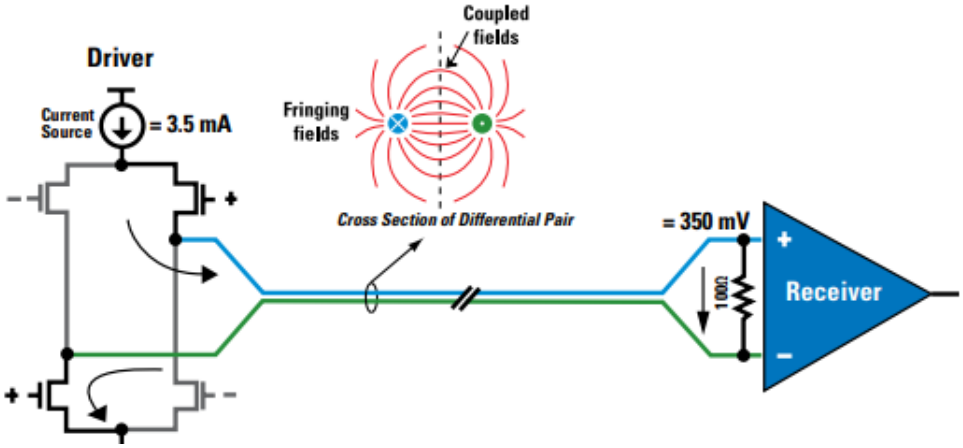


Figura 5: Esquema LVDS Interfaz

Como aparece en la imagen anterior, el transmisor introduce una corriente constante en los cables que conectan con el receptor con una dirección de corriente que marca el nivel lógico digital. Cuando la corriente llega al receptor, esta pasa por una resistencia de aproximadamente 100 ohmios, creando así una diferencia de

voltaje que, según su polaridad, el receptor detectará el nivel lógico de la comunicación.

Mientras exista un fuerte acoplamiento de campo eléctrico y magnético entre los dos cables, el LVDS reduce la generación de ruido electromagnético. Esta reducción de ruido se debe al flujo de corriente igual y opuesto en los dos cables que crean campos electromagnéticos iguales y opuestos que tienden a anularse entre sí. Además, los cables de transmisión estrechamente acoplados reducirán la susceptibilidad a la interferencia de ruido electromagnético porque el ruido afectará igualmente a cada cable y aparecerá como un ruido de modo común. El receptor LVDS al detectar el voltaje diferencial con ruido común no se ve afectado por los cambios de voltaje.

El bajo voltaje diferencial, alrededor de 350 mV, hace que el LVDS consuma muy poca energía en comparación con otras tecnologías de señalización.

LVDS funciona tanto en transmisión de datos en paralelo como en serie. En transmisiones paralelas, múltiples pares diferenciales de datos transportan varias señales a la vez, incluida una señal de reloj para sincronizar los datos. En las comunicaciones en serie, se dispone en serie múltiples señales en un solo par diferencial con una velocidad de datos igual a la de todos los canales combinados de un solo extremo. Por ejemplo, un bus paralelo de 7 bits de ancho serializado en un solo par que operará a 7 veces la velocidad de datos de un canal de un solo extremo. Los dispositivos para convertir datos en serie y en paralelo son el serializador y el deserializador, abreviado a *SerDes* cuando los dos dispositivos están contenidos en un circuito integrado.

2.1.4.1 UART

El receptor/transmisor asíncrono universal (UART) es el dispositivo principal de un sistema de comunicaciones serie. Su función principal es convertir los datos serie a paralelos cuando se reciben datos, y de convertir los datos en parelelo a serie para su transmisión.

En la Figura 6, se muestra un esquema general con los bloques básicos de un UART. En este esquema, se distinguen los registros de datos, tanto de recepción como de transmisión y sus registros de desplazamiento, los registros de control de transmisión y recepción y señales de sincronización para el comienzo/finalización de la comunicación (RTS, CTS).

Esta interfaz UART tienen otras aplicaciones como las señales handshaking que son necesarias para otras interfaces, como la RS-232 que comentaremos en el siguiente punto.

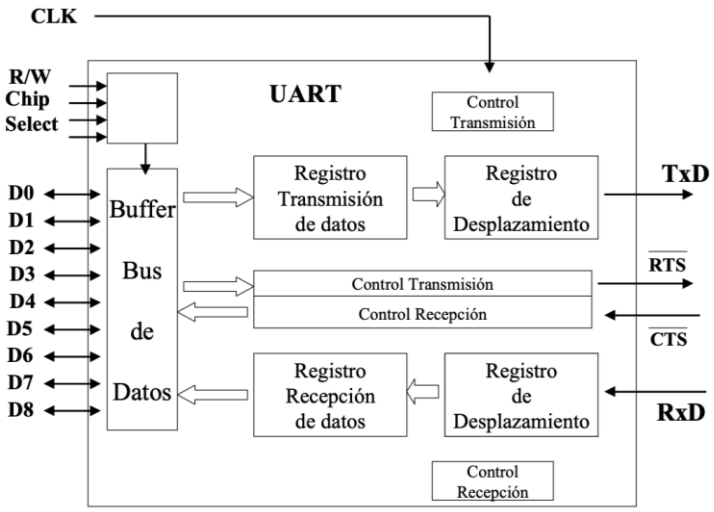


Figura 6: Esquema general UART.

Para un correcto funcionamiento de este sistema se ha de cumplir una serie de restricciones:

- Sincronización entre el receptor y el transmisor.
- Codificación de los datos.
- Prioridad en el envío de los bits.
- Tasas de envío de datos.
- Señales handshaking.
- Señales eléctricas de los valores lógicos.

El protocolo a seguir dentro de esta interfaz viene definido por una serie de pautas que se reflejan en la Figura 7. Es necesario un bit de comienzo que marque el incio de la transmisión, tras este bit se transmiten los datos a enviar, cuyo tamaño puede variar entre entre los 5 y 9 bits. A la hora de transmitir la información, esta siempre empieza por el bit menos significativo, LSB. Por último, se envía un bit de parada (Stop Bit). Además, cabe la posibilidad de añadir un bit de paridad para detectar si se ha producido algún error durante la comunicación.

Figura 7: Trama de comunicación RS-232.

2.1.4.2 RS-232

El estándar EIA/TIA-232-E, o más conocido RS-232, es uno de los más conocidos protocolos en serie que reconoce el estándar. La base de este protocolo se fundamenta en la transmisión serial de bits entre el sistema host (DTE) y el sistema periférico (DCE).

La interfaz no está diseñada para obtener velocidades altas, no más de 20 Kb/s, aunque puede conseguir velocidades mayores. La interfaz puede trabajar en comunicación asíncrona y síncrona, y en ella, se pueden distinguir distintos tipos de canal:

- Simplex. En este caso los datos solo viajan en una dirección; por ejemplo, desde el DTE hacia el DCE.
- Half-Duplex. Los datos pueden viajar en una u otra dirección, pero sólo durante un cierto periodo de tiempo.
- Full-Duplex. Los bits pueden ser transmitidos en ambos sentidos de manera simultánea.

Debido a la versatilidad de sus tipos de canal, se incluyen las líneas de handshaking como método para resolver este problema e indicar en que dirección deben viajar en un instante determinado.

Para los propóstios de la RS-232 estándar, se puede definir una conexión por un cable desde un dispositivo a otro. El conector DB-25 permite 25 conexiones en la especifiación completa, aunque en una interfaz cualquiera se pueden enconcontrar menos que esta, ya que mediante una comunicación full-duplex se puede establecer la comunicación con tan solo 3 cables.

Las interfaces DTE y DCE se diferencian básicamente en la ocupación de su conexión enchufable: los PCs e impresoras suelen contener el sistema DTE, mientras que los routers e impresoras presentan la ocupación del terminar DTE. Un esquema de conexionado general viene reflejado tal y como aparece en la siguiente imagen:

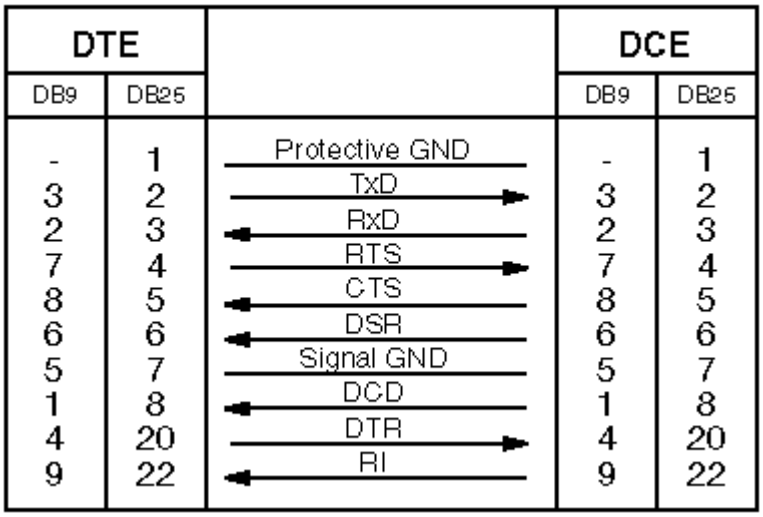


Figura 8: Conexionado RS-232

2.2 Conclusión

Tras comentar las principales interfaces de comunicación en serie, definir las más utlizadas dentro de los circuitos integrados: I2C y SPI; queda concluir que interfaz se adaptaría mejor a nuestro objetivo, y a los requisitos que nos impone el diseño de nuestro chip.

Si tuviesemos que realizar una comparación entre estos dos protocolos de comunicación, podrían distinguirse lo siguientes aspectos:

- **Rutado, Recuros y Topología.** I2C necesita únicamente dos líneas, mientras que el SPI formalmente define al menos 4 señales; si no se le añade ningún esclavo más al master. Sin embargo, existen variantes de esta interfaz que solamente requieren de 3 señales: convirtiendo la línea de transmisión en una línea bidireccional MISO/MOSI. Además, el SPI requiere trabajo y lógica adicional; y el único problema que presenta el I2C es la limitación de dirección del dispositivo de 7 bits. Por lo que, desde este punto de vista, el I2C es un claro ganador en el rutado de una placa y en la facilidad de construir una red.
- **Velocidad.** Si el dato debe ser transferido en alta velocidad, el SPI gana con creces a la otra interfaz. El SPI establece una comunicación full-duplex, ya que es una comunicación bidireccional capaz de estar transmitiendo y recibiendo mensajes simultáneamente. Aunque el SPI no define ningún limite de velocidad, las implementaciones físicas suelen ir alrededor de 10 Mb/s. El sistema I2C está limitado a 1Mb s en modo rápido y a 3.4 Mb/s en modo de alta velocidad, requiriendo este último, buffers de E/S específicos, que no siempre están disponibles fácilmente.
- **Elegancia.** A menudo se dice que I2C es mucho más elegante que SPI y que este es un protocolo muy "tosco". Realmente, ambos protocolos son igualmente elegantes y comparables en cuanto a robustez.
    - El I2C es elegante porque ofrece características muy avanzadas, como el manejo automático de conflictos de múltiples maestros y la gestión de dirección incorporada, en una infraestructura muy ligera. Sin embargo, puede ser muy complejo y sufre carencias de rendimiento.
    - El SPI, por otro lado, es muy fácil de entender e implementar y ofrece una gran flexibilidad para extensiones y variaciones. La simplicidad es donde reside la elegancia del SPI. Debe considerarse como una buena plataforma para construir pilas de protocolos personalizados para la comunicación entre los circuitos integrados. Entonces, de acuerdo con las necesidades del ingeniero, el uso de SPI puede requerir más trabajo, pero ofrece un mayor rendimiento de transferencia de datos y una libertad casi total.
    - Tanto SPI como I2C ofrecen un buen soporte para la comunicación con dispositivos de baja velocidad, pero SPI es más adecuado para aplicaciones en las que los dispositivos transfieren flujos de datos, mientras que I2C es mejor en aplicaciones de "acceso de registro" multimaestro.

Concluyendo la comparación, en el mundo de los protocolos de comunicación, ambas interfaces son a menudo consideradas como pequeños protocolos de comunicación si se comparan con Ethernet, USB y PCI-

Express. Sin embargo, no se debe olvidar el propósito de cada protocolo; Ethernet, USB y SATA están diseñados para "comunicaciones exteriores" e intercambios de datos entre sistemas completos; no para comunicaciones entre un circuito integrado y un conjunto de periféricos. En este ámbito, tiene sentido usar protocolos como I2C y SPI que se ajustan perfectamente a las necesidades que imponen este tipo de comunicación.

3 TÉCNICAS PARA LA REDUCCIÓN DE CONSUMO EN CIRCUITOS DIGITALES

Considerando el gran auge de los dispositivos alimentados a batería o, más recientemente, aquellos sistemas que carecen de esta mediante la aplicación de técnicas de caputara de energía del entorno (conocido como *energy harvesting*), la reducción del consumo de potencia es un requisitio fundamental. Por tanto, este capítulo se centra en la descripción de las principales técnicas para la reducción de la potencia dinámica y estática de los sistemas digitales modernos.

La reducción de la potencia dinámica, tradicionalmente, ha centrado el foco de las técnias de reducción del consumo, debido a que la disminución de la tensión de alimentación con el scalado tecnológico lo facilitaba. No obstante, en los sistemas digitales moderno, véase, por ejemplo, los microprocesadores, la contribución de la potencia estática se ubica entonor al 30 – 50 % [3] de la potencia total consumida. Este incremento se debe principalmente al aumento de las corrientes de fugas de los transisotres a medida que la longitud del canal se ha ido reduciendo. Por tanto, las técnicas actuales de reducción de consumo deben estar enfocadas en abordar ambas problemáticas.

Para mejorar la vida útil de la batería en portátiles y reducir los problemas de robustez inducidos por la temperatura, se ha investigado numerosas técnicas de ahorro de energía a nivel de circuito y arquitectura que se inclinan hacia una reducción de la potencia estática y/o dinámica. Por un lado, debido a la dependencia cuadrática de la potencia dinámica en la tension de alimentación, la reducción de la tensión de alimentación se ha convertido en una opción popular para la reducción dinámica de la potencia. Además de ello, otras técnicas importantes orientadas al diseño de bajo consumo son el tamaño de reducción de la capacidad de conmutación efectiva, el uso de una frecuencia de reloj variable o, incluso su desactivación (*clock gating*). También es una práctica común en los sistemas modernos, la desconexión de los raíles de alimentación de parte de los sistemas para minimizar su consumo (*power gating*). Por otro lado, las técnicas para minimizar la potencia estática de los circuitos lógicos y memorias incluyen el apilado de transistores, el uso de transistores CMOS con diferentes tensiones umbrales o la modificación de la tensión del sustrato.

Cabe destacar que estas técnicas consiguen proporcionar soluciones efectivas de ahorro de potencia, aunque muchas de ellas repercuten directamente en el rendimiento del sistema. Por tanto, es necesario combinarlas de forma adecuada y disponer de una lógica de control que permita su activación/desactivación, según requiera el sistema diseñado.

3.1 Tecnicas de reducción de la potencia dinámica

[3] Con el escalado de la tecnología, la potencia dinámica se reduce debido a la reducción de la tensión y alimentación y a la reducción de las capacidades parásitas (tanto de los transistores como las asociadas a la interconexión de los nodos). Sin embargo, al reducir la longitud de los transistores, así como sus capacidades parásitas, se logra incrementar su frecuencia máxima de operación. Además, para una misma área, el número de transistores que pueden integrase es mucho mayor, permitiendo el diseño de sistemas más complejos. Estos dos hechos causan un aumento significativo de la potencia dinámica y, por tanto, de la potencia media disipada.

Este último factor provoca un aumento de la temperatura del dado de silicio, generándose puntos localizados debido a la limitada capacidad de disipación que presentan tanto el sustrato como el encapsulado. Curiosamente, estos puntos también son una preocupación para el consumo de potencia estática, ya que las corrientes de fugas aumentan exponencialmente con la temperatura. Por tanto, es vital incorporar técnicas de reducción de potencia en los diseños nanométricos CMOS para reducir la disipación de potencia y evitar también problemas de robustez inducida por por la temperatura.

A continuación, discutiremos algunas de las principales técnicas de reducción de energía dinámica a nivel de circuito y arquitectura.

3.1.1 Clock Gating

[4] El reloj es el principal responsable del consumo de la potencia dinámica en los circuitos digitales. Este hecho se debe a que la línea de reloj presenta una gran carga capacitiva ya que está conectado a un gran número de elementos secuenciales, tales como registros o memorias. Además, para garantizar la integridad de señal del reloj facilita el rutado del reloj de un extremo a otro con el minimo “jitter” y “skew”, la red del reloj es tipicamente asociada con muchos buffers y elementos que reduzcan el efecto “skew”, que tambien agregran consumo a la potencia dinámica. Estos buffers son elementos que integrar el árbol de la línea de reloj, restructruran la señal de reloj y permiten que esta señal llegue con total integridad a todos los bloques.

[6] El árbol de reloj en circuitos digitales consiste en un generador o fuente de reloj y una serie de buffers de distribución que permiten entregar esta señal de reloj a todos los sistemas o bloques a los que están conectados. Por lo general, la distribución de la señal provoca latencias entre el punto donde se genera y el punto final.

Existen una serie de características que afectan a las limitaciones o incertidumbres de tiempo que se acaban de mencionar. El efecto estático “skew” hace referencia a la diferencia de tiempo que existe cuando una misma señal de reloj llega a distintos bloques en diferentes instantes de tiempo. Por otro lado, el efecto dinámico “jitter” explica la variación temporal de la señal de reloj. El principal efecto es la desviación de los pulsos de reloj, en otras palabras, que los flancos de subida y bajada no se producen en el instante deseado.

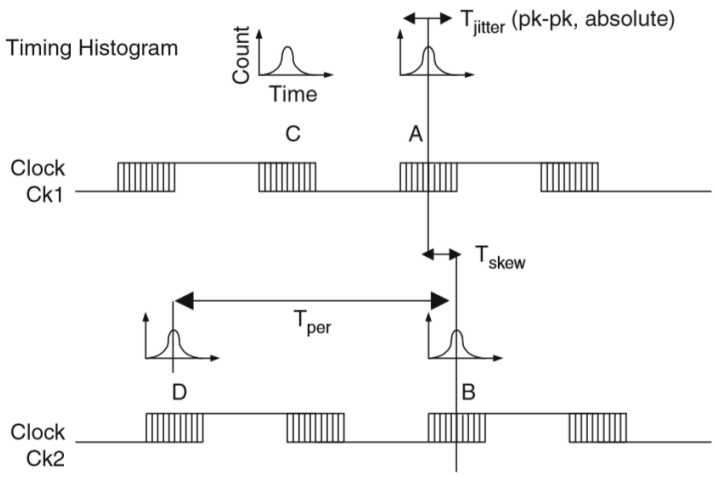


Figura 9: Histograma de tiempo de los efectos skew y jitter. [6]

Por último, otro de los principales problemas que presenta el árbol de reloj, es el alto consumo de potencia dinámica causado por la conmutación de los dispositivos activos, y por las altas frecuencias de reloj que se imponen en los circuitos digitales. Para subsanar esta disipación de energía, se realiza una parada selectiva del reloj en los bloques inactivos: *clock gating.*

Clock Gating es una técnica que reduce de manera efectiva el consumo de potencia dinámica desactivando el reloj de aquellos bloques secuenciales inactivos. Por tanto, al no existir frecuencia de reloj, la contribución de estos bloques a la potencia se anula.

Por lo general, la línea de reloj se activa mediante una puerta lógica AND que conecta la señal de habilitación del reloj y la propía línea del reloj. La desactivación del reloj, a parte de deshabilitar los bloques secuenciales donde el reloj accede, evita la carga y descarga de la carga capacitiva, así como la conmutación de los buffers del reloj en la red, ahorrando así energía dinámica.

3.1.1.1 Diseño Clock Gating

Una parte importante de la incorporación de esta técnica es determinar la lógica de control en el diseño, ya que es quién decide cuándo y cuánto tiempo se puede activar el reloj. En los diseños de sistemas convencionales, el reloj superior está conectado a cada uno de los bloques que necistan sincronización. Esto da lugar al consumo de energía en los bloques conectados a través de los siguientes métodos:

- Potencia consumida por la lógica combinacional que provoca el cambio de valores por cada flanco de reloj.
- Potencia consumida por flip-flops
- Potencia consumida por la red del buffer perteneciente al reloj.

El desafío de optimizar la potencia añadiendo Clock Gating es saber dónde y cuándo insertar la activación y desactivación del reloj. Normalmente, se introduce cuando las independencias entre bloques están bien

definidas, es decir, el diseño del sistema es modular. Aun así, es importante mencionar que su implementación require lógica adicional e interconexiones para generar las señales de habilitación del reloj.

Cualquiera lógica de Clock Gating tiene como objetivo generar una señal de reloj haciendola conmutar mediante una señal de control: permitiendo así propagarla solo cuando la funcionalidad del diseño lo requiera.

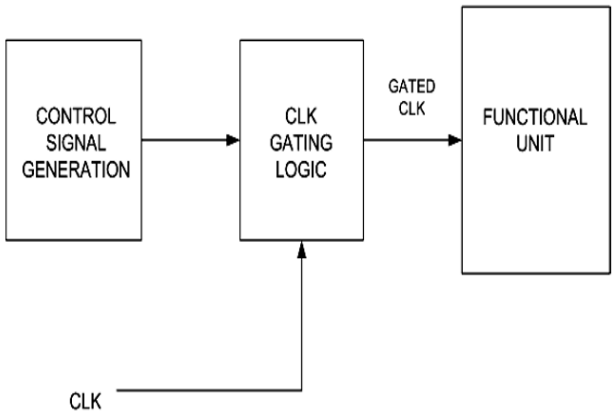


Figura 10: Esquema Funcional Clock Gating [4]

La lógica de Clock Gating apaga el reloj basándose en las señales de control. Cuando el reloj permanence inactivo el bloque funcional consigue ahorrar una gran cantidad de energía.

Para implementar todo lo anterior descrito, únicamente es necesario el uso de una puerta lógica AND que permite pasar la señal de reloj cuando lo habilita la señal "*enable*". Sin embargo, este diseño tiene una gran desventaja: provoca la aparición de glitches. La llegada tardía de la señal de habilitación puede dar lugar a espúreos no deseados en la señal de reloj. Incluso, en algunos casos, la salida puede terminar antes de lo esperado, lo que resta efectividad al sistema de Clock Gating. Por este motivo, aparece la celda mostrada en la Figura 11. La introducción de un biestable permite sincronizar la señal de activación con el flanco de reloj. Al ser el reloj quien marca los periodos donde puede cambiar el valor de salida del biestable y, por lo tanto, sincroniza la señal 'enable' con sus propios ciclos de reloj. Esta técnica ayuda a evitar la posibilidad de glitches en la señal de reloj hasta un cierto punto.

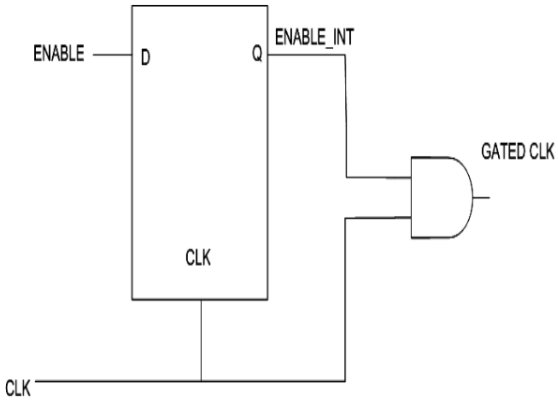


Figura 11: Clock Gating basado en Latch [4]

La inclusión de un biestable D almacena el último valor de salida en el biestable, durante el periodo inactivo, y permite cambiar las salidas de este durante el periodo activo. Así se consigue transferir la señal *enable* con total integridad, y anula cualquier tipo de glitch.

Otra técnica de reducción de glitches es la basada en Flip-Flop, tal y como se muestra en la Figura 12: Clock Gating basado en Flip-FlopsFigura 12. A diferencia de la técnica anterior, la salida de este biestable cambia con los flancos de reloj. Cuando el flanco de subida llega, el cambio de la señal *enable* será reflejada en la salida del FF. El periodo inactivo en esta técnica es mayor que en la anterior basada en Latch, por lo que hay una mayor oportunidad de perder el cambio producido en la señal *enable*.

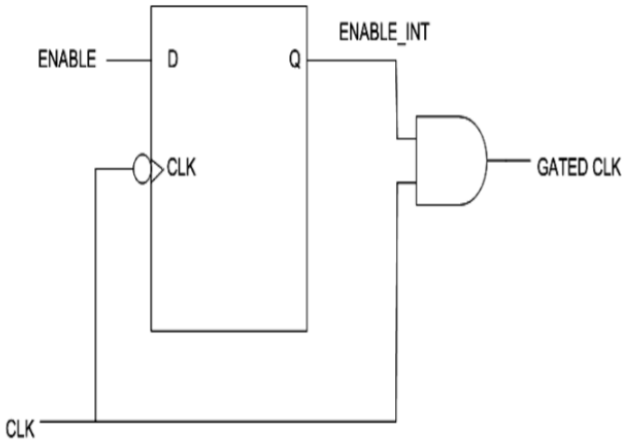


Figura 12: Clock Gating basado en Flip-Flops

3.1.2 Escalado dinámico de la frecuencia y tensión de alimentación (DVFS)

[7] El escalado dinámico de voltaje y frecuencia *(DFVS)* es una técnica común que varía la frecuencia y la tensión de alimentación en función de las cargas de trabajo instantáneas, permitiendo así la realización de todas

las tareas eliminando los periodos de inactividad. Esta técnica garantiza un ahorro de energía basado en la relación cuadrática y lineal del volataje y la frecuencia con la potencia dinámica, tal y como aparece en la siguiente ecuación:

$$P = CfV^2 + P_{static}$$

Ecuación 1: Potencia Dinámica

En esta ecuación, la capacidad del transistor viene determinada por *C*, la frecuencia de operación *f*, y la tensión de alimentación *V*.

Este sistema se basa en un algoritmo que analiza las cargas de trabajo individuales para concretar los niveles de tensión y frecuencia más optimos para minimizar la disipación de potencia dinámica. Realiza todas las tareas críticas a tiempo, y elimina los periodos ociosos.

Las tareas con alta exigencia de rendimiento computacional necesitan que sean ejecutadas a altas frecuencias. Sin embargo, para tareas de baja velocidad se requieren niveles de frecuencia más bajos ya que terminarlas antes de la fecha límite no genera beneficios. Aunque la reducción de la frecuencia de operación *f* disminuye la disipación de potencia dinámica, tal y como aparece en la Ecuación 1, esto no afecta a la energía total consumida, ya que el consumo de energía es independiente, tal y como muesta la Ecuación 2. Por lo que reducir la frecuencia por sí sola no mejora la vida útil de la batería. De manera similar, reducir solo el voltaje de un sistema, mientras se mantiene una frecuencia constante, mejora solo la eficiencia energética.

$$E = NCV^2$$

Ecuación 2: Energía

Por tanto, con el fin de minimizar la disipación de energía, los algoritmos DVFS determinan las frecuencias de operación óptimas del sistema acordes a las tensiones de alimentación. Al igual que se requieren diferentes niveles de voltaje para la implementación de DVFS, en función de la temperatura, el hardware implementado y las variaciones de procesos.

3.2 Optimización de Circuitos para bajo consumo

Las técnicas de diseño a nivel de circuito para la reducción estática de potencia suelen incluir el dimensionamiento para limitar el retraso de las puertas lógicas, con el fin de reducir la capacitancia de conmutación, y el uso de transistores con múltiples tensiones umbrales.

- **Dimensionamiento de tamaño**. Estas técnicas explotan principalmente la holgura de sincronización; disponible en los trayectos más cortos y las hacen más lentas: igualando así las trayectorias de sincronización.
- **Múltiples tensiones de alimentación**. Esta técnica se plantea como un problema de optimización donde seleccionamos la potencia como objetivo, y el retardo de trayecto como restricción.

3.2.1 Tecnicas para mitigar las corrientes de fugas en las puertas lógicas

El aumento de la potencia de fuga provoca serios problemas en diseño y testeo de circuitos. La suma total de todos los componentes de fuga constituye una fuente importante de disipación de energía en circuitos microelectrónicos. Algunas técnicas que regulan la fuga de potencia en los circuitos lógicos se muestran a continuación:

- **Control de Entrada**. Dado que la corriente de fuga total de un circuito depende de sus entradas primarias, la aplicación de los mejores vectores de entrada a algunos circuitos puede hacer que la corriente de fuga disminuya significativamente.

  Por un lado, hay que considerar la conducción de los transistores para el cálculo de la corriente de fuga. Si escogemos un caso donde se apilan tres tansistores encendidos, el sistema actuará como un circuito pequeño; y la corriente de fuga será la suma de los 3 PMOS. Al controlar la conmutación de los transistores, se pueden gestionar las tensiones Vs, Vgs y Vds, con el fin de reducir la corriente de fuga. Una tensión positiva Vs provoca una tensión negativa en Vgs, reduciendo asi la corriente de fuga. Además, una tensión positiva de Vs indica la existencia de "body effect" y una reducción en Vds. Ambos efectos aumentan el umbral de voltaje que permiten una reducción exponencial en el subumbral de fuga. Este efecto, llamado apilamiento, es muy eficaz para la reducción de corriente de fuga. Debido a la complejidad exponencial con respecto al número de entradas, se han desarrollado algoritmos eficientes para determinar la solución casi óptima basada en la búsqueda aleatoria o algoritmos.

  Por otro lado, el efecto de apilamiento no es considerablemente efectivo en la tecnología CMOS en nanoescala. Por esta razon, es necesario considerar el impacto del efecto de apilamiento en la fuga de estos componentes. En estos casos, la corriente de tunel de puerta puede verse incrementada con el aumento de la corriente del subumbral.

  Debido a esto, un vector de entradas '00' reduce la fuga de potencia y establece la minima corriente de subumbral; mientras que la entrada '10' administra la minima corriente de compuerta al sistema. Ante esta ambigüedad, se desarrollan algoritmos géneticos y redes neuronales que pongan solución a estos problemas de optimización.

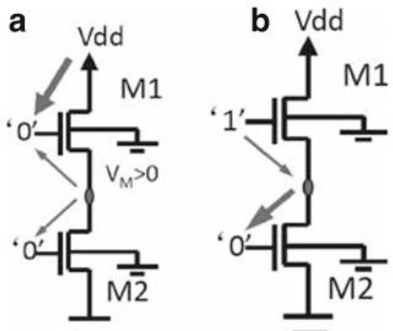


Figura 13: Control de entradas [3]

- **Diseño Dual de Tensión de Umbral**. En un circuito lógico, se pueden asignar un gran umbral de voltaje a algunos transistores en los caminos no criticos con el fin de reducir la corriente de fuga, a la vez que se mantiene bajos umbrales en los caminos críticos donde se requiere una alta velocidad de conmutación, y mejor así también, la eficiencia en la fuga de potencia. Por lo tanto, no es necesario transistores de control de fuga adicionales para conseguir grandes prestaciones y baja disipación de potencia de manera simultánea.

  En la tecnología CMOS, una tensión de umbral dual (Dual-Vth) que nos permite tener grandes valores de umbral en caminos donde no se requieren grandes prestaciones en velocidad de conmutación y un umbral pequeño en los caminos críticos de conmutación, tiene el mismo retraso que una tensión baja de umbral (low-Vth) para todos los recorridos del circuito. La posibilidad de imponer una gran tensión de umbral en los caminos no críticos consigue reudcir considerablemente la corriente de fuga. Por esta razón, esta técnica consiguer reducir considerablemente el ahorro de energía por fuga en ambos modos: en reposo y activo; sin recurrir retrasos.

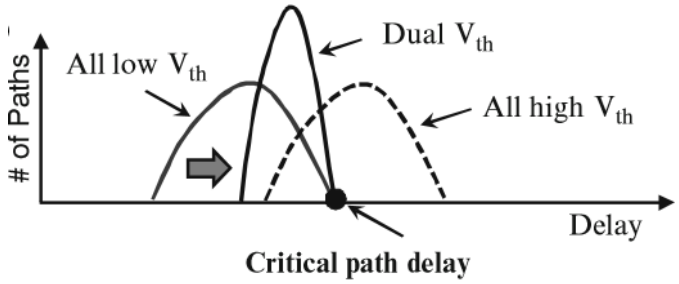


Figura 14: Gráfica de comparación Vth en función del número de recorridos y el retraso. [3]

Como aparece en la figura anterior, se diseña una tensión de umbral con un retraso medio comprendido en la zona de superposición de los umbrales alto y bajo. Además, se configura un recorrido mayor dentro del circutio a la tensión Dual-Vth, es decir, la capacidad de llegar a todos los caminos críticos y no críticos a los que se llegaba independientemente con los otros dos umbrales.

3.3 Power Gating

Ante lo anteriormente descrito, se presenta una nueva técnica de reducción de consumo que pretende reducir las fugas de potencia y la potencia dinámica: Power-Gating. [5] Como ya se ha comentado, existen multitud de aplicaciones de bajo consumo donde el consumo en reposo supera las perdidas dinámicas de potencia. El incremento de las corrientes de fuga debido al escalado de la tecnología y a la complejidad de los sistemas son las responsables de esta tendencia.

Por otro lado, la mayoría de circuitos integrados, diseñados con gran cantidad de submodulos, no utilizan todas sus funciones al mismo tiempo, por lo que muchos de estos bloques permanecen inactivos, y dependiendo del usuario incluso puede haber partes del circuito que nunca son usadas durante la vida del producto. Por esta razón, la técnica Clock Gating puede ser usada para apagar grandes partes del chip durante la mayoría del tiempo. Esta técnica resolvía el problema del consumo hasta que el escalado de los circuitos impusieron grandes corrientes de fuga en los bloques inactivos, anulando todo el ahorro conseguido con el Clock-Gating.

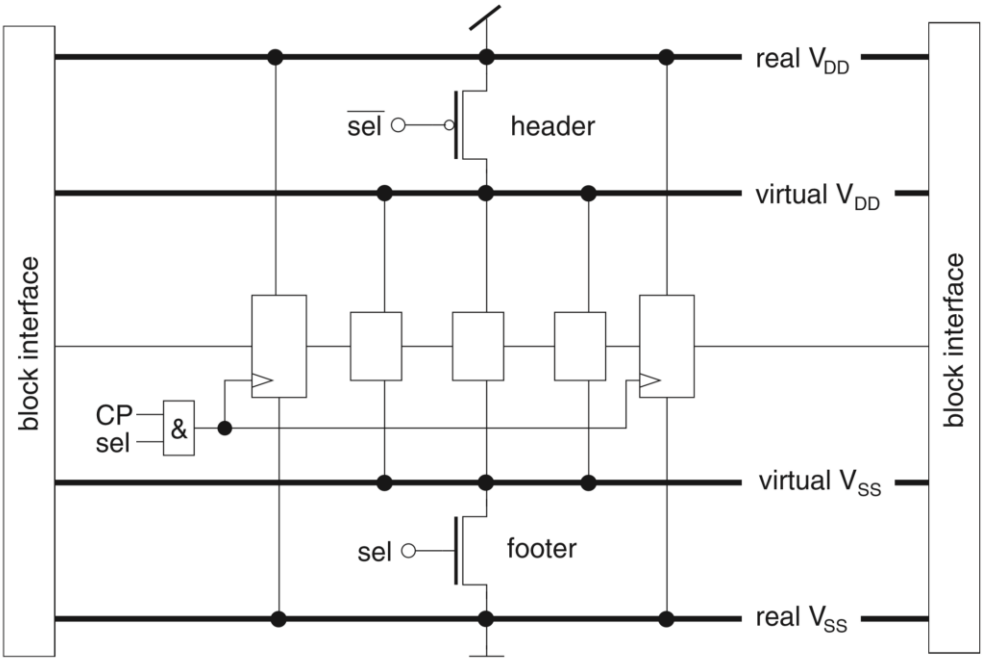


Figura 15: Esquema conexionado Power Gating

Para solucionar este problema, se propone una combinación de ambas técnicas. Como se muestra en la anterior figura, el bloque del circuito en reposo no está separado solamente por el árbol del reloj sino también de la alimentación. Por lo tanto, un "sleep" transistor es insertado entre el bloque del circuito y los voltajes VDD o VSS. Este transistor puede colocarse entre cualquiera de los dos voltajes e incluso en los dos. La red de alimentación local conectada a la vía de alimentación del transistor se le denomina: alimentación virtual.

Desconectar el bloque del circuito reduce la mayoría de la corriente fuga, sin embargo, la corriente residual es principalmente determinada por las propiedades y dimensiones del interruptor. Por esto, el transistor que realiza la desconexión con la alimentación virtual es preferiblemente un dispositivo con un alto umbral de tensión y una delgada puerta de óxido. De esta manera, el consumo en reposo puede ser reducido en dos o tres órdenes de magnitud. La corriente en el dispositivo "sleep" es reducida inmediatamente tras el corte de alimentación. Así las corrientes de fuga en el circuito lógico siguen su flujo y descargan las capacitancias internas.

El bloque del circuito puede ser activado encendiendo el transistor "sleep" de nuevo. Todas las capacidades son recargadas y una gran corriente tiene lugar durante esta fase, por lo que para evitar problemas con los módulos de alrededor el bloque se se enciende de manera controlada. En modo activo, el dispositivo de corte no tiene influencia sobre el consumo del bloque. Por lo que esta técnica es únicamente beneficiosa si existen

periodos considerables donde el circuito no es requerido.

En definitiva, power gating consiste en la desconexión de un circuito digital de su tensión de alimentación cuando este no se requiere mediante la inserción de un interruptor. De esta forma, se consigue un doble efecto en la minimización de la potencia consumida. Por un lado, al apagar esa parte del circuito sin uso, la potencia dinámica se elimina. Por otro lado, la potencia estática también se minimiza ya que las corrientes de fugas se reducen varios ordenes de magnitud.

3.3.1 Propiedades básicas de Power Gating

La implementación de esta técnica puede parecer una tarea trivial, sin embargo, los requerimientos para una económica y efectiva implementación deben seguir unas características propias que se definene a continuación:

- Alto radio de reducción de fuga. Por ejemplo, una muy alta conexión del circuito a la alimentación en el modo inactivo.
- Pequeño impacto en la conmutación al modo activo.
- Realización de un power gating modular que permita la desconexión de los módulos de manera independiente.
- Rápida conmutación entre los diferentes modos con una pequeña energía.

4 DESCRIPCIÓN DE LA ARQUITECTURA PROPUESTA

4.1 Definición del sistema

Nuestra interfaz de comunicación viene contendida en un módulo global que comprende los dos sitemas (SPI y QSPI) conjuntamente en un mismo módulo. El sistema contará como líneas principales la descritas en el capítulo 2, un reloj de sistema impuesto por el master (QSCK), cuatro líneas de transmisión bidireccionales, y una señal de habilitación y elección del slave que, en nuestro caso, se denominará CSN (Chip Selection); tal y como define el estándar.

En nuestro caso, se ha desarrollado el dispositivo slave que será el que se integre en el circuito integrado, y se ha emulado el dispositivo maestro, que más tarde explicaremos en el capítulo de verificación. Por tanto, es necesario de un dispositivo externo, por ejemplo, un microcontrolador, que inicie la misma. A su vez, el formato de comunicación, definido por la polaridad (CPOL) y la fase del reloj (CPHA), está impuesto por diseño con la idea de simplificar el funcionamiento del módulo.

En nuestro caso, usaremos la configuración denominada "modo 0" de SPI con los siguientes valores de configuración:

- CPOL = 0
- CPHA = 0

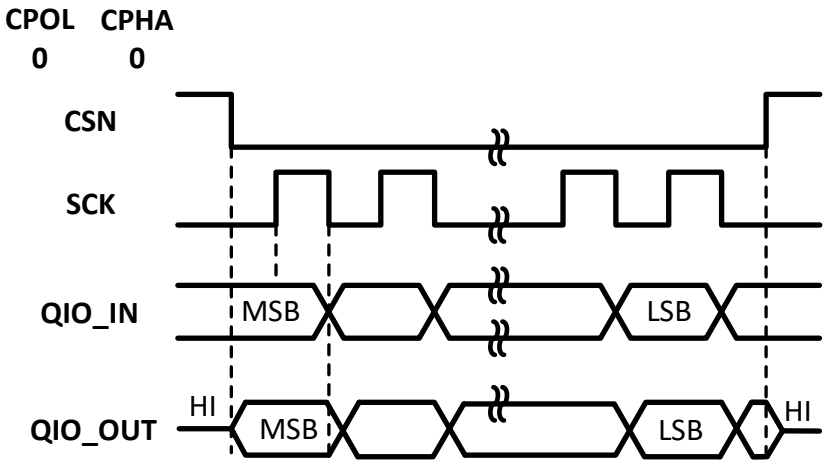


Figura 16**.** Cronograma Fase/Polaridad QSPI

La comunicación comienza con la transición de nivel alto a bajo de la entrada CSN. Asimismo, el reloj QSCK debe permanecer siempre a 0 al comenzar la comunicación. Además, es necesario tener en cuenta las siguientes consideraciones:

- Los datos en QIO_IN siempre serán estables en primer flanco de subida de QSCK (serán muestreados por el chip en los flancos de subida)
- Los datos en QIO_OUT serán puestos por el chip en el flanco de bajada de QSCK, de forma que sean muestreados por el SPI MASTER en el flanco de subida.
- El inicio de la comunicación comienza tras la baja de la señal CSN. La subida de CSN se realizará una vez la comunicación ha sido completada, es decir, se haya completado la escritura o lectura del registro. Se contarán flancos de subida de QSCK y, si tras los ciclos definidos no se ha completado correctamente la escritura o lectura, esta desechará la trama y esperará la siguiente bajada de CSN.

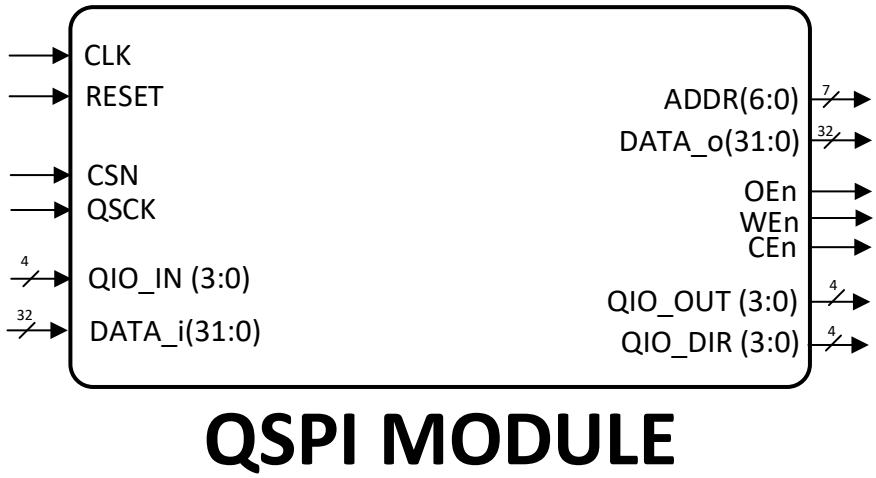


Figura 17: Bloque Módulo QSPI

El sistema definido presenta los puertos de entrada y salida mostrados en la Figura 17. A continuación, se detallan el significado de cada uno de ellos:

- **QSCK.** Reloj de sincronización impuesto por el dispositivo maestro. Esta señal se encargar de sincronizar la transmisión y recepción de los datos. Dependiendo de la capa física activa, SPI o QSPI, la sincronización de los datos se realizará de una forma diferente.
- **RESET.** Reset síncrono del módulo que se activa a nivel bajo.
- **CSN.** Señal de habilitación y de comienzo de la comunicación que será activada por el dispositivo maestro. Cuando esta señal permanece a nivel alto, los canales de comunicación permanecen en alta impedancia para permitir que otros dispostivios coenctados al bus puedan comunicarse. Una vez se activa, CSN = 0, la comunicación entre el maestro y el esclavo se llevará a cabo. La comunicación siempre debe comenzar con la línea QSCK a nivel bajo.
- **QIO_IN**. Bus de entrada del módulo QSPI que permite recibir datos del maestro de la comunicación.
- **QIO_OUT.** Bus de salida del módulo QSPI que permite enviar datos al maestro de la comunicación.
- **QIO_DIR.** Marca la dirección de entrada y/o salida de cada puerto de la interfaz de comunicación. En concreto, esta señal se utilizará para controlar cuando el PAD del circuito integrado actúa como entrada o como salida.

- **ADDR.** Dirección del registro donde se va a realizar la operación de escritura o lectura.
- **DATA_i.** Datos de entrada al módulo QSPI desde el sistema, ya proceda de un registro o de la memoria.
- **DATA_O.** Datos de salida a transmitir al sistema, ya sea a un registro o a la memoria.
- **OEn.** Señal que activa la salida de memoria. Es activa a nivel bajo.
- **WEn.** Señal que indica si la operación a realziar sobre la memoria se corresponde con una escritura (WEn = 0) o lectura (WEn = 1).
- **CEn.** Señal que habilita el uso del reloj de la memoria. Esta señal es activa a nivel bajo.

4.2 Descripción de la trama

La trama de comunicación correspondiente a la interfaz QSPI constará de 40 bits. Cada comunicación se compondrá de la trama mostrada en la Tabla 1. Como se puede observar, está compuesta de un byte de instrucción, que indica el tipo de operación a realizar y sobre qué registro llevarla a cabo, y de 32 bits de datos. Por tanto, esto supone una transmisión total de 40 bits.

CMD			**Dato**		
0		7	8		32

Tabla 1: Representación del formato de trama usado para la interfaz QSPI desarrollada.

El formato del byte de instrucción se recoge en la Tabla 2 y este está compuesto de los siguientes campos que se listan a continuación:

- **Bit de Operación:** OC (Operating Code).
  - OC = ‘0’ Operación de escritura.
  - OC = ‘1’ Operación de lectura.
- **Dirección:** ADRR (Address). Este campo de 7 bits indica la dirección del registro sobre la que se va a realizar la operación.

CMD							
MSB							**LSB**
OC Bit de Operación	ADDR[6]	ADDR[5]	ADDR[4]	ADDR[3]	ADDR[2]	ADDR[1]	ADDR[0]
	Dirección						

Tabla 2: Descripción de los bits de instrucción

4.3 Operaciones de transmisión

La operación de transmisión estará consituida por las siguientes reglas:

- Cada comunicación constará de un bit de operación más la dirección del registro que se desea leer o

escribir codificada en 7 bits y 32 bits de datos que corresponden al contenido del registro interno o de memoria. Al conjunto compuesto por el bit de operación (en adelante OC – operation Code) y la dirección del registro, se denomina byte de instrucción. Por último, Esto supone una transmisión total de 40 bits.

- La transmisión del byte de instrucción se realiza por la línea QIO_IN [0]. Mientras tanto, el resto de líneas, QIO_IN[1], QIO_IN[2] y QIO_IN[3], permanecen inactivas.

En las instrucciones de escritura, tras enviar el byte de instrucción por la lína QIO_IN [0], se debe enviar el contenido a escribir por las cuatro líneas de transmisión QIO_IN.

Si la operación es de lectura, será necesario un periodo de cambio de dirección en las líneas. Este periodo se define como "dummy". Si la operación es de escritura, no es necesario este cambio de dirección, ya que las líneas de transmisión están correctamente configuradas.

En las instrucciones de lectura, tras enviar el byte de instrucción y esperar el periodo dummy, a través de las cuatro líneas de transimisión QIO_OUT se devolverán los datos correspodientes.

4.4 Arquitectura del sistema

El módulo QSPI se compone de los siguientes bloques:

• **QSPI Control**: bloque que habilita la comunicación tras la bajada del CSn. Es el encargado de habilitar y deshabilitar el clock gating del resto de los bloques del módulo QSPI.

• **QSPI Transfer**: contiene la capa física del sistema, y se encarga de almacenar y transmitir los datos de la comunicación.

• **QSPI Interpreter**: se encarga de procesar las instrucciones, indicando si la operación es de lectura o escritura y sobre qué registro recae dicha operación.

• **Registro de Estado**: contiene un registro de los errores que han ocurrido en el sistema. Es un registro accesible sólo en modo lectura desde la interfaz QSPI. Se restablece tras efectuar una operación de lectura sobre dicho registro.

La distribución de los bloques anteriormente expuestos, se muestran gráficamente en la Figura 18. A continuación, en los próximos apartados se describen cada uno de los bloques de forma individual.

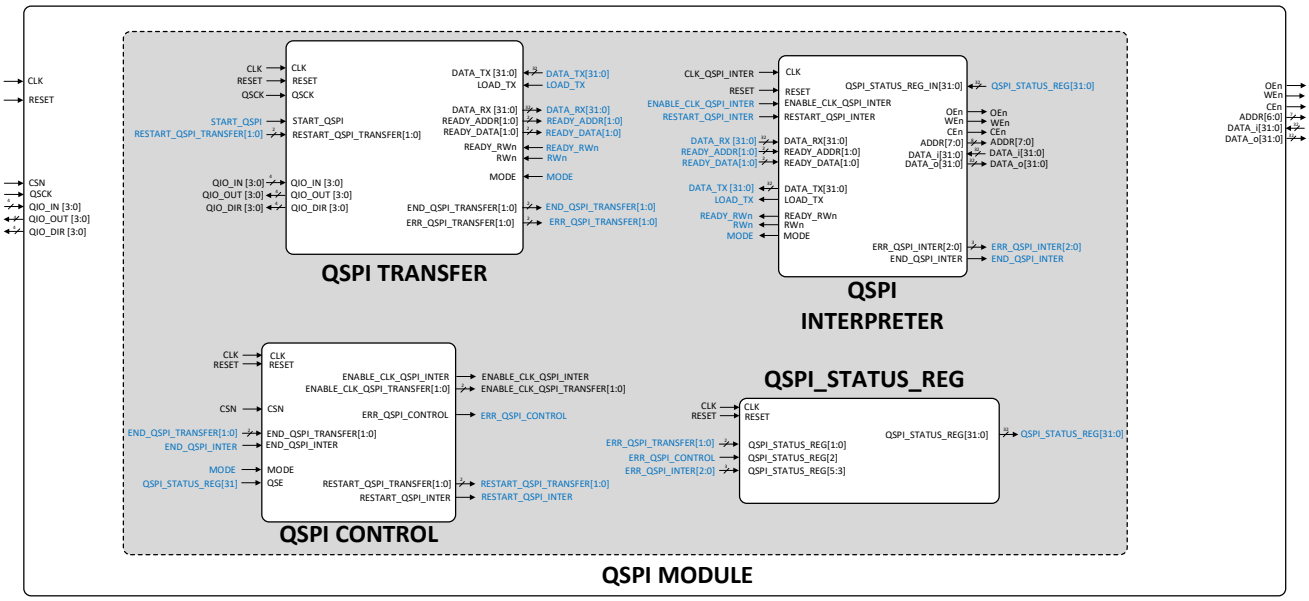


Figura 18: Arquitectura QSPI Módulo

4.4.1 Bloque QSPI Control

El bloque *QSPI Control* es el encargado de iniciar y finalizar la comunicación en el esclavo; habilitar y deshabilitar los relojes de los dos subsistemas que forman nuestro módulo SPI: *Transfer* e *Interpreter*. Además, este bloque permanece como encargado de los errores en los otros dos subsistemas, interrumpiendo la comunicación si se produce algún error en alguno de ellos.

La Figura 19 muestra el esquema representa el bloque *QSPI Control*, especificiando sus entradas y salidas, así como el tamaño de los buses de control y datos utilizados.

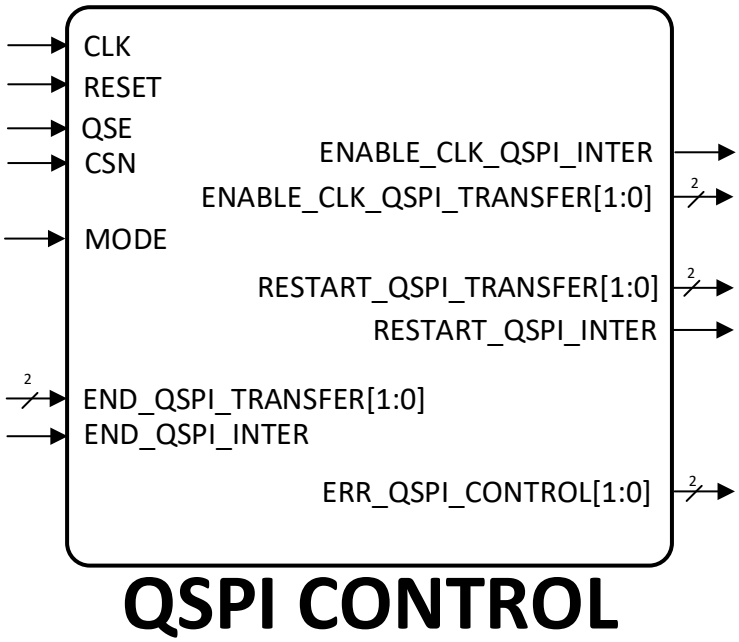


Figura 19: Bloque QSPI Control

La máquina de estados realiza los siguientes pasos:

1. En el estado inicial, llamado IDLE, se esperaa que la señal Chip Select CSN pase a nivel bajo para iniciar la comunicación. Según el estándar SPI en modo 0, la comunicación debe comenzar con *SCK=0* cuando se active la señal de Chip Select, en caso contrario se produce un error considerado en el sistema como error de comunicación. Mientras la señal permanezca a '1', se permanece en este estado hasta que se active a nivel bajo. Además, se transfiere el modo de transmisión del módulo.

2. Una vez iniciada la transmisión correctamente, se avanza al siguiente estado ENABLE CLK, donde se habilita el *enable* de los relojes de ambos subsistemas, y se mantiene a la espera de que cualquiera de los otros dos módulos indique que ha terminado el proceso. En el caso del reloj del bloque *QSPI Transfer*, se habilita uno de los dos bits del *enable* dependiendo del modo transmisión que este configurado.

3. Si por un caso, el bloque SPI Interpreter pusiese su salida *end_spi_interpreter* a 1 antes que el bloque SPI Transfer, se pasaría al estado DISABLE CLK SPI INTER. Si, por otro lado, se activa antes la salida que marca el fin del bloque SPI Transfer, se pasa al estado DISABLE CLK SPI TRANSFER. Para los

dos, se deshabilita el paso del reloj del bloque en el estado donde se encuentran. En cualquiera de estos dos estados, se espera a la activación de la señal de finalización del otro bloque, es decir, si estamos en DISABLE CLK SPI INTER, nos mantendremos en este estado hasta que el bloque SPI Transfer haya finalizado, y si estamos en el estado DISABLE CLK SPI TRANSFER, esperaremos a la finalización del proceso del SPI Interpreter.

4. En ambos casos, se avanza hasta el estado WAIT CSN donde se deshabilitan ambos relojes. Si en alguno de estos estados, se activa a nivel alto la señal de CSN o nos llega la señal de error (*QSE*), se detendrá la comunicación y se avanzará hasta el estado ERR. Tanto en el estado WAIT CSN como ERR, se espera a la subida de la señal CSN, en el primer caso para finalizar la comunicación correctamente, y en el segundo caso para reiniciar la comunicación tras un fallo.
5. El estado que sigue, RESTART**,** es un estado de transición para alcanzar el punto de reposo, IDLE.

Este bloque tiene como salida un vector de errores de dos bits que indica si el error viene producido por otro bloque, o si viene provocado por un fallo en la polaridad.

El diagrama de estados de la FSM se muestra en la Figura 20.

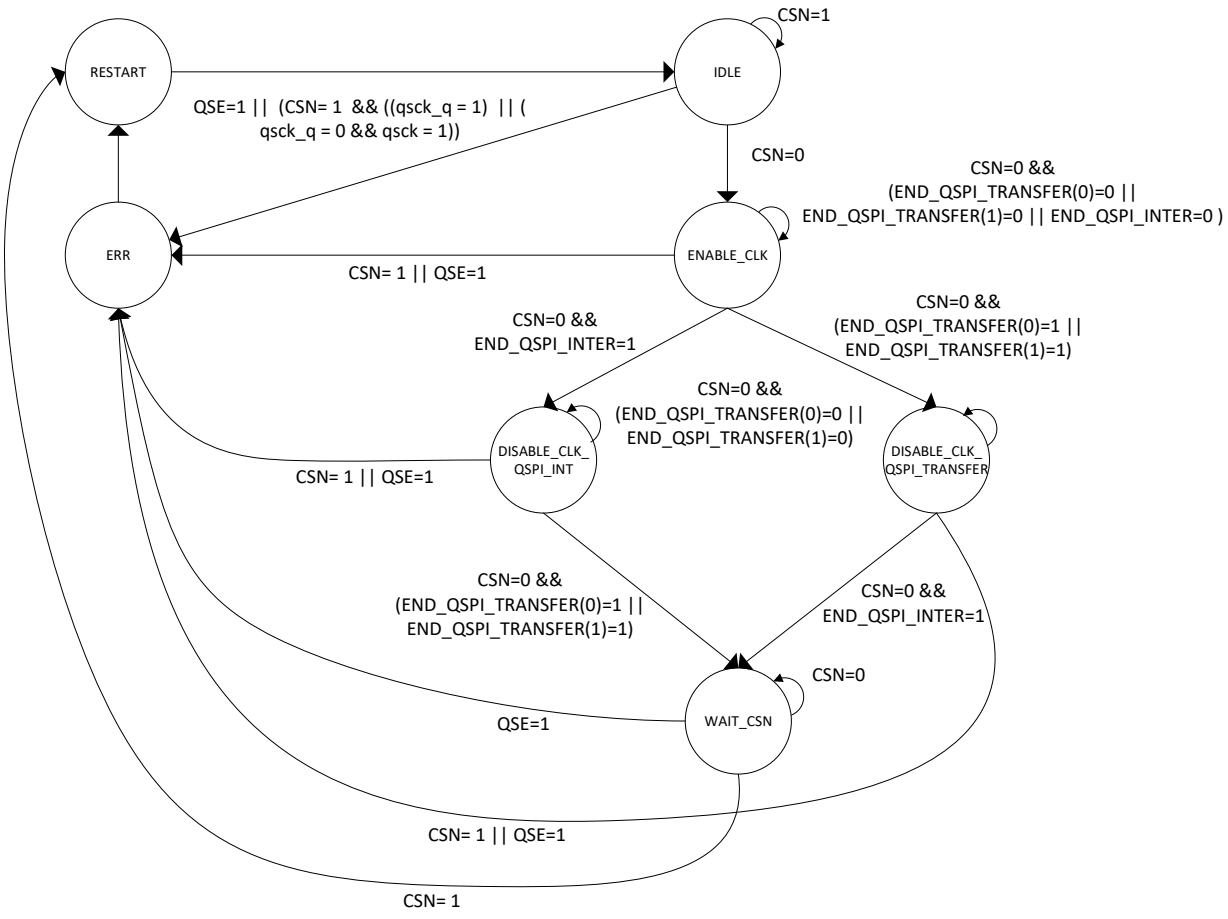


Figura 20: Máquina de control QSPI Control

4.4.2 Bloque QSPI Transfer

La función de este bloque es transmitir los bits en paralelo por las líneas de transmisión desde los registros de desplazamiento, y generar las señales *ready_addr*, la cual indica que ya se ha recibido la dirección, y *ready_data*, que alerta de que el dato ya está listo.

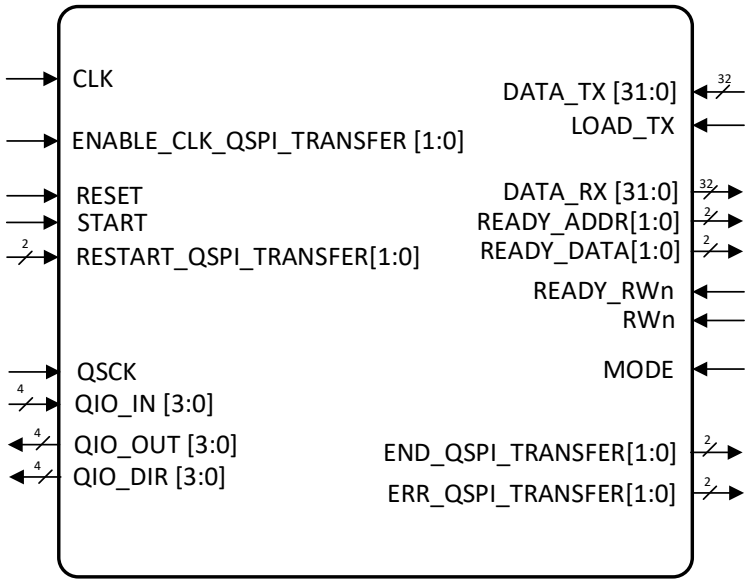


Figura 21: Bloque QSPI Transfer

Este bloque contará con los puertos de entrada y salida mostrados en la Figura 21. A su vez, la Figura 22 representa un diagrama de bloques de los diferentes elementos que componen este módulo.

Este bloque se caracteriza por contener la capa física del sistema QSPI, en su vertiente SPI, *SPY Physical Layer*, y QSPI, *QSPI Physical Layer*. Además de contener dos registros: uno donde se va almacenando los bits recibidos en serie, *reg_rx*, y otro desde donde se van serializando los bits para su transmisión, *reg_tx*.

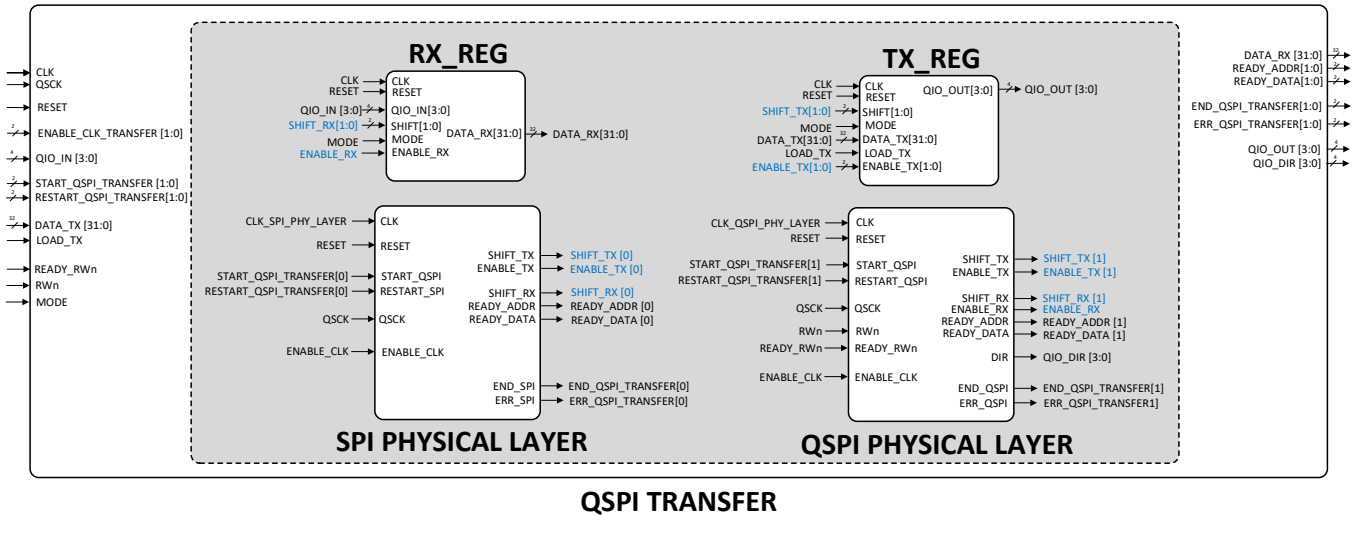


Figura 22: Arquitectura QSPI Transfer

4.4.2.1 Bloque SPI Physical Layer

El bloque *SPY Physical Layer* es el bloque encargado de suministrar las señales de control que permite el desplazamiento de los dos registros comentados anteriormente cuando el transceptor desarrollado trabaja en modo SPI.

La Figura 23 muestra el bloque del módulo *SPI Physical Layer*.

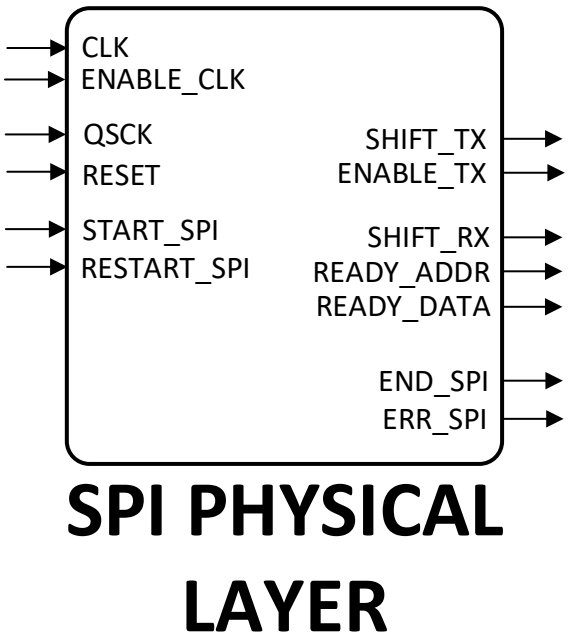


Figura 23: Bloque SPI Physical Layer

La máquina de estados que gobierna el bloque *SPI Physical Layer* comienza en el estado de reposo a la espera de que la señal start indique el comienzo de la comunicación.

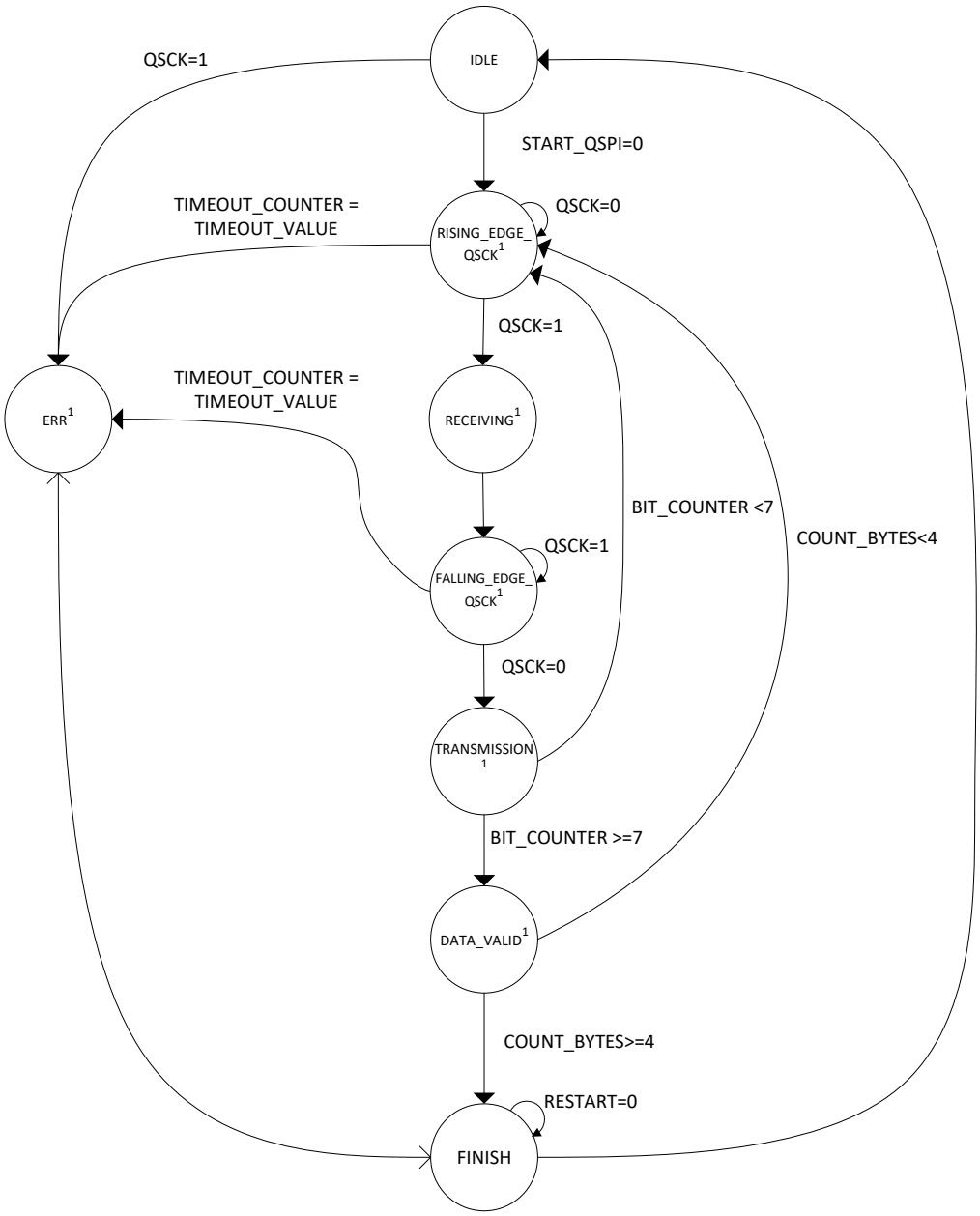
1. El proceso secuencial continua hasta el siguiente estado RISING EDGE QSCK, donde espera al flanco de subida del reloj de la interfaz para pasar al siguiente estado: RECEIVING.
2. En este último estado, se llevará a cabo la recepción de bits en serie.
3. El siguiente estado FALLING EDGE QSCK, espera al flanco de bajada del reloj de la interfaz del SPI.
4. El siguiente estado TRANSMISSION habilita el traspaso de bits en serie por el canal de salida de la interfaz. Este estado además cuenta con un contador del número de bits hasta completar el byte (8 bits).
5. Si se cumple la condición anterior pasará al siguiente estado DATA VALID para comprobar si se han completado los 4 bytes de la comunicación, y pasar al último estado FINISH, o por lo contrario volver al estado RISING EDGE QSCK.
6. El último estado, FINISH, deshabilita la comunicación en el PAD de salida, e indica la finalización de la secuencia.

Los estados FINISH, RISING EDGE QSCK y FALLING EDGE QSCK cuentan además con un

temporizador que marca el tiempo de ejecución del sistema, si este sobrepasa un valor predefinido, denominado *TimeOut*, el proceso evolucionará hasta el estado de ERR. Una vez alcanzado este estado, la única forma de que la máquina de estados evolucione es aplicando la señal restart. El bloque contiene un pin de salida que se activará si se ha producido un error de *TimeOut*.

En todos los estados de la secuencia, la activación a nivel alto de la señal *restart* reiniciará el sistema, evolucionando éste hasta el estado de reposo, IDLE.

El bloque *SPI Physical Layer* se ha implementado como una máquina de estados regida por el diagrama mostrado en la Figura 24.



[1] La siguiente transición no está representada para clarificar el diagrama

```
if restart_spi = 1 then
        n_state <= IDLE;
end if
```

Figura 24: Máquina de estados SPI Physical Layer

4.4.2.2 Bloque QSPI Physical Layer

El bloque *QSPY Physical Layer* es el bloque encargado de suministrar las señales de control que permiten el desplazamiento de los dos registros comentados anteriormente para su transmisión en paralelo dependiendo de si la operación es de lectura o escritura. Además, es el encargado de gestionar la señal de dirección que configura los pads conectados a los puertos del transceptor.

La Figura 25 muestra el bloque del módulo *QSPI Physical Layer*.

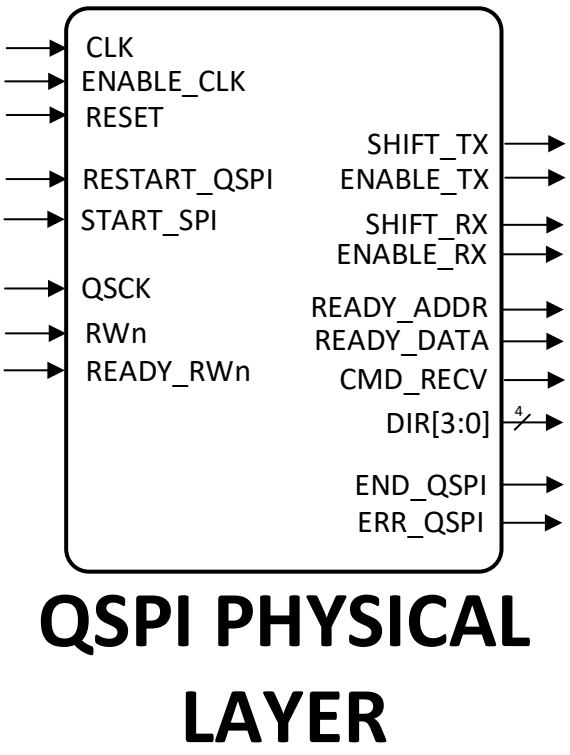


Figura 25: Bloque QSPI Physical Layer

El bloque QSPI Physical Layer se ha implementado como una máquina de estados que se representa en la Figura 26. La máquina de estados realiza los siguientes pasos:

1. El proceso del QSPI Physical Layer comienza en el estado de reposo a la espera de que la señal start indique el comienzo de la comunicación.
2. El proceso continúa en el estado RISING EDGE QSCK CMD, donde espera al flanco de subida del reloj de la interfaz para pasar al siguiente estado: RECEIVING CMD.
3. En RECEIVING CMD, se habilitan las señales de transmsión para la recepción de bits en serie y se comprueba si el número de bits recibido hasta entonces es menor que 7 para dar por finalizado la recepción de los bits de instrucción. Si es mayor que 7, el CMD está listo, y el sistema avanza hasta READY CMD. Si no, el sistema espera al flanco de bajada, y vuelve al estado RISING EDGE QSCK CMD.
4. En el estado READY CMD, se activa la señal que indica que están listos los bits de instrucción, y

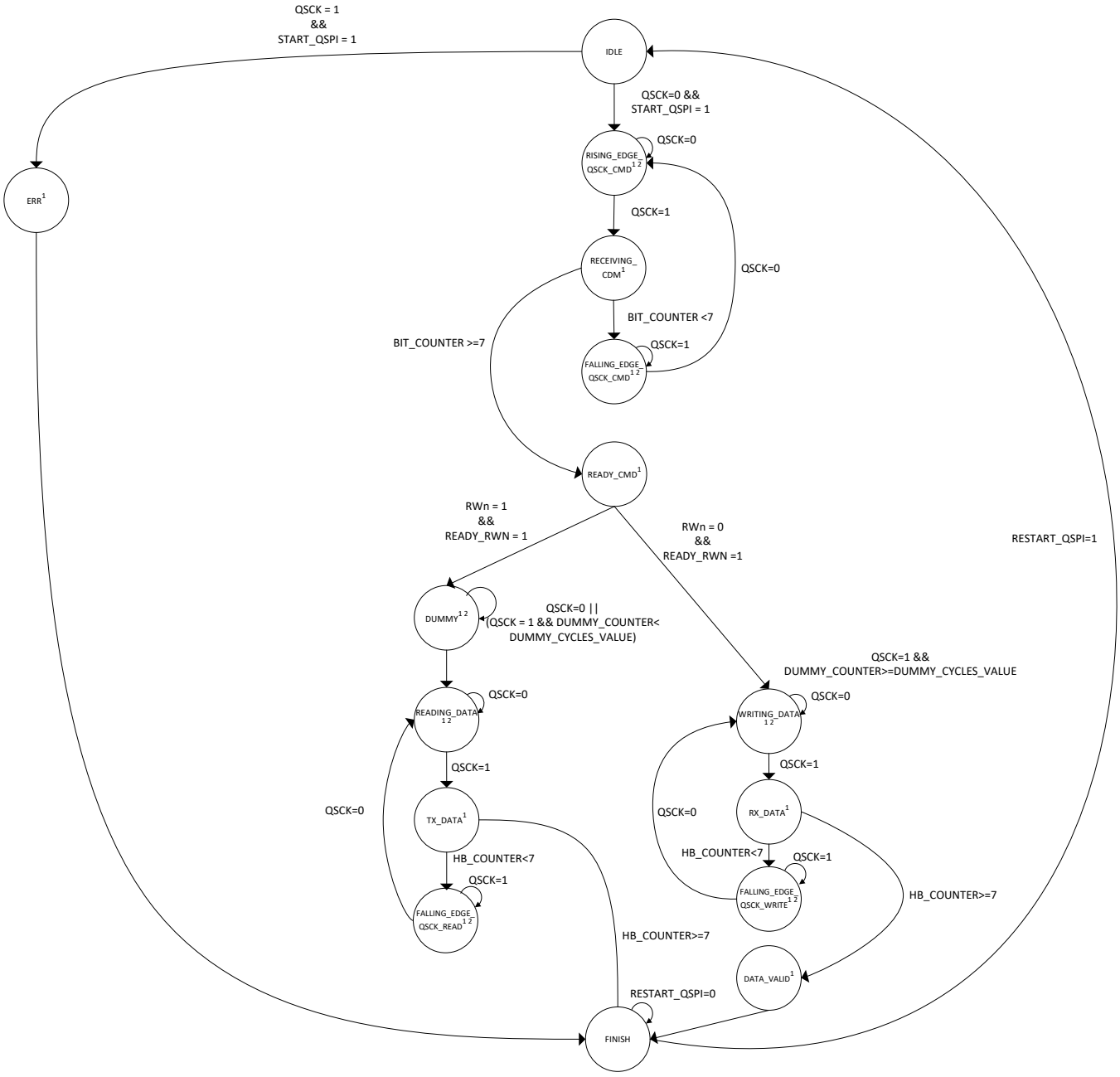
según la operación sea de lectura o escritura será necesario el periodo dummy. Si la operación es de escritura, no será necesario cambiar las líneas de dirección por lo que el sistema avanzará hasta WRITING DATA. En cambio, si la operación es de lectura, habrá que realizar un cambio de dirección, y el siguiente estado sería DUMMY FEDGE.

- Si el camino elegido es WRITING DATA, desde ese estado avanzará hasta RX DATA, donde se activarán las señales de transmisión de escritura y se comprobará si se ha completado la palabra de 8 bits. Si no es así el sistema avanzará hasta FALLING EDGE QSCK WRITE para esperar el flanco de bajada, y repetir el proceso. Si el sistema llega al estado DATA VALID, se comprueba si se han transmitido los 32 bits, y avanza hasta el estado FINISH.
- Si, por otro lado, es necesario el periodo dummy, el sistema irá intercalando estados de DUMMY en flanco de subida y bajada, y mediante un contador se llevará a cabo la cuenta de este periodo. Una vez se cumpla este tiempo se cambiarán las direcciones de las lneas y se avanzará hasta el siguiente estado: READING DATA. Los pulsos que desplazan los registros se activan en el estado TX_DATA, donde además se comprueba si se han transmitido todos los bits alcanzando el último estado FINISH; si no el estado siguiente es FALLING EDGE READ, donde se espera a un pulso de subida y el sistema pasa de nuevo al estado READING DATA.

5. En el último estado, se activa la señal de finalización de la secuencia y se espera la señal restar para volver al estado de inicio.

Los estados FINISH, RISING EDGE QSCK y FALLING EDGE QSCK cuentan además con un temporizador que marca el tiempo de ejecución del sistema, si este sobrepasa el valor del TimeOut, el proceso evolucionará hasta el estado de ERR; de donde volverá al estado inicial únicamente con un restart. El bloque contiene un puerto de salida que se activará si se ha producido un error de TimeOut.

En todos los estados de la secuencia, la llegada de la señal *restart* reiniciará el sistema evolucionando hasta el estado de reposo, IDLE.



[1] La siguiente transición no está representada para clarificar el diagrama
if restart_qspi = 1 then
n_state <= IDLE;
end if

[2] La siguiente transición no está representada para clarificar el diagrama
if TIME_COUNTER= (TIMEOUT_VALUE-1) then
n_state <= ERR;
end if

Figura 26: Máquina de estados QSPI Physical Layer

4.4.2.3 Registros de transmisión y recepción

Dentro del bloque *QSPI Transfer*, existen dos registros de desplazamientos:

- **Registro Rx:** registro de 32 bits que va almacenando los bits en serie conforme se reciben desde el maestro de la comunicación.
- **Registro Tx:** registro de 32 bits que contiene el dato a transmitir por el SPI Module, y va desplazando los bits para ir transmitiéndolos por la salida de nuestro módulo.

Dependiendo el modo seleccionado, los bits se desplazan en serie de uno en uno (modo SPI) o se van desplazando de 4 bits en 4 bits (modo QSPI) por cada pulso de la señal *shift_rx* o *shift_tx*, respectivamente. Además, el modo QPSI consta de un modo adicional para la escritura de los bits de insturcción iniciales, el cual transmite los bits de igual manera que en el modo SPI.

La representación del diagrama de bloques de estos registros se muestra en la Figura 27.

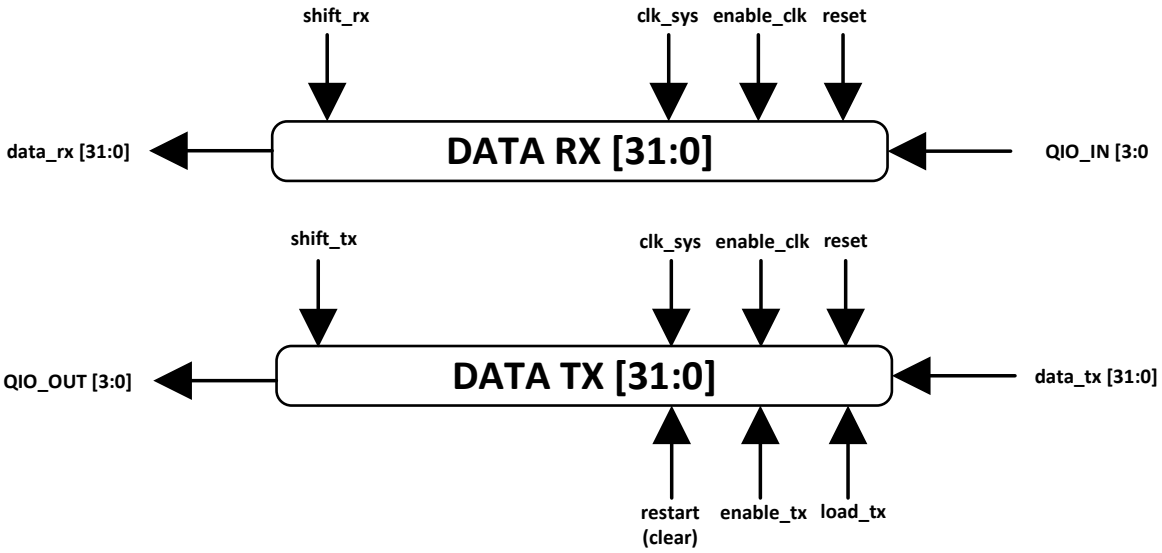


Figura 27: Registros de Transmisión – QSPI Transfer

4.4.3 Bloque Interpreter

El bloque *QSPI Interpreter* se encarga de procesar el byte de operación, para determinar si la operación debe realizarse sobre la memoria interna del System-on-Chip desarrollado o si, por el contrario, se debe efectuar sobre la memoria RAM existente. Para este último caso, este bloque tiene conexión al bus de direcciones y datos de dicha memoria. Por último, gestiona la señal "*rwn*" que permite identificar a los demás módulos la operación que se está realizando.

La Figura 28 muestra el bloque del módulo *QSPI Interpreter*.

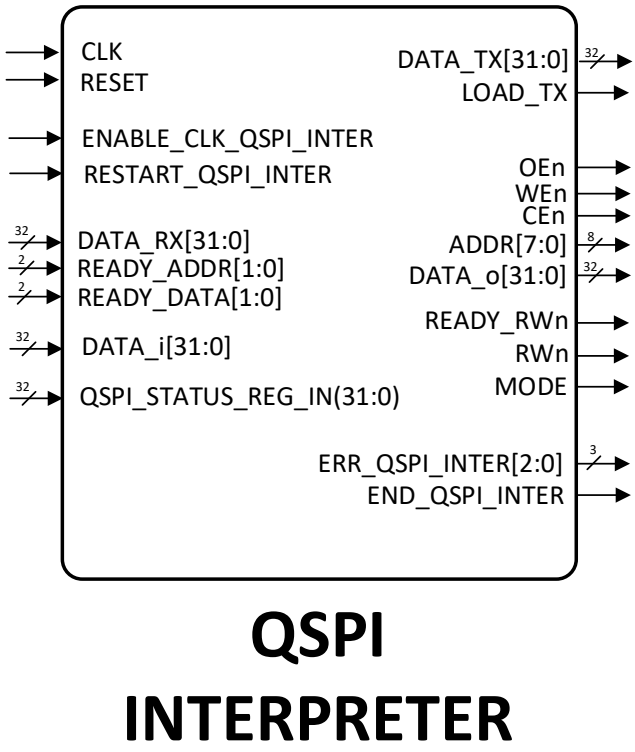


Figura 28: Bloque QSPI Interpreter

El módulo QSPI Interpreter se ha implementado como una máquina de estados que se presenta en la Figura 29. A continuación, se describen el funcionamiento de la máquina de estados:

1. En el estado de reposo, el bloque espera que le indiquen que los bits de instrucción (CMD) han sido recibidos satisfactoriamente, hasta ese momento el sistema no avanza al estado OPERATION.
2. En el estado OPERATION. se decide qué operación y sobre qué memoria se va a realizar. Se pueden distinguir los siguientes casos:
    - Si el bit de operación es '0', se elige una operación de escritura.
        - La escritura en el registro de estado está prohibida, y el intento de ello, hará saltar un error, pasando al estado ERR.
        - Si la dirección de escritura coincide con el primer registro de configuración, también se producirá un error, y el sistema avanzará hasta el estado ERR.
        - La escritura en el registro de configuración permite cambiar el modo de transmisión. Si la dirección coincide con éste, se avanza hasta el estado "WAIT DATA QSPI CONFIG", donde se espera a que se haya recibido todos los bits para cargarlo en el registro en el siguiente estado: WRITE_CONFIG. Desde este último, el sistema avanza de manera sucuencial hasta el último estado.

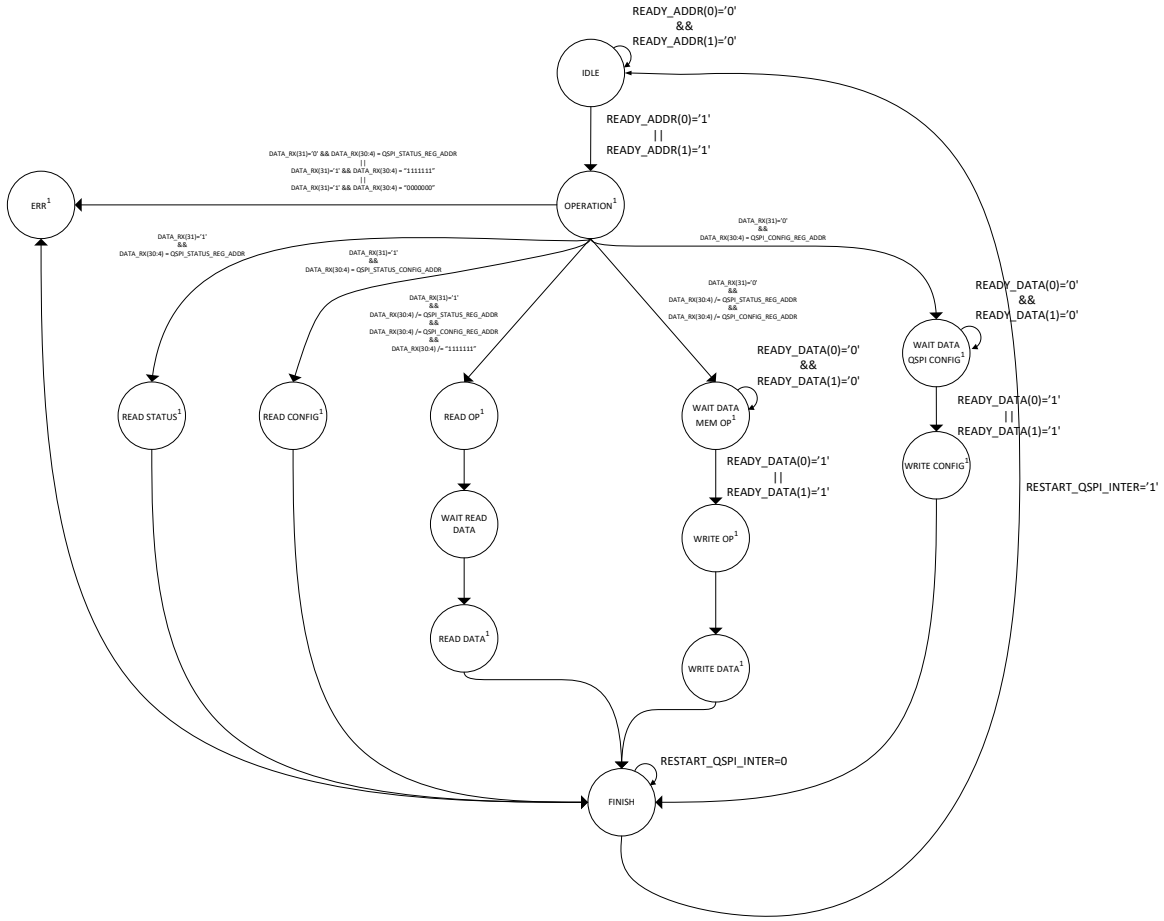
- Por último, si la dirección de operación no coincide con ninguno de estos casos, se escribirá en uno de los registros de memoria y se seguirá la siguiente secuencia:
    - El proceso a seguir será:
        - La habilitación de la salida de la memoria (*OEn*) en el estado WAIT DATA MEM OP.
        - Esperar la activación de la señal que indica que el dato ha sido recibido por el SPI.
    - Tras ello, se pasa al estado WRITE OP, donde se activa el reloj de la memoria (*CEn*), y la señal (*WEn*) a nivel bajo que habilita la escritura en memoria, se transfiere la dirección y el dato a escribir. Este es un estado transitorio que da paso al estado WRITE DATA.
    - En WRITE DATA, se desactiva el reloj de memoria, y el dato se mantiene en el puerto de entrada de memoria.
- La máquina de estados finaliza en el estado FINISH, donde se activa la señal de finalización del bloque; siempre que no se haya producido un error o no se haya activado la señal *restart*.

- Si el bit de operación es 1, la operación corresponde a una lectura.
    - La lectura en el registro 0x7F, es decir, en la última posición de memoria no está permitida.
    - La lectura del registro de estado sí es posible. Sin embargo, conlleva un camino diferente, debido a que supone, además, una limpieza del registro cada vez que se lea.
        - Al ser un registro interno del SPI, se cargará instantáneamente en el puerto de salida, y se activará la señal *rwn* que activa la operación de lectura.
    - Por último, si la dirección es diferente a los casos anteriormente descritos, se procederá a leer un registro de memoria.
        - El sistema avanzará hasta el estado READ OP, donde se activará el puerto de salida de la memoria (*OEn*), y la señal *rwn.*
        - En el siguiente estado WAIT READ DATA, se habilita el reloj de memoria (*CEn*), se coloca la dirección del registro y se manitene en nivel alto la señal (*WEn)* para indicar que es una lectura.
        - Por último, se transfiere el dato leído al QSPI Transfer en el último estado: READ DATA; y se desactiva el reloj de memoria.
    - Los diferentes procesos finalizan en el estado FINISH, donde se activa la señal de finalización del bloque SPI Interpreter.

- Si en cualquiera de los estados se activa la señal *restart*, el sistema vuelve al estado inicial de reposo ignorando la comunicación. Si se produce algún error en el estado de operación, se avanzará hasta el estado ERR, y, de manera consecuente, al estado FINISH, donde se activará la señal que indica la finalización del proces. Este paso devolverá al sistema al estado inicial únicamente con un *restart*.

El bloque consta de un vector de salida que identifica qué error se ha producido. Se contemplan los siguientes errores:

- Escritura en el registro de estado
- Escritura en el primer registro de configuración
- Lectura de la última posición de memoria.



[1] La siguiente transición no está representada para clarificar el diagrama

```
if restart_qspi_inter = 1 then
        n_state <= IDLE;
end if
```

Figura 29: Máquina de estados QSPI Interpreter

5 VERIFICACIÓN

Tras la implementación modular del sistema, en el presente capítulo se describe el procedimiento de verificación seguido para comprobar el correcto funcionamiento del sistema. Este sistema probará toda la funcionalidad del diseño, la robustez de la aplicación y la validación de las técnicas utilizadas en nuestro bloque. Para ello, se ha optado por una verificación funcional, en la que se han desarrollado nuevos bloques que permiten emular el comportamiento de agentes externos que ayudarán a validar las especificaciones planteadas en nuestra interfaz.

5.1 Test Bench

Se propone un sistema de verificación que lleve a cabo todos los tests de validación, simulando el maestro de de la interfaz QSPI desarrollada, y registrando la información en un fichero de texto. Este último paso es necesario para la posterior automatización de los resultados obtenidos, reduciendo la interacción del usuario con el sistema. Además, este test bench se ha planteado de forma que pueda escalarse fácilmente a medida que el desarrollo del System-on-Chip avance.

Con el fin de lograr lo anteriormente descrito, se ha desarrollado un sistema de verifcación que valide el correcto funcionamiento de la interfaz de comunicación en cada uno de sus casos de uso, y con la comprobación exhausta de su protocolo. Este sistema viene determinado por lo siguientes elementos:

- **Fichero de configuración normalizado**. Contiene los datos de cada uno de los registros a configurar por el maestro dependiendo de la prueba seleccinada.
- **Fichero de resultado normalizado.** Almacena los resultados obtenidos a través de la comunicación de la interfaz QSPI.
- **Bloques VHDL no sintetizables.** Entidades que emulan el comportamiento de los elementos externos a nuestro Módulo QSPI, y permiten, por otro lado, la configuración y ejecucción de las distintas pruebas.
- **Scripts de Matlab.** Realización de diferentes scripts de Matlab para automatizar la creación de los ficheros de configuración; facilitando así la realización de ficheros de configuración de todos los casos de uso.

Este tipo de estructura permite la realización de las pruebas de manera simplificada y la adaptación a los posibles cambios en el diseño. Por otro lado, adapta el entorno de verificación a la fácil automatización de configuración y creación de nuevas pruebas.

5.2 Arquitectura del sistema

El entorno de simulación contempla tanto los bloques dígitales implementados como los bloques auxiliaries de simulación. La arquitectura del entorno de prueba viene definida por los siguientes bloques.

- **QSPI TB.** Entidad superior que engloba los demás elementos del sistema. Su función principal es interconectar los diferentes bloques, aunando los bloques sintetizables con aquellos que no lo son.
- **FSM Control.** Máquina de estados que se encarga de configurar, ejecutar y almacenar los resultados de la prueba elegida.
- **QSPI Master.** Maestro de la comunicación QSPI, se encarga de iniciar la comunicación y ejecutar secuencialmente el protocolo a seguir por la interfaz QSPI.
- **QSPI Wrapper.** Entidad que agrupa el sistema desarrollado en el capítulo anterior junto con la memoria de doble puerto, DPRAM, así como como los PADs utilizados para conectar el dispositivo con el exterio.

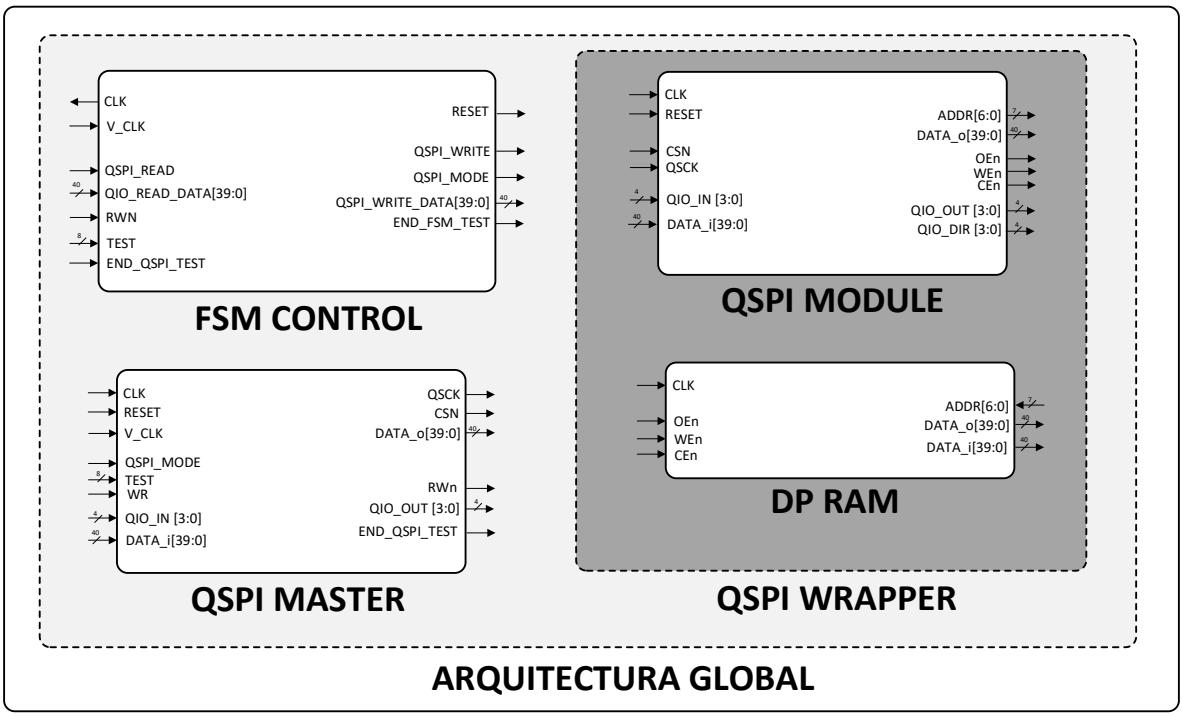


Figura 30: Arquitectura Global del Test Bench

5.2.1 Módulo QSPI Master

El módulo *QSPI Master* emula la transmisión de los bits de acuerdo con las especificaciones de dicha interfaz, según trabaje en modo SPI o QSPI. La información que debe transmitir se recibe del módulo *FSM Control*, a través del puerto *DATA_I[31:0]*.

Para ello, este módulo está compuesto de los siguientes bloques:

- **SPI Master:** Emula la capa física del maestro si el modo de transmisión elegido es el SPI. Ejecuta la secuencia de transmisión en función del test a realizar. Este módulo permite representar los diferentes casos de transmisión y forzar los diferentes errores que se quieren detectar en la comunicación. Se utiliza una secuencia distinta para cada prueba seleccionada, y una secuencia general para la lectura/escritura correcta del SPI.
- **QSPI Master.** Emula la capa física del maestro si el modo de transmisión elegido es el QSPI. Este bloque permite continuar con las pruebas diseñadas para el modo de comunicación QSPI. Al igual que en capa física del SPI, se tiene una secuencia distinta para cada una de las pruebas, y una secuencia general para la comunicación correcta del QSPI.
- **PADs.** Se añaden los PADs de interconexión de salida/entrada ya que las líneas de transmisión del maestro son bidireccionales. Estos tienen una señal de habilitación que marcan si debe actuar como entrada o salida.

La Figura 31 muestra el diagrama de bloques del módulo *QSPI Master*. En ella, se pueden observar tanto las entradas como las salidas, así como el tamaño de los buses utilizados. Por su parte, la descripción de cada una de estas señales se recoge en la Tabla 3.

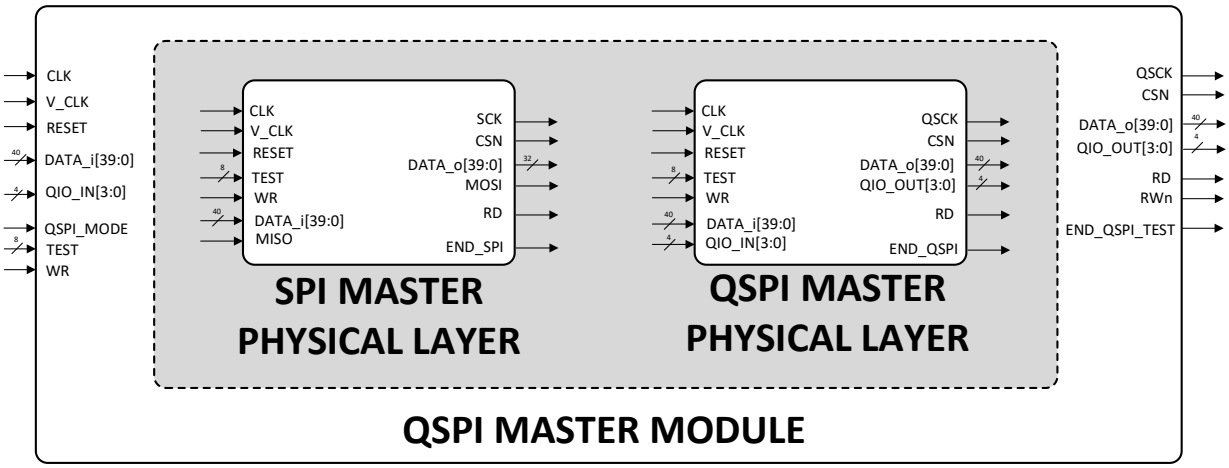


Figura 31: Arquitectura QSPI Master

Nombre Puerto	Entrada/Salida	Alcance	Descripción
Reloj y Reset			
CLK	Entrada	Externa	Señal de Reloj Sistema.
V_CLK	Entrada	Externa	Señal de Reloj Virtual de sincronización.
RESET	Entrada	Interna	Reset Síncrono (activo a nivel bajo).
Señales de Comunicación con el Slave			
CSn	Salida	Externa	Es el maestro quien gestiona la señal de Chip Select. Si esta señal permanece a nivel alto, la comunicación QSPI no se establece. Tras su bajada, la comunicación se iniciará. La comunicación siempre comenzará con QSCK a nivel bajo.
QSCK	Salida	Externa	Reloj que marca la temporización de la interfaz serie. Los datos presentes en QIO_OUT serán muestreados en el flanco de subida de QSCK. Los datos de salida en QIO_IN son actualizados en el flanco de bajada de QSCK, para que sean muestreados por el maestro en el flanco de subida.
QIO_OUT [3:0]	Salida	Externa	Salidas para transferir datos desde el esclavo hacia el maestro de la comunicación. QSPI maestro al chip. Será muestreada en los flancos de subida de QSCK.
QIO_IN [3:0]	Entrada	Externa	Entradas por la cual se reciben los datos desde el maestro de la comunicación. Los datos son actualizados en el flanco de bajada de QSCK.
Señales de Interconexión con FSM Control			
DATA_I [39:0]	Entrada	Interna	Datos leídos del fichero de configuración a transmitir por el maestro.
DATA_O [39:0]	Salida	Interna	Datos recibidos del SPI Slave y transferidos a la FSM Control para la escritura en el fichero de resultados.
RWn	Salida	Interna	Señal que indica a la FSM si la operación es de escritura o lectura.
WR	Entrada	Interna	Señal que indica que el dato, leido del fichero de configuración, está listo para ser transferido.
RD	Salida	Interna	Señal que indica que el dato leído ya está listo para la transmisión a la FSM.
TEST [3:0]	Entrada	Interna	Número de prueba a ejecutar.
END_QSPI_TEST	Salida	Interna	Marca el fin de la secuencia, y por lo tanto la finalización del test.

Tabla 3: Entradas/Salidas del bloque QSPI MASTER

5.2.2 FSM Control

La entidad *FSM Control* es una máquina de estados encargada de controlar el flujo de ejecución de las pruebas. Se comunica con el QSPI Master con el fin de cargar la configuración de las pruebas mediante una interfaz propia. También monitoriza las transmisiones del maestro QSPI hacia el esclavo. Sus principales funciones son:

- Lectura de ficheros de configuración, y carga de los datos al módulo *QSPI Master*, así como la habilitación de la comunicación de la interfaz.

- Gestionar la comunicación del *QSPI Máster* con el bloque implementado.
- Generar el reloj de sincronización de la entidad top.
- Recogida de los datos leídos por la interfaz de comunicación QSPI
- Desactivación de la comunicación
- Creación y escritura de los ficheros de resultados para su posterior procesamiento.

La Figura 32 muestra el esquema del bloque *FSM Control*, representando sus entradas y salidas.

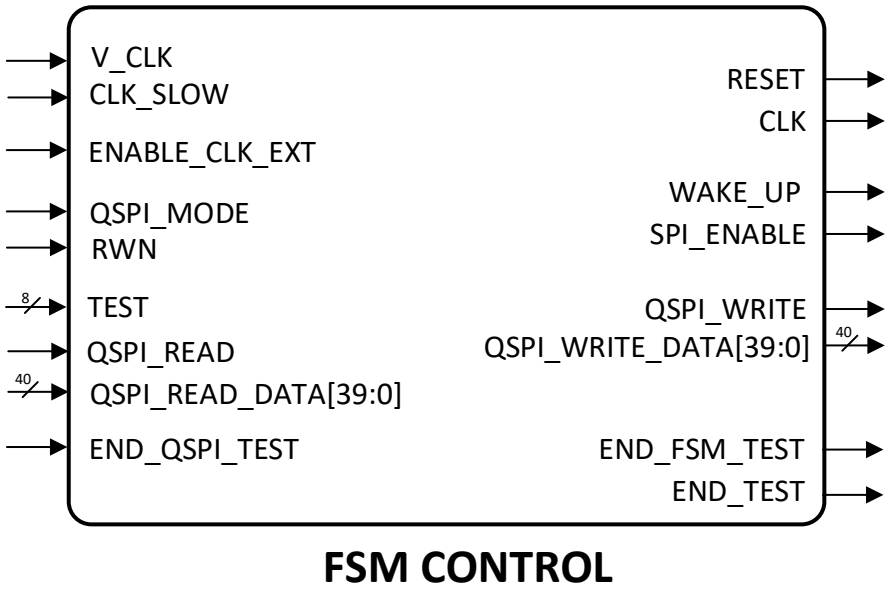


Figura 32: Bloque FSM Control

5.3 Sistema de Verificación

El diseño del sistema de verificación no sólo incluye la implementación de los bloques ya comentados, sino el desarrollo de un banco de pruebas que comprueben la funcionalidad de los distintots modos de funcionamiento, y validen la robustez del diseño ante fallos forzados. Para ello, se han diseñado 77 pruebas diferentes. Este conjunto puede dividirse en dos grandes bloques:

- **Bloque SPI.** Las 50 pruebas de este conjunto están destinadas a validar el funcionamiento de este modo, tanto en la transmisión como en la recepción. Dentro de este se encuentran:
    - **Polaridad y Fase CSN (Pruebas 1-5)**. Las 5 primeras pruebas están destinadas a la comprobación de la bajada de CSN al inicio de la comunicación, considerando siempre el valor del reloj QSCK en el momento de la bajada.
    - **Interrupción de la comunicación (Pruebas 6-45)**. Las 40 pruebas siguientes están orientadas a comprobar que el sistema detecta una interrupción de la comunicación, es decir, que se levanta el Chip Select en cualquiera de los bits de transmisión de la trama. El sistema debe detectarlo y resturar los registros a la espera de una nueva transmisión. Este error no puede ser bloqueante.

- **Otros errores (Pruebas 46 – 50).** Las últimas pruebas de este modulo verifican que no se produce ninguna escritura/lectura prohibida en memoria, así como el vencimiento del temporazador de la comunicación. También se incluye una prueba adicional para la comprobación de que las operaciones de escritura y lectura en la memoria son correctas.

- **Bloque QSPI.** Las 27 pruebas de este conjunto están destinadas a validar el funcionamiento de este modo, así como la conmutación entre ambas interfaces: SPI y QSPI. Dentro de este se distinguen los siguientes casos:

  - **Conmutación entre modos (Pruebas 51).** La primera prueba de este bloque está destinada a la comprobación del paso de un método de funcionamiento a otro.

  - **Interrupción de la comunicación (Pruebas 52 – 75).** Este banco de pruebas fuerza la interrupción de la transmisión en cada uno de los grupos de 4 bits de la trama levantando el Chip Select. Se distingue los diferentes métodos de operación: escritura y lectura; por esta razón, se obtienen un número total de 24 pruebas.

  - **Funccionamiento Correcto (76– 77).** Se proponen dos últimas pruebas que verifiquen la transmisión correcta en la comunicación QSPI, validando los dos métodos de operación.

En el bloque QPSI no se fuerzan los errores cometidos por el bloque *Interpreter*, ya que quedan verificados con las pruebas del SPI. Tampoco se vuelven a validar los errores de fase y polaridad, ya que la detección de estos fallos es independiente del método seleccionado.

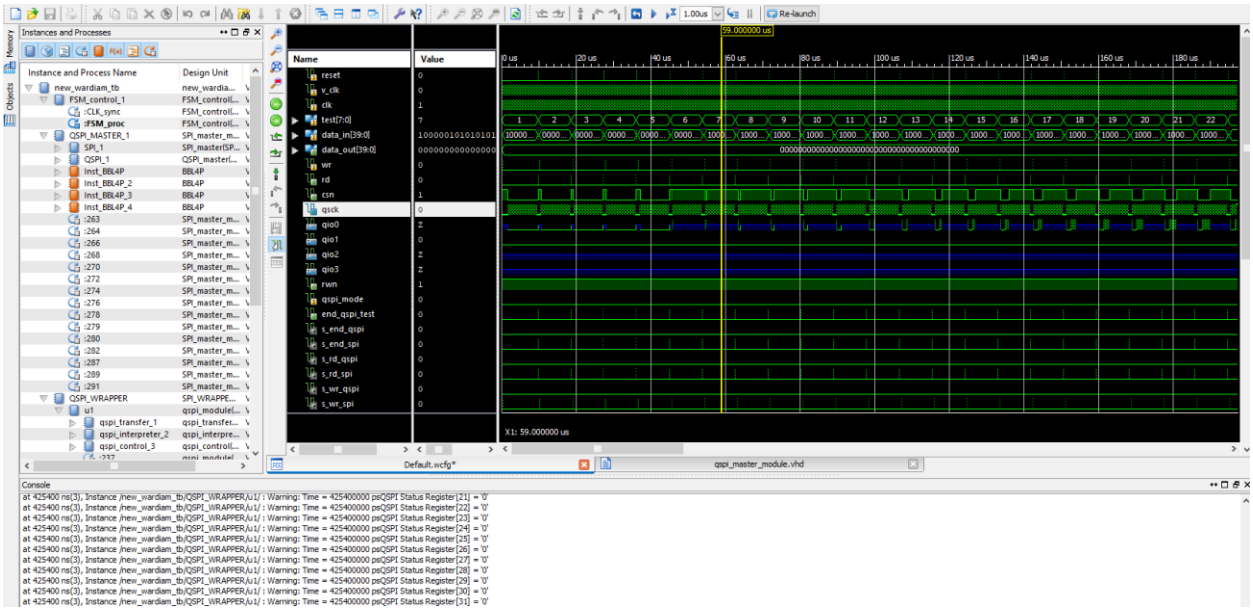


Figura 33: Cronograma ejemplo de los test

5.3.1 Estructura del fichero de configuración normalizado

Para la realización de las pruebas es necesario cargar una configuración predeterminada que ajuste los parámetros específicos de cada test. La *FSM Control* es quien se encarga de abrir el fichero de texto, leer la información y transmitir los datos al bloque *QSPI Master* para su transmisión al módulo esclavo. El fichero de texto está normalizado para unificar todas las pruebas a realizar: simulación RTL, a nivel de puerta y física. En el fichero, se pueden identificar las siguientes líneas:

- **Número de líneas** total a leer.
- **Número de repeticiones.** Esta línea permanece a 1 para la verificación de la entidad desarrollada.
- **Datos de configuración.** El resto de líneas del fichero define los 40 bits que debe transmitir el maestro de la comunicación. Estas pueden estar referidas a la memoria interna o a la memoria de doble puerto usada.

Estos ficheros son generados como paso previo a la ejecución del test mediante scripts de Matlab. De esta forma, el diseñador tan sólo necesita modificar los valores de los registros que desea cargar en estos scripts. Este hecho permite dotar al sistema de verificación de una interfaz sencilla y una automatización en la generación de los ficheros de configuración.

53.conf

```
1  91
2  1
3  0000001000000000000000000000000011110011
4  0000000100111111111100000000000000000000
5  0000001100000000000000000000001001011000
6  0000011100000000000000100111111110000000
7  0000100000000000000000000000000001100100
8  0000100100000000000000000000000001100100
9  0000101000000000000000000100111000100000
10 0000101100000000000000000000000000000101
11 0000110000000000000000000100111000100000
12 0000110100000000000000000000111110100000
13 0000111011111111111111111111111111111111
14 0000111100000000000000000000000001100100
15 0001000000000000000000000000000000010100
16 0001010000000000000000000000000010100100
17 0001011000010010000000000000000000000000
18 0001011100000000000000000000000001100100
19 0001100000000000000000000000000001100100
20 0001100100000000000000000100111000100000
21 0001101000000000000000000000000000000101
22 0001101100000000000000000100111000100000
23 0001110000000000000000000000111110100000
24 0001110111111111111111111111111111111111
25 0001111000000000000000000000000001100100
26 0001111100000000000000000000000000010100
27 0010001100000000000000000000000010100100
28 0010010100100100000001100111111110000000
29 0010011000000000000000000000000001100100
30 0010011100000000000000000000000001100100
31 0010100000000000000000000100111000100000
```

Figura 34: Fichero de configuración

Con el fin de lograr lo anteriormente descrito, y simplicar la generación de pruebas; los scripts de Matlab desarrollados se dividen en tres ficheros:

- **Constantes.** En este script se definen las constantes que se utilizarán en la definición de los registros.
- **Definición de registros.** Este script lleva a cabo la definición de los registros para cada uno de las pruebas. Se combinan las diferentes constantes para obtener los diferentes casos de uso, y se gestiona la organización de los registros para la fácil escritura en el fichero en el siguiente script.
- **Creación del fichero.** El último script se encarga de escribir los distintos registros de configuración en concordancia con las pruebas, indicar el número de líneas a escribir, y representar el número de iteraciones de la secuencia. De esta forma, se consigue obtener un fichero txt por cada prueba, y seguir una estructura normalizada.

La arquitectura seguida en la generación de los ficheros de configuración conlleva una simplificación a la hora de cambiar valores en las pruebas ya realizadas, y una fácil inclusión de nuevos tests a los ya desarrollados.

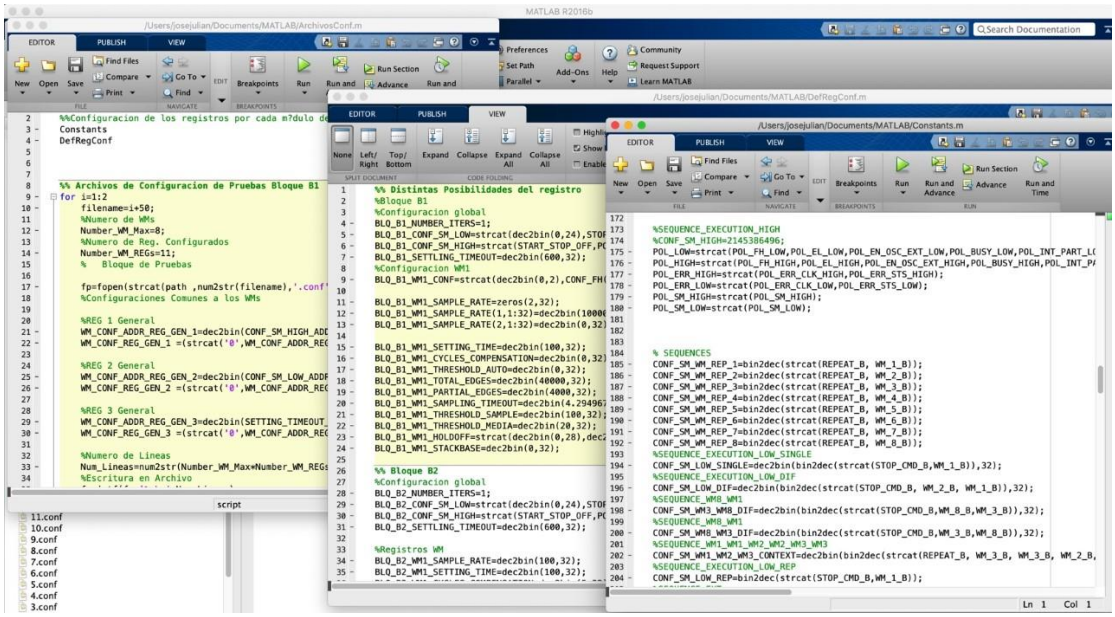


Figura 35: Scripts de Matlab

Por ejemplo, si quisiésemos cambiar el valor a escribir/leer en una de las transmisiones de comunicación, únicamente sería necesario cambiar el valor de esa constante en el primer script, y la generación de las nuevas pruebas con ese valor sería automática. Por otro lado, la definición de nuevos tests resulta muy fácil para el usuario, debido a que solo tiene que seleccionar las constantes que reflejan los modos de configuración que presenta ese caso de uso, y las constantes de los valores a transmitir.

5.3.2 Estructura del fichero de resultados normalizado

La información generada por la ejecución de cada prueba es almacenada en un fichero de texto cuya estructura se muestra en la Figura 36. Al igual que en el caso anterior, la FSM Control está encargada de gestionar el fichero de resultados, en este caso, es esta quien crea el fichero y realiza la escritura de los resultados obtenidos. Con el fin de lograr una serie ficheros unificados, se sigue una estructura normalizada en la creación de ellos recogiendo la información relevante de la secuencia que se describe a continuación:

- **Información de la prueba.** En la primera parte del fichero se muestra la información del test: número de prueba, número de líneas de configuración
- **Datos escritura/lectura.** El fichero de resultados recoge los datos devueltos tras la escritura de configuración por el SPI, y una posterior lectura de los registros configurados para la comprobación de una correcta escritura.
- **Valor temporal.** Todos los datos monitorizados se recogen cronológicamente acompañados de una pequeña decripción que indica el tiempo en el que se han tenido lugar.

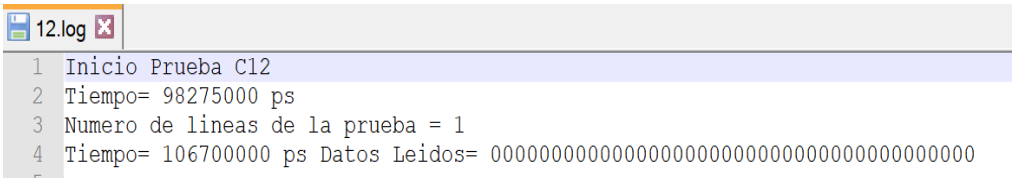


Figura 36: Fichero de resultados

5.4 Verificación Experimental

De forma paralela a la verificación por simulación, el bloque digital ha sido evaluado experimentalmentepara determinar sus prestaciones. Para ello, se ha hecho uso de un sistema de desarrollo basado en una FPGA del fabricante "Digilent Inc" [8] y una tarjeta de evaluación microcontroladora, recogidos en la Tabla 4. A su vez, para la programación de ambos sistemas de desarrollo se han utilizado los entornos suministrados por los fabricantes. En el caso del kit de evaluación ZedBoard [9] con el SOC Zinq-7000, se ha empleado el software Vivado Design Suite [10]; mientras que para el microcontrolador, se ha empleado la plataforma mBED [11].

Sistema de desarrollo	Modelo
FPGA	ZINQ 7000 - ZedBoard
Microcontrolador	Tarjeta STM32F207ZG Nucleo

Tabla 4: Elementos de desarrollo de la verificación

En primer lugar, se realiza la verificación de comunicación del microcontrolador con la FPGA emulando el comportamiento que tendrá la interfaz de comunicación en modo SPI. Para la realización de estas pruebas han

sido necesarios los siguientes elementos:

- **Programa Python.** Desarrollo de un programa en Python que leyese los ficheros de configuración, y cargase mediante una comunicación en serie los datos obtenidos en el microcontrolador.
- **Programación Microcontrolador.** Se desarrolla un ejecutable que permita recoger los datos por UART directamente de la computadora, los convierta, y los transmita por SPI a la tarjeta ZedBoard. De esta forma, el microcontrolador se establece como maestro de la comunicación, ejecutando las mismas funciones así que el módulo QSPI Master.
- **Implementación Interfaz QSPI Slave.** Se implementa el módulo QSPI desarrollado en la FPGA, y se configuran los pines para la recepción de datos por parte del microcontrolador. Además, se realiza una configuración particular de los pines con el fin de obtener el error concreto en el caso de que se produjese algún fallo.

5.4.1 Módulo Python

Este módulo se basa en una ejecutable programmable en lenguaje Python que permite efectuar la verificación experimental. La función principal de este programa es determinar el test a ejecutar, leer los ficheros de configuración correspondientes a dicho test y cargar la información en el microcontrolador.

El módulo ejectuable de Python se ha implementado como una máquina de estados que presenta el diagrama de estados mostrado en la Figura 37.

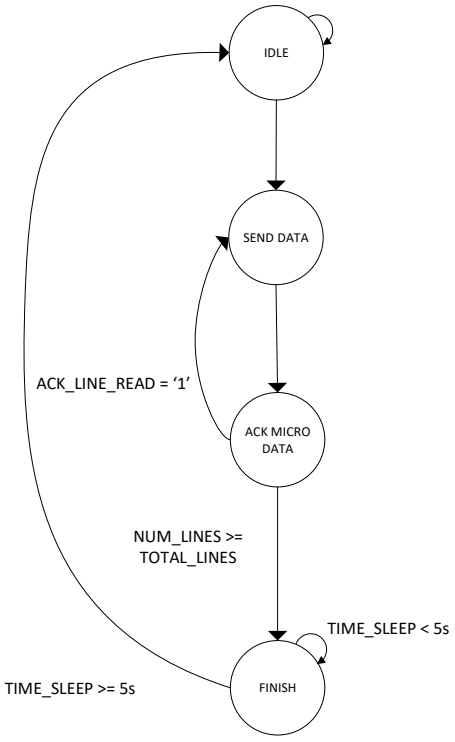


Figura 37: Máquina de estados Módulo Python

A continuación, se describe la máquina de estados del módulo Python:

1. En el estado inicial, se procode a la lectura de las líneas de configuración del test, y carga de los datos en un array. Además, en este estado se le envía el número de líneas leidas al microcontrolador.
2. El siguiente estado, SEND_DATA, en primer lugar, se muestra por pantalla el número de test y el número de línea que se enviará. Tras esto, se envía la informacion de la línea seleccionada, y se muestra por pantalla un mensaje para confirmar el envio.
3. En este estado, el sistema permanence hasta que se recibe un ACK del microcontrolador, cuando es así muestra un mensaje por pantalla. Si en este paso el número de líneas que se han envíado es igual que el número total del archivo de configuración, se avanza hasta el estado FINISH.
4. En este último estado, se establece un tiempo de espera para que el microcontrolador termine la comunicación con la FPGA. Tras esto, se vuelve al estado inicial.

5.4.2 Módulo Microcontrolador

Este módulo se basa en la unidad STM32F207ZG y la programación de un ejecutable en la plataforma mBed. El microcontrolador integra la comunicación con el ordenador y la FPGA, transforma los datos para poder ser transmitidos a la FPG mediante SPI.

El módulo ejectuable de Python se ha implementado como una máquina de estados que presenta el diagrama de estados mostrado en la Figura 38.

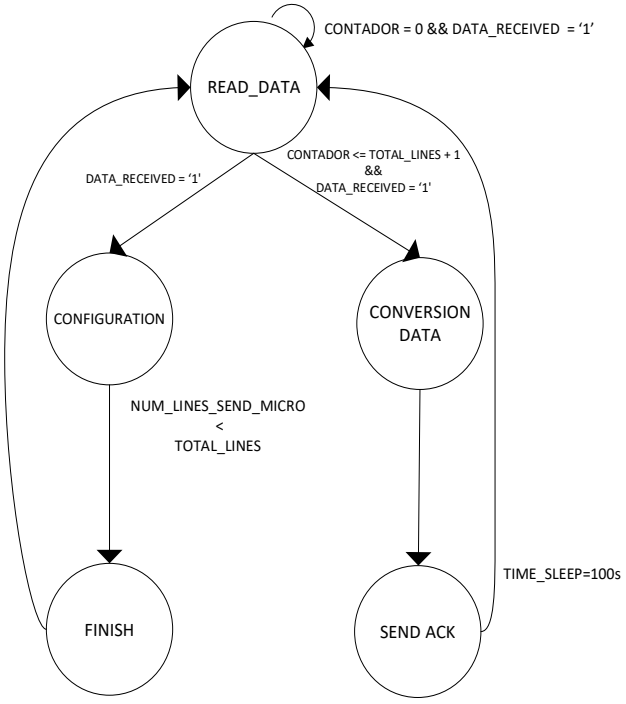


Figura 38: Máquina de estados del microcontrolador

A continuación, se describe la máquina de estados del módulo del microcontrolador:

1. El primer estado, READ DATA, se encarga de recibir los datos enviados por el computador. Tras la recepción de un paquete de 40 bits, dependiendo de la cuenta del contador se distinguen estos casos:
    a. Si el contador del sistema es igual a '0', el microcontrolador está recibiendo el número de líneas totales del archivo de configuración y, por lo tanto, el número de paquetes de 40 bits a transmitir posteriormente a la FPGA. En este caso, el sistema avanzará hasta este mismo estado en el siguiente ciclo.
    b. Durante el periodo, donde el contador es menor al número de líneas total (y ya se ha superado el primer ciclo); el microcontrolador recibe los datos que más tarde enviará mediante SPI a la FPGA. El siguiente estado al que avanza el sistema es CONVERSION_DATA.
    c. En el último caso, si no se cumple ninguno de los requisitos anteriores, el sistema avanza hasta el estado CONFIGURATION; ya que la carga del archivo de configuración ha finalizado, y el microcontrolador contiene toda la información a transmitir.
2. En el estado CONVERSION_DATA, el sistema tranforma los datos recibidos en bits a decimal. Separando los bits de instrucción y los datos a transmitir. El siguiente estado corresponde con el envió de ACK: SEND_ACK.
3. En el estado, SEND_ACK, el microcontrolador envía un mensaje de ACK al ordenador para que el programa Python continue con su ejecución. El siguiente estado al que avanza el sistema es, de nuevo, READ DATA.
4. Si por otro caso, se avanza hasta el estado CONFIGURATION, el microcontrolador envía mediante SPI todos los datos que habían sido cargados desde el ordenador. El siguiente estado es FINISH.
5. Por último, el sistema envía un último mensaje al computador que indica que ha terminado la secuencia del test y que, por lo tanto, puede iniciarse uno nuevo, en caso de que se hubiese configurado así. El sistema avanza hasta READ DATA, y permanece a la espera de que reciba un nuevo mensaje para avanzar.

5.4.3 Descripción General del Sistema Verificación

Por último, los pasos a seguir para la verificación experimental del sistema en general se describen a continuación:

1. Selección de la prueba a realizar, lectura del fichero de texto y almacenamiento de los datos en una estructura interna.
2. Envío mediante UART de los registros de 40 bits al microcontrolador. Para acreditar la correcta

recepción de los datos por parte del microcontrolador, este le envía un ACK que permite continuar el proceso de envío desde Python.

3. Una vez recibidos todos los registros de configuración, el microcontrolador adapta los datos recibidos para la correcta transmisión por SPI hasta la FPGA, y los envía mediante esta interfaz de comunicación.
4. Tras la transmisión completa por el SPI, el microcontrolador envía un ACK para confirmar que ha terminado su comunicación con la FPGA permitiéndole así al sistema que continúe con la siguiente prueba si así está configurada.

6 CONCLUSIÓN

En el presente trabajo se ha realizado un estudio de las principales interfaces de comunicación que existen en la actualidad, indagando en las más utilizadas dentro del mundo de la microelectrónica. Además, se ha hecho una comparativa entre ellas, destacando sus ventajas y desventejas y concluyendo finalmente cual se adaptaba mejor a los objetivos marcados al inicio.

A continuación, se ha llevado a cabo un estudio del estado del arte que introducen las técnicas de bajo consumo en circuitos dígitales, y que más tarde han sido utilizadas en nuestro diseño de interfaz QSPI para cumplimentar lo requerimentos establecidos.

Por otro lado, se ha desarrollado a nivel de esquemático una interfaz de comunicación (QSPI) con un diseño modular que permite la implementación de una de las técnicas más utilizadas de bajo consumo: clock gating.

Finalmente, se ha realizado la verificación de las distintas funcionalidades y de la robustez de nuestro sistema mediante un banco de pruebas propuesto. En primer lugar, se realizó una verificación de todos los casos de uso, y una detección de los errores en simulación; y más tarde una verificación del sistema implementado físicamente.

De este modo, se puede concluir el diseño, implementación y verifiación a nivel RTL de la interfaz de comunicación planteada al principio. Sin embargo, este proyecto de investigación deja posibles líneas de trabajo para continuar que podrían ser:

- Sintesís de los módulos digítales implementados y simulación haciendo uso de una tecnología CMOS.
- Implementación física de la entidad desarrollada.
- Verifcación de la interfaz post-sintesís.

Realizando lo anteriormente descrito, el proceso de desarrollo de un chip digital quedaría finalizado, y listo para fabricar. De esta manera, se consiguiría completar todas las fases de diseño, ya que se ha evolucionado desde la fase de definición de requisitos hasta su validación en el laboratorio.

7 BIBLIOGRAFÍA

[1] Federic Leens. An introduction to I2C and SPI Protocols. IEEE Instrumentation & Measurement Magazine. February, ***2009****.*

[2] Junwei Zhou and Andrew Mason. Communication Buses and Protocols for Sensor Networks. Department of Electrical and Computer Engineering, Michigan State University, **2002**, *2, 244-257.*

[3] Patrick Girard, Nicola Nicolici and Xiaoqing Wen. Power-Aware Testing and Test Strategies for Low Power Devices. Springer. February, ***2010****.*

[4] Anand N, George Joseph and Suwin Sam Oommen. Performance Analysis and Implementation of Clock gating techniques for Low power applications. IEEE Instrumentation & Measurement Magazine. IEEE-3233, ***2014****.*

[5] S. Henzler. Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies. Springer. February, ***2007****.*

[6] Thucydides Xanthopolus. Clocking in modern VLSI systems. Springer. ***2009****.*

[7]Etienne Le Sueur and Gernot Heiser. Dynamic Voltage and Frecuency Scaling: The Laws of Diminishing Returns. NICTA – University of New South Wales

[8] Digilent, Inc https://www.digikey.es/

[9] ZedBoard http://zedboard.org/product/zedboard

[10] Vivado https://www.xilinx.com/products/design-tools/vivado.html

[11] mBed https://www.mbed.com/