UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo de un módulo I2C con mecanismos de tolerancia a fallas en FPGA

T E S I S

Que para obtener el título de

INGENIERO ELÉCTRICO ELECTRÓNICO

PRESENTA:

Guillermo Raúl Saint Martin Robles

DIRECTOR DE TESIS:

Dr. Saúl de la Rosa Nieves

Ciudad Universitaria, CDMX. 2023

"En tres semanas sale..."
(Guillermo Saint Martin, 2022)

Dedicatoria

A Edith, mi madre. Gracias por orientarme, por confiar en mi capacidad y por siempre motivarme a lograr lo que me propongo. Ni esta, ni ninguna vida me servirá para devolverte todo lo que has hecho por mí.

A mi hermana, Estrella, por ser el pilar de mi vida. Que sepas que a cada paso que des, estaré yo contigo para apoyarte.

A mi abuelita, mi *mamá Maru*, por su apoyo incondicional y por nunca quitar el dedo del renglón. Sin tu ayuda, nada de lo que soy ahora habría sido posible.

A mi papá *Robles*, porque te prometí conseguirlo. Fue un gusto compartir un poco de tu vida terrenal para recibir tu consejo y tus anécdotas que tanto extraño.

Igualmente, agradezco a los demás miembros de mi familia por estar presentes en todo momento, gracias por su apoyo.

A Paola, gracias por tu amistad sincera y por brindar tu apoyo constante durante todos estos años, es reconfortante tener gente como tú a mi lado.

A Ciro, Joshua, Nicole, Vanesa y a todos mis amigos y compañeros del *LIESE*, gracias por su compañía y por los buenos momentos.

Agradezco igualmente al Dr. Saúl y al M.I. Aldair, por su disposición, su ayuda y sobretodo por los consejos que me brindaron a lo largo de todo este proyecto.

Para finalizar, gracias a toda la gente que me ha acompañado en algún momento de mi vida: Andrea, Ceci, Chucho, Jorge, Mary, Max entre muchos otros más. Que sepan que les aprecio y que han marcado una parte de mi ser.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Presentación del problema

Debido a que en las misiones espaciales los sistemas electrónicos de los satélites se encuentran especialmente susceptibles a la radiación del medio ambiente espacial, es necesario garantizar la fiabilidad de estos sistemas ya que los efectos de las partículas cargadas que componen estos ambientes pueden generar una modificación en el comportamiento del dispositivo y con ello derivar en la corrupción de la información e incluso provocar la pérdida de la misión.

A día de hoy con el desarrollo de los CubeSat es normal encontrar propuestas de sus módulos SCMI[1] montados con componentes COTS[2] a los que se les aplican mecanismos para mitigar los efectos de lo anteriormente mencionado. Sin embargo, con las tecnologías como circuitos integrados únicamente se pueden realizar cambios en sus elementos de memoria, mas no pueden ser re-configurados en su estructura interna si esta ya fue dañada severamente, por lo que al perderse un elemento crítico en la configuración, no es posible restablecerlo.
Esto se convierte en un problema prioritario en el caso de los módulos IIC ya que estos son bastante empleados dentro de los sistemas de CubeSats creados con componentes COTS y esto es debido a que los mismos dispositivos normalmente ya incluyen internamente este tipo de comunicación por defecto, debido a su enfoque comercial.

Con el propósito de reducir estos efectos, es necesario emplear tecnologías más flexibles que permitan la protección y hasta el reemplazo del hardware interno. Tecnologías como los FPGAs permiten la modificación en su descripción interna. Sin embargo, a estos igualmente se le deben agregar mecanismos de tolerancia a fallas que los prepare para estos ambientes agresivos y que asegure la correcta comunicación con los demás sistemas del satélite.

1.2. Hipótesis

Un diseño desde el más bajo nivel de abstracción para un módulo I2C descrito en VHDL y configurado en FPGA permite una mayor flexibilidad para aplicar mecanismos de tolerancia a fallas que aumenten la protección contra los SEE.[3] lo cual resulta en una propuesta capaz de asegurar una fiabilidad suficiente para establecer comunicación IIC entre módulos en misiones espaciales.

1.3. Objetivo

Desarrollar un módulo I2C capaz de actuar como periférico para un microprocesador y que además pueda garantizar la continuidad y confiabilidad del dispositivo embebido dentro de un FPGA que estará preparado para funcionar como parte del SCMI de un satélite bajo el estándar CubeSat, que se encontrará sometido a las condiciones de radiación espacial presente en las órbitas bajas de la tierra (LEO[4]) dentro del rango que va

[1]Sistema de Comando y Manejo de Información
[2]Commercial Off-The-Shelf: Productos comerciales y sin restricción de acceso
[3]Single Event Effects: Eventos causados por la interacción con partículas de alta energía
[4]Low Earth Orbit

desde los 400 hasta los 700 [km].

1.3.1. Objetivos particulares

- Desarrollo del módulo I2C con la mayor eficiencia en elementos lógicos posible.
- Identificación de los elementos críticos para la operación del sistema.
- Introducción de mecanismos de tolerancia a fallas para el periférico en los elementos críticos.

1.4. Justificación

Debido a que en instituciones y emprendimientos con presupuesto limitado es una opción más viable el empleo de componentes COTS dentro del desarrollo de nanosatélites bajo el estándar CubeSat, es común encontrar que estos módulos incluyen comunicación por medio de I2C, por lo tanto se convierte en una prioridad generar un módulo capaz de comunicarse con esta clase de componentes que además ofrezca ventajas como su fiabilidad en ambientes con radiación espacial.

Capítulo 2

Marco teórico

2.1. Especificación Inter-Integrated Circuit - NXP Rev. 7.0

El protocolo I2C es una especificación de comunicación serial síncrona con orientación a trabajar sobre 8 bits. Este fue desarrollado en la década de los ochentas por Phillips Semiconductors. Su nombre proviene de las siglas de Inter-Integrated Circuit y a menudo es igualmente abreviado como Inter-IC e IIC.

Este protocolo fue creado con la intención de brindar una interfaz de comunicación sencilla entre controladores y otros circuitos integrados, de ahí el nombre.

Para su funcionamiento emplea dos líneas de transmisión bidireccionales llamadas SDA[1] y SCL[2], la primera corresponde a la línea de datos; mientras que la segunda es el reloj que brinda sincronía a la comunicación entre los módulos maestro y esclavo.

La cantidad de módulos a conectar depende teóricamente de la cantidad de bits empleados para la dirección del esclavo; normalmente se utilizan 7 bits de direccionamiento, esto nos permite conectar hasta 127 componentes. Aún así, como se ve en la Tabla 2.1 hay opción de tener un direccionamiento de hasta 10 bits, aumentando la cantidad de direcciones para dispositivos hasta los 1023; sin embargo, debemos considerar que físicamente las líneas contienen capacitancias parásitas, pero este limitante se estudia más a fondo en la subsección 2.1.1.

El protocolo ofrece la ventaja de que contempla una arquitectura multi-controlador, esto quiere decir que más de un módulo puede tomar el control de las líneas; para esto es necesario agregar un sistema de arbitraje que asegure que en la competencia por obtener el control no se corrompan los datos.

La generación del reloj es una tarea encomendada a quien tenga el rol como maestro, las señales de reloj únicamente se permiten ser alteradas cuando se tiene un esclavo con velocidades de transmisión más lentas que las que exige el maestro; para esto recurre a mantener baja la línea de reloj.

En la Tabla 2.1 se muestran las características que ofrece el protocolo, catalogándolas como obligatorias, opcionales y no aplica (n/a), según la arquitectura que se planea usar.

2.1.1. Características físicas

Líneas de transmisión

De principio, su operación está enfocada en dos líneas principales: SDA y SCL. Estas por especificación están sujetas a trabajar con módulos que internamente tienen una configuración del tipo colector abierto, por lo que se necesita una fuente de corriente; normalmente es una configuración de resistores en Pull-Up que mantenga las líneas en alto mientras se encuentra en estado inactivo; entonces, únicamente se tienen que tener conectadas ambas líneas a una fuente de poder como se muestra en la Figura 2.1.

Este tipo de configuración además de requerir sólo 2 terminales, ofrece la ventaja de que para agregar o quitar nuevos módulos no se requiere de modificar el hardware de los demás integrados, volviendo la comunicación entre los mismos bastante simple.

[1]Serial Data
[2]Serial Clock

Características	Configuración		
	Un maestro	Multi-Maestro	Esclavo
Condición de Start	M	M	M
Condición de Stop	M	M	M
Acknowledge	M	M	M
Sincronización	n/a	M	n/a
Arbitraje	n/a	M	n/a
Estrechamiento de reloj	O	O	O
Direcciones de 7 bits	M	M	M
Direcciones de 10 bits	O	O	O
Dirección de llamada general	O	O	O
Reinicio por software	O	O	O
Byte de start	n/a	O	n/a
Identificador de dispositivo	n/a	n/a	O
M = Obligatorio, O = Opcional y n/a = no aplica.			

Tabla 2.1: Catálogo de características en módulos de I2C.[1]

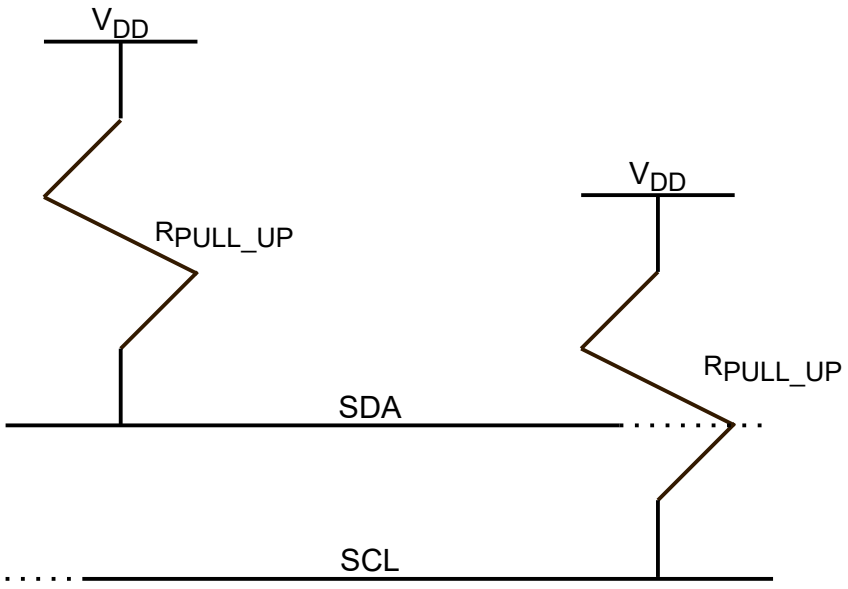


Figura 2.1: Ejemplo de conexión de las líneas en IIC. [1]

Para los modos de High-Speed y Ultra-Fast, que son los más rápidos, se requiere una modificación interna de los módulos y aunque la restricción para las capacitancias en la línea es mayor, se pueden emplear puentes que seccionen la línea que ayuden a alcanzar los parámetros de capacitancia requeridos en la especificación. Aún así, existe la ventaja de que todos los dispositivos son capaces de trabajar a tasas de transmisión menores a las que fueron diseñados, por lo que si no existen los elementos para disminuir las capacitancias, se puede recurrir a trabajar con tasas de transmisión más lentas.

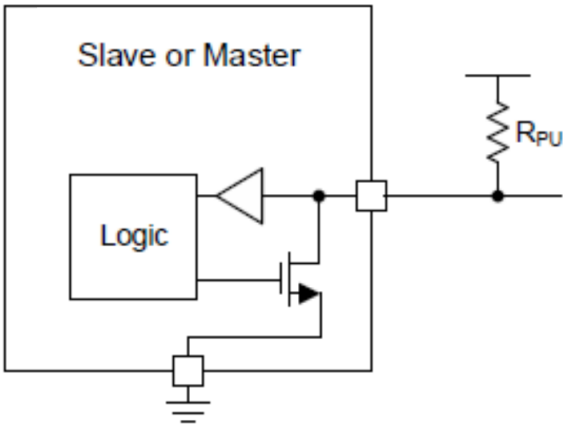


Figura 2.2: Equivalente de la estructura interna en un módulo I2C.

Velocidades de operación

Al inicio de la especificación se estableció una velocidad de transmisión de 100 [kbits/s]. A partir de ahí se fueron agregando con el tiempo las velocidades vistas en la Tabla 2.2.

Todos los dispositivos pueden ser operados a velocidades inferiores de las que fueron diseñados, únicamente el Modo Ultra-Fast no es compatible con otras versiones puesto que este tiene la característica de ser unidireccional.

Para los modos desde el Estándar hasta el Fast-Plus se mantienen las recomendaciones en cuanto a sus conexiones, es decir: las resistencias, la estructura interna de los módulos y las tramas son similares.

Una vez que llegamos al modo Hs, encontramos modificaciones internas y en las tramas, como que existe un código de controlador que no se envía en otras configuraciones de trama; aunque en principio existe retrocompatibilidad con velocidades menores por lo que aún puede trabajar sobre el mismo bus. Mientras que para el modo Uf las líneas son unidireccionales y se puede prescindir de las resistencias de Pull-Up, pero deja de ser compatible con dispositivos bidireccionales.

Abreviatura		Modo	Tasa de transmisión de bits
F/S	Sm	Estándar	+100 [kbits/s]
	Fm	Fast	+400 [kbits/s]
	Fm+	Fast-Plus	+1 [Mbits/s]
Hs		High-Speed	+3.4 [Mbits/s]
UFm		Ultra-Fast	+5 [Mbits/s]

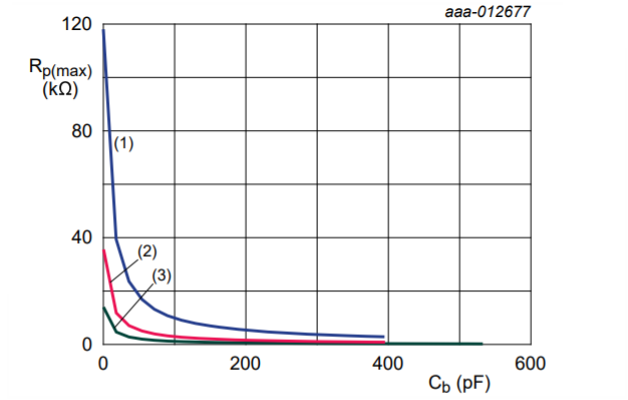
Tabla 2.2: Velocidades de transmisión usadas en módulos de I2C.

Cálculo de resistencias y análisis de limitaciones físicas

Como se mencionó en la Sección 2.1.1, se utilizan unas resistencias en configuración Pull-Up como fuentes de corriente que mantengan las dos líneas en estado alto.

Además debemos considerar que al tratarse de líneas físicas compuestas por alambre, la comunicación se ve sujeta a capacitancias parásitas producto del mismo material, de

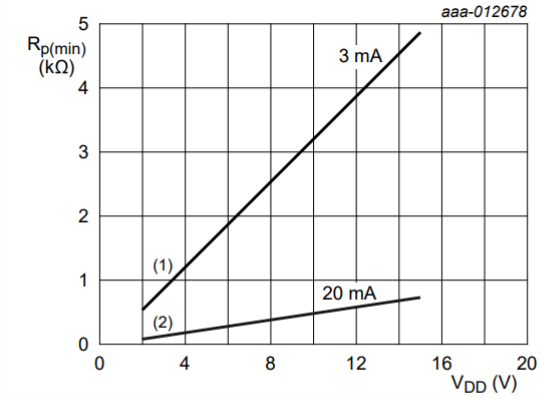
las conexiones y sus terminales; esta capacitancia afecta al tiempo de levantamiento de la línea y se define una capacitancia máxima de la línea de 400[pF] para los modos Estándar y Fast; mientras que para el modo Fast-Plus permite hasta 550[pF].



1. Standard-mode
2. Fast-mode
3. Fast-mode Plus

$R_{p(max)}$ como función de la capacitancia de la línea

Figura 2.3: $R_{PULL-UP(MAX)}$ como función de la capacitancia en la línea. [1]



1. Fast-mode and Standard-mode
2. Fast-mode Plus

$R_{p(min)}$ como función de V_{DD}

Figura 2.4: $R_{PULL-UP(MIN)}$ como función de V_{DD}. [1]

Debido a esto, el cálculo de las resistencias de Pull-Up se hace considerando las gráficas que se muestran en las Figuras 2.3 y 2.4. De ellas se desprenden las relaciones 2.1 y 2.2 que sirven para obtener los valores de resistencia mínima y máxima necesarios para cumplir con los requisitos eléctricos y de tiempo de la especificación.

$$R_{PULL-UP(MAX)} = \frac{t_r}{0{,}8473 * C_b} \tag{2.1}$$

Donde C_b = Capacitancia del bus
t_r = Tiempo de levantamiento

t_r [ns]	Modo
1000	Estándar
300	Fast
120	Fast-plus

Tabla 2.3: Tiempos máximos de levantamiento de señales para modos *F/S*.

$$R_{PULL-UP(MIN)} = \frac{V_{DD} - V_{OL_{MAX}}}{I_{OL}} \tag{2.2}$$

Donde V_{DD} = Voltaje de la fuente
$V_{OL_{MAX}}$ = Voltaje de salida cuando el bus está en estado bajo ≈ 0.4 [V]
I_{OL} = Corriente de salida cuando el bus está en estado bajo

I_{OL} [mA]	Modo
3	Estándar
3	Fast
20	Fast-plus

Tabla 2.4: Corrientes de fuga cuando la línea está en estado bajo.

Una vez establecidas las relaciones para obtener el valor mínimo y máximo de la resistencia, el bus estará completo para trabajar sobre los modos de transmisión F/S, pues como se explicó en la Sección 2.1.1, las limitantes para los modos más veloces son más estrictas en cuanto a la capacitancia y la estructura de las líneas.

2.1.2. Interfaz de comunicación

En esta parte, se detalla cómo realizar la comunicación dentro de los módulos I2C a través de sus modos, se agregan los diagramas de flujo que definen a las distintas operaciones y se muestran las configuraciones de sus respectivas tramas.

Antes de mostrar esto, hay que definir que existen 4 tipos de configuraciones posibles para los dispositivos que emplean I2C:

- Maestro o controlador

  Transmisión

  Recepción

- Esclavo u objetivo

  Transmisión

  Recepción

En todos ellos se emplea una trama como la que se muestra en la Figura 2.5. En ella se puede observar qué partes de la trama corresponde formar a cada una de las configuraciones y cuáles dependen del modo en que sean configurados el maestro y esclavo. Más adelante, se examinarán las tramas particulares de cada modo. Además, se puede ver que los elementos constantes en la comunicación son las condiciones de Start, Stop, Acknowledge y Not Acknowledge.

Figura 2.5: Estructura de trama básica.

Condición de Start/Stop y bit de modo

Las condiciones de Start y Stop se definen por ser una transición de estados en la línea SDA de alto a bajo, para el Start y de bajo a alto para el Stop; mientras que la línea de SCL está en alto como se muestra en la Figura 2.6. Es necesario que el maestro sea el encargado de generar ambas condiciones de inicio y paro.

La condición de Start se encarga de indicar a los módulos que el bus estará en estado activo y por ende, ocupado. Consecuentemente, esta condición será seguida por los bits de dirección junto con un modo de operación. Las condiciones de Start y Start repetido son idénticas.

Mientras tanto, la condición de Stop se encarga de informar a los módulos que la transmisión se detendrá y que por lo tanto, el bus estará libre; por lo que internamente los módulos deberán regresar a su configuración inicial, esto con la finalidad de que cuando vuelvan a tener señales de reloj, no continúen en la configuración previa; ya sea en forma de recepción de datos o de transmisión y que en su lugar tendrán que esperar a recibir tramas con la dirección de esclavo y el modo envío.

El bit de modo se encarga de informar al esclavo si el maestro espera recibir datos con un '1' o enviarlos con un '0', a este campo de bit también se le denomina como R/W por Receive/Send.

Acknowledge y Not Acknowledge

Las confirmaciones de una comunicación exitosa se dan a través de las señales de Acknowledge (ACK); la característica de estas es que el dispositivo que recibe los 8 bits se encarga de bajar el nivel del bus para indicar al dispositivo transmisor que ha recibido los datos correctamente y que puede continuar con la transmisión.

Un caso especial de estos reconocimientos se da cuando se requiere enviar un Not Acknowledge ($NACK$), es decir: mantener la línea del bus en alto. La utilidad de este caso reside en que cuando hay una comunicación donde uno de los dispositivos está enviando información como transmisor, no hay forma de que el receptor indique que desea detener la comunicación, puesto que si se intenta simplemente dar una condición de Stop, tanto controlador como objetivo tratarían de usar el bus al mismo tiempo. En cambio, si se da un NACK, tenemos que el transmisor sabrá que no puede continuar

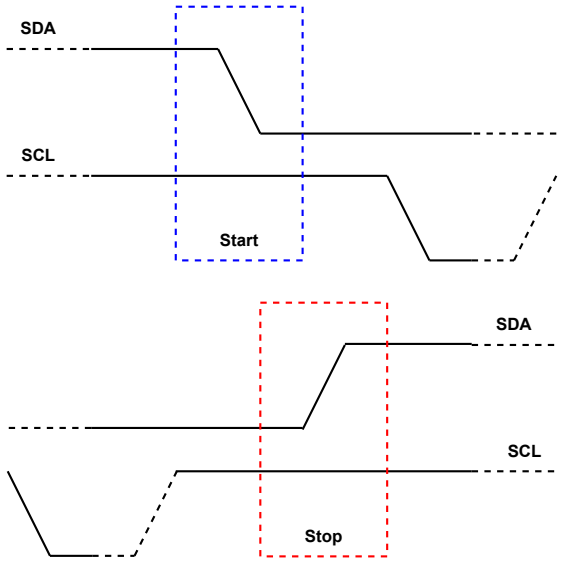


Figura 2.6: Condiciones de Start y Stop.

puesto que ocurrió algo del lado receptor. Una vez que se envió este NACK, ya se puede enviar la condición de Stop. Este caso únicamente es permisible cuando el dispositivo esté en modo de transmisión y quien es receptor necesita detener la comunicación; en cualquier otro caso, indicaría un problema en la comunicación.

Maestro o controlador

El maestro es el módulo capaz de generar una trama compuesta por las condiciones de inicio y paro así como la dirección de esclavo, el bit de Lectura/Escritura (R/W) una sección de acknowledge y una o varias secciones de datos ya sean de recepción o envío, como se ve en la Figura 2.5. Así mismo, es el encargado de generar el reloj de la línea de SCL. Es importante aclarar que, a pesar de ser una arquitectura que permite el modo multi-maestro, no es posible que dos módulos configurados como maestro se comuniquen entre sí, es necesario que uno de ellos sea el controlador y otro el esclavo. Además de que para trabajar con más de un maestro, es necesario implementar un arbitraje.

A continuación, se muestra cómo trabaja el maestro en modo de transmisión y recepción así como sus particularidades.

Transmisión Cuando el maestro es configurado como transmisor, quiere decir que el envío de datos dentro de las tramas será por parte del mismo, mientras que el esclavo, configurado como receptor, se encargará de almacenar estos datos e indicar el ACK como confirmación de una transferencia exitosa y así continuamente hasta una condición de Stop; para esto, las tramas deben ser configuradas como la que se muestra en la Figura 2.7. Como se puede observar, el bit de R/W es configurado como '0', indicando que se realizará una operación de escritura en los registros del dispositivo.

Cabe recalcar, que el modo de transimisión es la única configuración sobre la que trabajan los módulos IIC que trabajan a velocidades de transmisión UFm.

Figura 2.7: Estructura de trama para transmisión.

Recepción Para el caso de un maestro receptor, la Figura 2.8 nos permite visualizar la dirección de la comunicación, dentro de la cual, es relevante observar que el bit R/W con valor de '1' indica que se realizará una operación de lectura de los registros del esclavo. Para detener esta lectura de datos, el maestro debe indicar mediante un NACK que desea detener la recepción de datos, seguido de la condición de Stop.

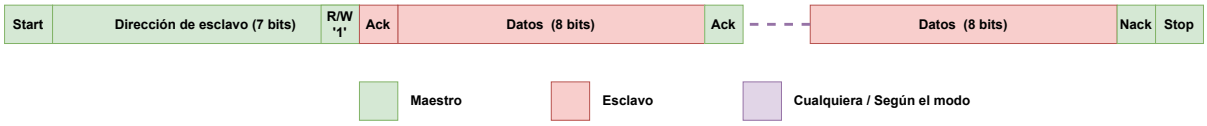


Figura 2.8: Estructura de trama para recepción.

Esclavo u objetivo

El esclavo es el bloque encargado de operar como el objetivo en la comunicación. Este al igual que el maestro tiene la capacidad de envío o recepción de datos, según sea la solicitud que envíe el maestro a través del bit de Escritura/Lectura. Cada uno tiene su propia dirección que normalmente es de 7 bits aunque como se muestra en la Tabla 2.1 es posible que sea de hasta 10 bits.

Como parte de la especificación, el módulo de esclavo únicamente recibe la señal de SCL y no la genera, aunque hay posibilidad de que pueda mantener la línea de reloj en un estado bajo para forzar al maestro a esperar un momento en el que pueda recibir nuevamente los datos.

Transmisión Para su configuración como Esclavo transmisor, la estructura de la trama es la misma que la mostrada en la Figura 2.8, que corresponde a la de un maestro receptor. Por lo que una vez que el maestro desee de dejar de recibir datos, enviará un NACK, que el esclavo deberá reconocer para detener la transmisión y esperar la condición de paro soltando el bus.

Recepción Cuando el esclavo está configurado como receptor, igualmente comparte la trama con el maestro transmisor de la Figura 2.7, por lo que la única tarea del módulo es almacenar los datos que se le envían y en consecuencia enviar los ACK. En este caso no es necesario hacer uso de los NACK puesto que el maestro es quien realiza el control de la línea SCL y por lo tanto de la sincronización; por lo que puede hacer la condición de Stop después del campo de ACK cuando lo requiera sin que se cause una colisión con los datos.

En síntesis, el diagrama de flujos que explica como funciona la comunicación para todos estos modos es la que se muestra en la Figura 2.9.

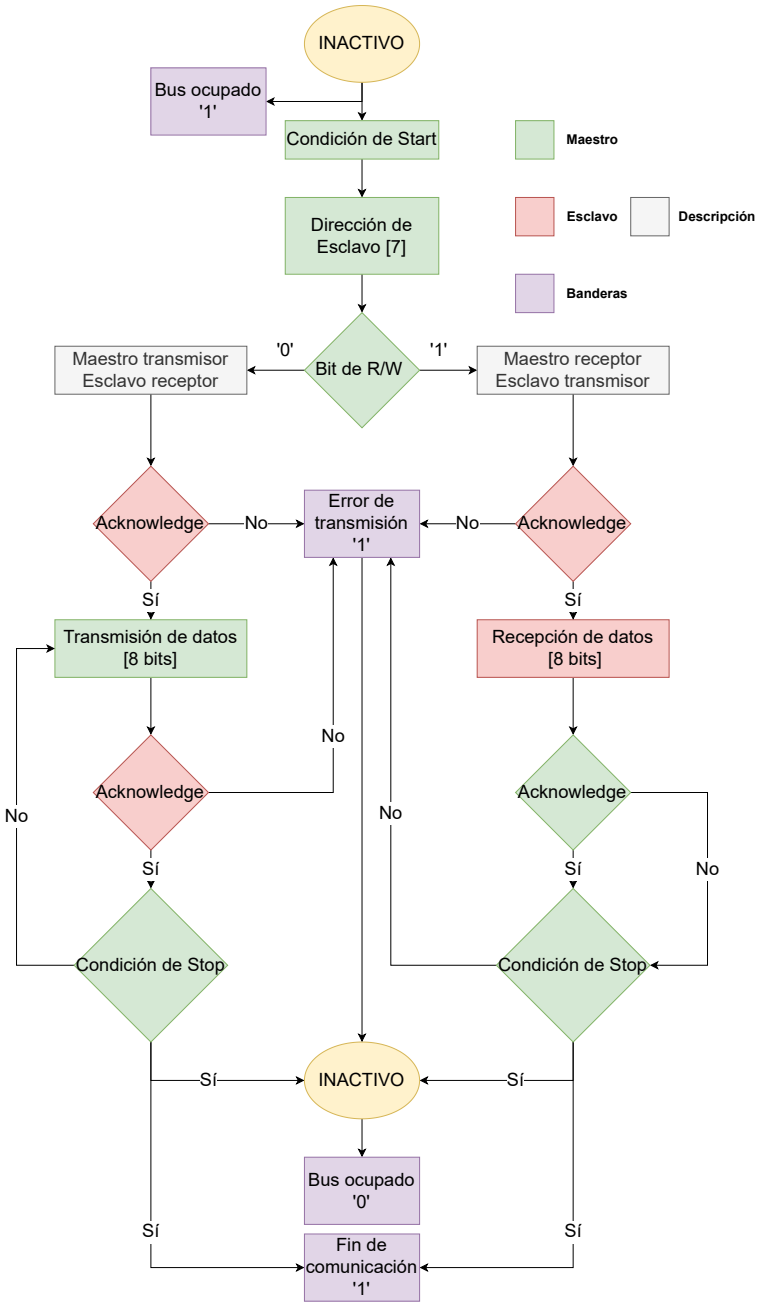


Figura 2.9: Diagrama de flujo que explica el proceso de comunicación.

Caso especial: Start repetido El Start repetido es un caso especial para cuando se busca acceder a un registro específico de un esclavo. Primeramente, se configura el maestro como transmisor, indicando la dirección de esclavo seguido del bit R/W en ’0’ indicando que se escribirá en el esclavo, aunque internamente estaremos indicando la dirección del registro interno. Una vez que se mandó esta dirección interna, se espera el ACK y se genera una nueva condición de Start, esto sería lo que llamamos el Start repetido o *Sr*; después del Sr, reenviamos la dirección de esclavo con el bit R/W en ’1’ indicando la lectura del registro interno que se direccionó previamente, por lo que la información que se envíe como respuesta será parte de este registro; una vez enviado su contenido, el apuntador cambiará al registro interno siguiente.

2.2. Tecnología FPGA

Los Arreglos de Compuertas Programables en Campo o FPGA[3] por sus siglas en inglés, son un una tecnología que permite la descripción de su lógica interna después de su manufactura; es decir, podemos generar un comportamiento en el integrado y modificarlo tantas veces como se necesite y tan rápido como se haga la descripción de la lógica necesaria con la desventaja de que en los FPGA a comparación de los Circuitos Integrados de Aplicación Específica también llamados ASIC[4], los FPGA tienen un área de silicio bastante mayor que puede ser de 1.5 a 10 veces mayor que el encontrado en los ASIC [11].

Históricamente, esta tecnología de reconfiguración por hardware nació con los PLD[5] cuyo módulo básico eran los GALs y PALs. Inicialmente, únicamente podían ser configurados una vez por medio de tecnologías de las tecnologías de fusible y antifusible; posteriormente, estos avanzaron y permitieron su re-configuración a través de memorias EEPROM y EPROM [11]. Así mismo, a través de la interconexión entre estos módulos en un sólo chip, se dio cabida a los CPLD[6]. La diferencia principal entre los CPLD y FPGA, además de que sus bloques básicos tienen arquitecturas distintas, es que en el primero únicamente se puede la interconexión con bloques vecinos, mientras que los FPGA permiten el ruteo con bloques lógicos distantes.

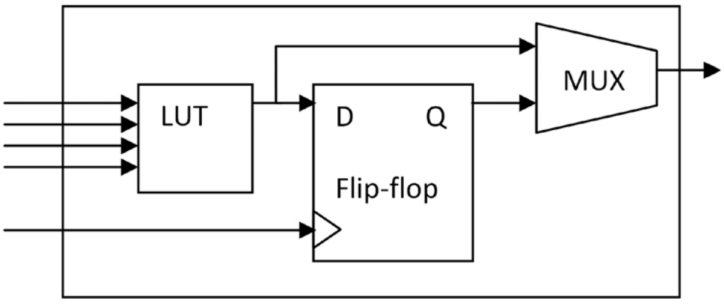


Figura 2.10: Bloque lógico configurable genérico de un FPGA. [2]

Su arquitectura interna está compuesta por Bloques Lógicos Configurables o CLB[7] organizados de forma matricial, estos CLB generalmente están formados por los elementos que se ven en la Figura 2.10.

Es a través de las LUT[8] que se carga el comportamiento descrito en el archivo de bits o *bitstream*. Estas funcionan con una especie de *memoria S-RAM* que almacena dichos bits a los cuales se les denomina máscara de LUT, cuyas salidas van conectadas a multiplexores que a través de sus líneas de selección se encargan de mostrar a la salida el valor deseado según la tabla de verdad descrita. En la Figura 2.11 se puede visualizar una LUT básica internamente.

[3]Field-Programmable Gate Array
[4]Application-Specific Integrated Circuit
[5]Programmable Logic Device
[6]Complex Programmable Logic Device
[7]Configurable Logic Element
[8]Look-Up Table: Tabla de búsqueda

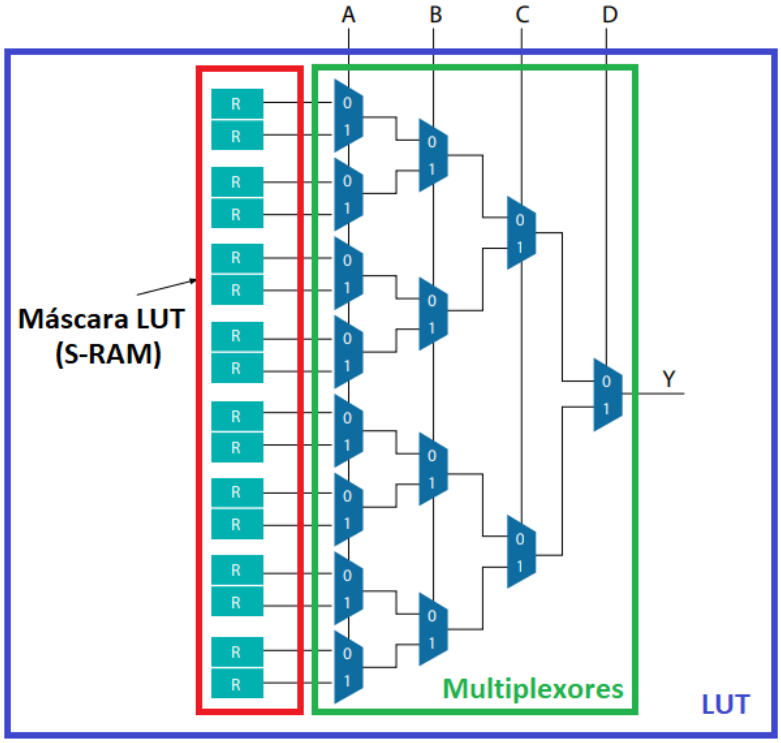


Figura 2.11: LUT de un FPGA a detalle. [3]

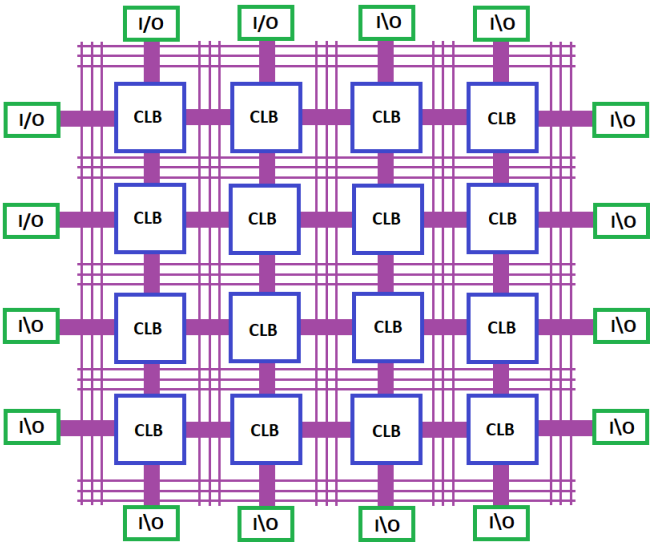


Figura 2.12: Ejemplo de configuración interna de un FPGA. [2]

2.2.1. Tipos de memoria

Como se mencionó previamente, para describir el comportamiento y las conexiones internas de un FPGA, se necesita de un archivo que contenga los bits de datos y configuración, por ello es necesario contar con una memoria que permita almacenar la información. Para esto, los más usuales dentro de los COTS son:

Antifuse

Las Anti-Fuse son un tipo de memoria *PROM*[9]. Estas son ampliamente usadas en la industria aeroespacial debido a su robustez y resistencia a la radiación, aunque con la desventaja de que sólo puede ser programada una vez, por lo que no es útil para hacer pruebas.

Su programación se da a través de la aplicación de una tensión por encima del voltaje umbral en las terminales de las celdas, provocando que la resistencia interna, normalmente con un rango de valores en los giga-ohms, se reduzca considerablemente hasta magnitudes cercanas a las decenas de ohms [12].

Una ventaja de este tipo de memoria es que es no volátil, lo que quiere decir que aún con un corte de energía, la información almacenada se mantiene, ahorrando en los tiempos de carga del programa, aunque esto mismo termina eliminando la capacidad de reconfiguración del FPGA.

Flash

La memoria flash es un tipo de memoria basada en la EPROM. Su principal ventaja reside en que al igual que los Antifuse, esta no pierde los datos cuando no tiene energía y que en este caso sí se permite más de una escritura de datos. Como desventaja se encuentra que para su implementación requiere de más procesos de manufactura debido a la inserción de resistencias de Pull-Up [13], complicando su escalabilidad. Además de que estas resistencias agregan consumo de poder. [14]

S-RAM

Las S-RAM son las más rápidas, densas y ampliamente usadas; son la opción predilecta en FPGAs. A diferencia de las memorias Antifuse y Flash, esta memoria sí borra sus datos si se pierde la alimentación de voltaje, por lo que es necesario volver a cargar el archivo de configuración o *bitstream* cada vez que esto ocurre, generando un retraso entre el encendido del FPGA y su funcionamiento.

Existen distintas configuraciones para estas celdas. La más común es la configuración consistente en seis transistores; esta arquitectura ofrece mayor robustez, menor consumo de potencia y un voltaje de operación pequeño [4]. Una celda de esta configuración con tecnología CMOS[10] se muestra en la Figura 2.13. Aún así, también existen configuraciones con cuatro transistores que ofrece una mayor densidad de memoria; aunque la fabricación se complica pues en procesos a escalas nanométricas hay más variaciones en los parámetros de los transistores. Asimismo, puede ocurrir que en operaciones de escritura se generen cambios en otras celdas que compartan columna con la celda que se busca escribir. [4] Su operación consiste en un circuito biestable similar al funcionamiento de un Latch SR, por lo que mantiene ambos estados internamente; para las operaciones de escritura y lectura se utilizan los transistores Q5 y Q6 que permiten el acceso y modificación de la información en la celda.

[9] Programmable Read-Only Memory: Memoria programable de solo lectura
[10] Complex Metal-Oxide Semiconductor

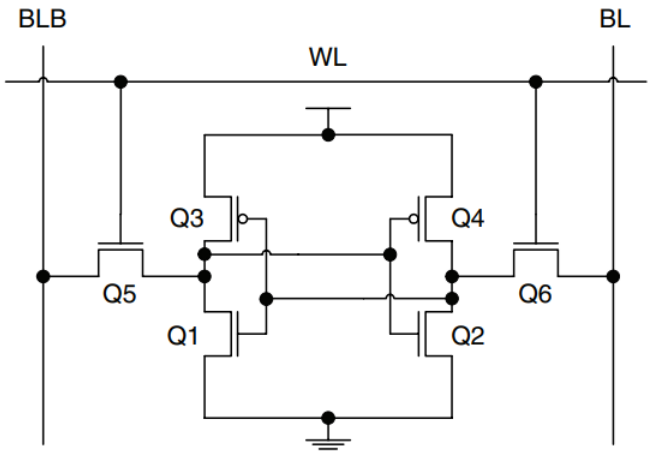


Figura 2.13: Configuración de celda S-RAM. [4]

2.3. Radiación Espacial

En el medio ambiente espacial se encuentran partículas cargadas y radiación electromagnética que puede afectar al funcionamiento de los sistemas electrónicos a bordo de un satélite, por ello es importante conocer sus tipos y los efectos que estos tienen sobre los equipos en servicio.

2.3.1. Tipos de radiación espacial

La mayor parte de la radiación a la que se ven expuestos los cuerpos en el espacio circundante a la Tierra es a la que proviene por la actividad solar o SPE[11]; el otro tipo de radiación notable es la de origen externo al sistema solar, denominada GCRs[12] y finalmente, las radiación contenida en los cinturones de Van Allen.

Radiación Solar - SPE

La principal fuente de radiación que recibe la tierra es la solar y puede ser categorizada en dos principales tipos: lenta y eruptiva. La primera es caracterizada por los vientos solares que se conforman principalmente por un gran flujo de partículas de baja energía y de pocas de alta energía. [5]

El tipo eruptivo está definida por los eventos CME[13] y por erupciones solares. Es durante estos eventos que se liberan grandes cantidades de partículas de alta energía [5]. Es importante hacer notar que, debido a la naturaleza cíclica de 11 años en la actividad solar, durante los periodos de baja intensidad, es menos probable que ocurran estos fenómenos, no así cuando el sol se encuentra en periodos de alta intensidad.

[11]Solar Particle Events: Eventos de partículas solares
[12]Galactic Cosmic Rays: Rayos cósmicos galácticos
[13]Coronal Mass Ejection: Expulsión de masa coronaria

Rayos Cósmicos Galácticos - GCRs

Como se mencionó anteriormente, las partículas provenientes de este tipo de radiación son de fuentes externas al sistema solar. Su principal característica es que el flujo, a pesar de ser pequeño, su contenido en partículas de alta energía es grande. Está principalmente conformado por protones de alta energía, partículas alfa, electrones y componentes con número atómico grande. [15]

Cinturones de Van Allen

Debido a la influencia de los campos magnéticos terrestres, se forma una zona alrededor de la tierra que está principalmente compuesta por protones y electrones formando una especie de cinturón. Esta zona se divide en dos, la parte interna y externa; la primera está principalmente compuesta por protones de altas energías y electrones de medias energías. Mientras que la sección externa contiene principalmente electrones con altas energías [16].

2.3.2. Efectos de la radiación

Dosis Total Ionizante - TID

Es un efecto acumulativo que se da en los materiales, resultado de la exposición por mucho tiempo a partículas cargadas, como protones y electrones que se depositan en los componentes electrónicos. Como resultado, puede modificar las magnitudes en los voltajes de disparo, aumentar la corriente de fuga del transistor y causar tiempos de skew [17].

Se cataloga como un efecto degradante en los dispositivos que a largo plazo puede influir en una falla funcional. [18]

Daño por desplazamiento

El daño por desplazamiento es un efecto provocado por el choque de partículas de alta energía que alteran la red cristalina de los semiconductores, haciendo que los componentes del cristal adquieran energía mecánica que provoca su desplazamiento generando un defecto en la composición de la red [5]. Este tiene efectos de degradación similares a los que se ven en los TID, por lo que su efecto se ve reflejado en la variación de las especificaciones intrínsecas por fabricación en los componentes.

Single Event Effects - SEE

Un SEE es un proceso estocástico que está definido por la colisión de una partícula de alta energía que atraviesa a los materiales semiconductores. De esta interacción, la partícula pierde energía a través de la dispersión de Rutherford. Asimismo, al interactuar con los núcleos de la red cristalina del semiconductor deja pares de electrones y hoyos formados predominantemente por las interacciones de Compton [19]; estos pares al ser cargas libres se depositan en los electrodos del material, causando excesos de carga transitorios que pueden modificar los valores de operación.

Estos eventos se han vuelto especialmente relevantes a medida que la miniaturización de los circuitos electrónicos ha hecho que las cargas involucradas en su operación sean cada vez menores; por lo que al darse eventos como este, se produce un efecto más notable en la operación del sistema, generando desde glitches hasta fallas catastróficas [19]. Asimismo, se trata de un evento crítico para los dispositivos que no pueden disponer de un reseteo o de un cambio en su configuración mientras se encuentran operando, tales como equipos de autopilotaje, marcapasos y la electrónica empleada para misiones espaciales. [20]

Single Event Upset - SEU Este tipo de evento se da cuando la colisión afecta a la carga contenida en los elementos que almacenan memoria, provocando un cambio en el estado lógico interno, es decir, altera la información. Se trata de un error catalogado dentro de los tipo *Soft* o *Transitorio*, ya que se trata de un cambio temporal en la información.

Multiple Bit Upset - MBU Al igual que los SEU, este evento se ve definido por el cambio en la información contenida por un elemento con memoria; la diferencia radica en que con una sola colisión se provoca el cambio en más de un bit, como su nombre lo indica.

Single Event Functional Interrupt - SEFI Es un tipo de SEU que se da cuando el cambio de nivel lógico ocurre en la unidad de configuración o control, por lo que el módulo genera un fallo o tiene un comportamiento erróneo. Se considera que es una falla del tipo persistente, ya que continuará hasta que el dispositivo sea reconfigurado.

Para el caso de específico en los FPGAs, es más dirigido a una falla global del dispositivo, como el generado por un reseteo, que eliminaría el *bitstream* de la memoria S-RAM. Igualmente si se da un evento en la interfaz de configuración como el JTAG.

Single Event Transient - SET Se trata específicamente del fenómeno transitorio que experimentan los dispositivos analógicos -como sensores- debido a la colisión. Este cambio afecta a los valores de operación que modifican la señal analógica que está siendo registrada. Este fenómeno pertenece igualmente a los fallos del tipo *soft*.

Single Event Latch-Up - SEL Es un tipo de cortocircuito provocado por la presencia de un circuito SCR presente de manera parasitaria en los CMOS, por lo que este efecto es propio de esta familia.

Single Event Burnout - SEB Cuando la colisión se localiza en una juntura PN es posible que el veloz reacomodo de las cargas produzca una diferencia de potencial en inversa, lo suficientemente grande como para alcanzar el voltaje de ruptura en la juntura, que provoque una corriente de cortocircuito que queme el dispositivo.

Single Event Gate Rupture - SEGR Cuando la partícula atraviesa la capa de óxido del capacitor en la terminal de compuerta de un transistor tipo MOS se genera una corriente transitoria, que si es lo suficientemente grande puede perforar el dieléctrico, provocando un paso directo a la corriente entre la terminal de compuerta y el sustrato, inhabilitando el dispositivo.

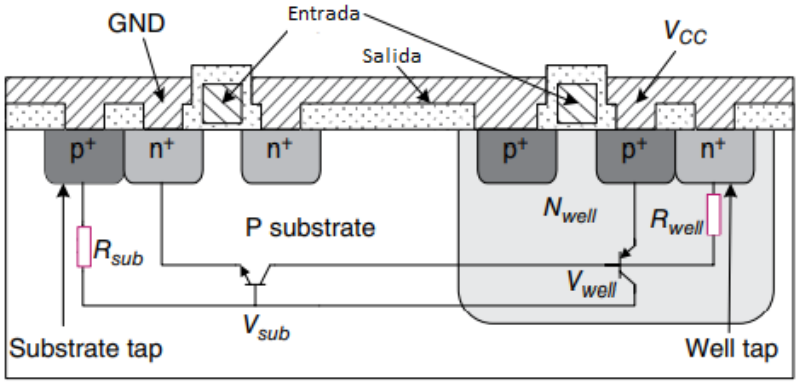


Figura 2.14: Estructura de un CMOS para una compuerta NOT. [5]

Para los últimos tres eventos *SEL,SEB y SEGR* se les considera dentro de los errores de tipo *Hard* o *permanentes*. Aunque la probabilidad de ocurrencia para estos es reducida en FPGAs.[5]

2.4. Mecanismos de tolerancia a fallas para hardware

Todos los sistemas cuentan con un factor de confiabilidad para su operación, basada a su vez en los parámetros de: fiabilidad, o la capacidad del sistema de operar correctamente en un intervalo de tiempo; disponibilidad, que es la métrica que indica la probabilidad de que el sistema para ejecutar su función en un momento dado; seguridad, es la característica que asegura que el sistema no va a comprometer o interrumpir el funcionamiento de otros módulos; rendimiento, es la medida de la probabilidad de que un subconjunto de funciones operen correctamente después de un fallo, es similar al concepto de confiabilidad con la diferencia de que este último asegura el correcto funcionamiento de *todas* las funciones; mantenibilidad, es la facilidad con la que se puede reparar el sistema una vez que ha fallado y finalmente, comprobabilidad que indica la habilidad de probar ciertos atributos de un sistema.

Cuando se inicia el desarrollo de un sistema se debe asegurar una operación confiable para las condiciones a las que será expuesto, por lo que existen técnicas que permiten elevar este parámetro de confiabilidad en el sistema.

Para el caso del Hardware contamos principalmente con la redundancia modular, por lo que principal enfoque de esta sección es mostrar las técnicas que se emplean para aumentar los parámetros de confiabilidad en el hardware de los módulos que se ven susceptibles a fallas. Para ello, se muestran técnicas de redundancia pasiva, activa e híbridas.

2.4.1. Técnicas de redundancia pasiva

La redundancia pasiva se refiere a la contención de fallas mediante su enmascaramiento; es decir, no se detecta la falla, únicamente se contiene y se filtra la respuesta comparándola con la de otros módulos que trabajan en paralelo a través de un comparador también llamado *Votador mayoritario*. Este tipo de redundancia permite contener las

fallas sin la intervención de un operador externo o del monitoreo constante del sistema.

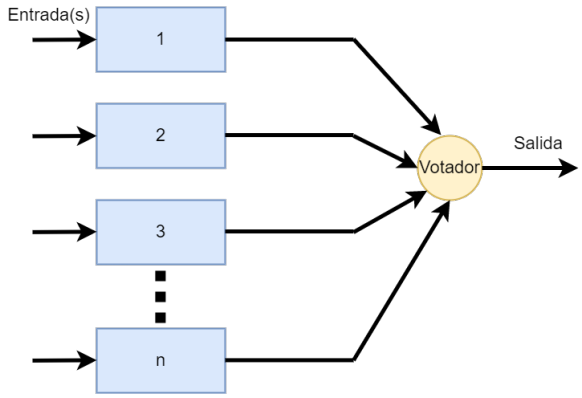


Figura 2.15: Estructura genérica para redundancia de n módulos.

Votador mayoritario

El votador mayoritario es, como su nombre indica, el bloque encargado de mostrar a la salida la señal que tienen en común la mayoría de módulos. El módulo mostrado en la Figura 2.15 es el ejemplo para un votante ideal, es decir, que tiene una fiabilidad de 1 y que por lo tanto no falla. Esto en la vida real no es así, por lo que en caso de ocurrir un fallo, se convierte en un nodo único de falla. Para evitar esto, también se puede aplicar la redundancia entre votadores mayoritarios. Como se ve en la Figura 2.16.

Como se puede intuir, para que este esquema funcione es necesario que la cantidad de módulos redundantes sea un número impar, con la finalidad de que siempre exista una mayoría.

Redundancia Modular

El esquema de redundancia pasiva más básico es el ofrecido por la redundancia modular triple o *TMR*[14], que permite detectar hasta un bloque con falla. Esta arquitectura consiste en tres módulos que realizan la misma tarea de forma paralela, cuyos resultados independientes se comparan a través de un votador mayoritario, visto en la Sección 2.4.1. La ventaja de este tipo de configuración es que es capaz de filtrar hasta un módulo con falla, aunque en el caso de haber más de un módulo afectado, es posible que se filtre aquella respuesta que compartan ambos bloques defectuosos.

Para enmascarar más de un error, se escala la composición a N módulos en paralelo, como en el ejemplo de la Figura 2.15. Conocida como la arquitectura de un NMR[15], que es la generalización de la redundancia pasiva de hardware. Esta estructura es capaz de funcionar correctamente hasta con $\frac{N-1}{2}$ módulos defectuosos [7].

[14]Triple Modular Redundancy
[15]N-Modular Redundancy

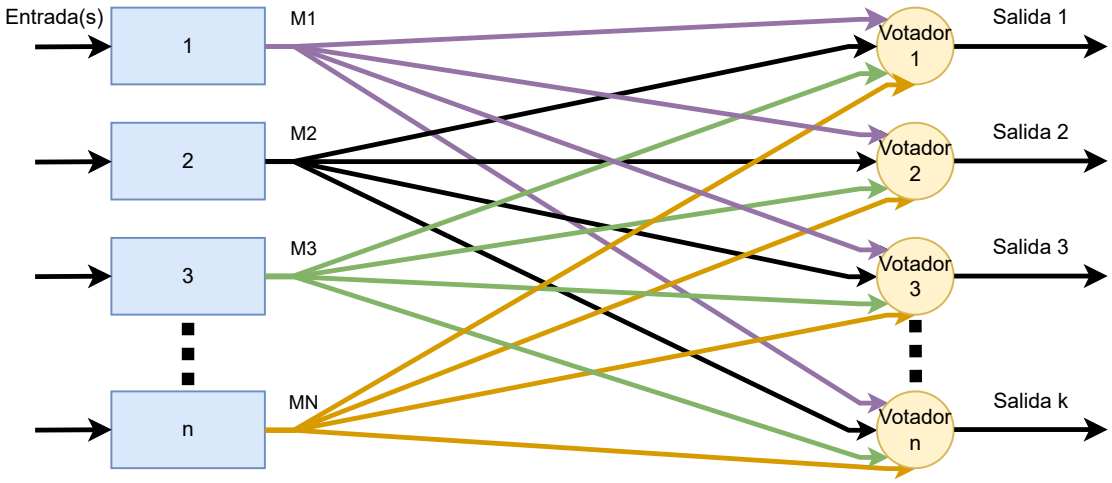


Figura 2.16: Estructura genérica para redundancia de n módulos con votadores mayoritarios no ideales.

2.4.2. Técnicas de redundancia activa

A diferencia de la redundancia pasiva, la activa se encarga de detectar la falla, localizar el punto de falla y consecuentemente tomar acción sobre ella para devolver el sistema a la operación normal. También se le denomina método *dinámico*. Aunque este método no es capaz por sí mismo de enmascarar el error, por lo que si ocurre alguno puede modificar la operación del sistema. La fortaleza de este mecanismo reside en la capacidad de provocar que el sistema recupere su funcionamiento normal después de darse la falla, por lo que los sistemas con este esquema deben ser capaces de tolerar errores temporales con la certeza de que se recuperará la funcionalidad general en un periodo de tiempo razonable.

Duplicación con comparación

Este esquema, mostrado en la Figura 2.17, consiste en un par de módulos idénticos trabajando en paralelo, por lo que únicamente tiene la capacidad de detectar la falla, mas no puede asegurar la continuidad de la operación, pues al no tener la posibilidad de reconocer el módulo defectuoso únicamente contiene la propagación de la falla; esto resulta conveniente para los casos en los que el evento no genera una falla permanente, por lo que aún podría continuar su operación. Para su funcionamiento requiere una etapa de comparación que marcará un error en el caso de no coincidir ambas respuestas.

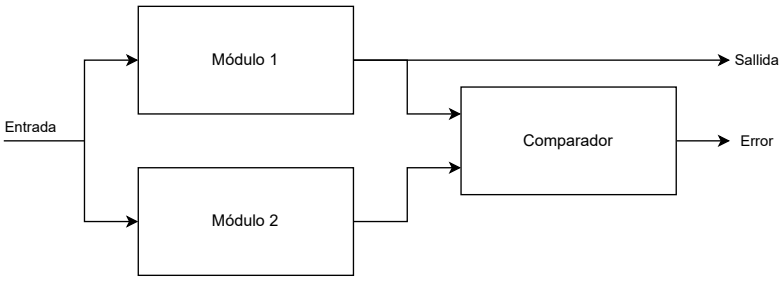


Figura 2.17: Arquitectura de una Duplicación con comparación.

Repuesto en espera

Esta configuración se caracteriza por tener un módulo operacional y $n-1$ repuestos, que en conjunto tienen la capacidad de detectar el error y localizar el módulo con falla, con la intención de sacar de operación dicho módulo defectuoso y reemplazarlo con uno que esté operando correctamente.

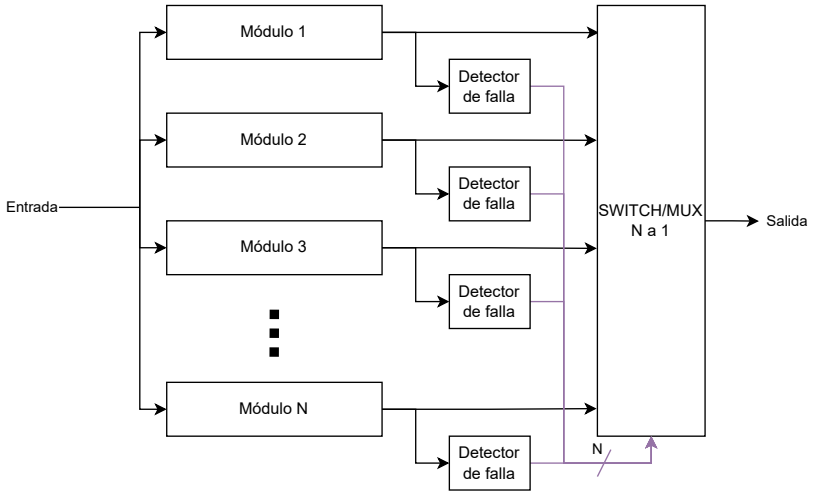


Figura 2.18: Arquitectura de un Repuesto en espera.

Dentro de este esquema hay dos vertientes principales: Reemplazo caliente, que consiste en que todos los repuestos están trabajando simultáneamente, pero sólo uno muestra su salida; y el reemplazo en frío, en el que los módulos de repuesto están desenergizados, lo que provoca que se necesite inicializar y poner a punto el módulo que reemplazará al defectuoso, aumentando el tiempo que requiere poner al sistema nuevamente en operación, aunque como ventaja se encuentra que los módulos de reemplazo no estarán consumiendo potencia mientras no estén siendo utilizados. [7]

Técnica de Par y Repuesto

Como se muestra en la Figura 2.19, esta técnica se caracteriza por emplear las características de las dos arquitecturas anteriores. Para ello, siempre hay dos módulos trabajando en paralelo y cuando en uno de ellos es detectada una falla, el sistema compara con los demás módulos de repuesto para conocer cuál de los dos bloques que están operando es el que se encuentra fallando y así cambiarlo por un reemplazo.

2.4.3. Técnicas de redundancia híbrida

Este tipo de técnicas se encargan de compartir las características de contención de fallas de la redundancia pasiva junto con la habilidad de detección de fallas, su localización y posterior recuperación del sistema propias de la redundancia activa. Es especialmente empleado en entornos donde la seguridad es crítica, como en reactores nucleares, equipo médico, etc [7]. Aunque su implementación resulta mucho más costosa en términos de hardware empleado. [6]

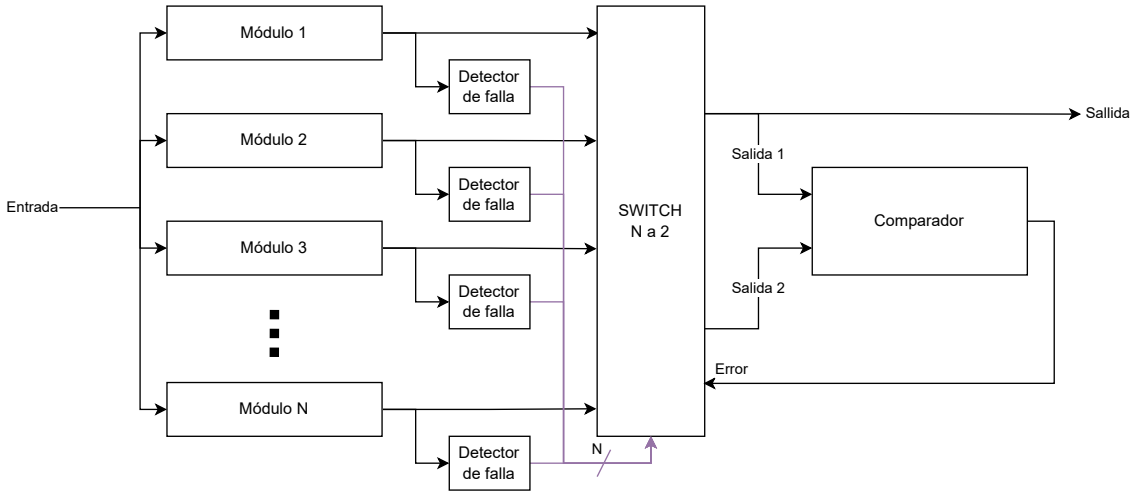


Figura 2.19: Arquitectura de un Par y Repuesto.

Redundancia de N Módulos con Repuesto

Cuando nos encontramos con técnicas de redundancia híbrida, nos encontramos con que la mayoría se basan en arquitecturas NMR [6]. Este mecanismo está constituido por un *núcleo* de N módulos, que son los que participan activamente en la etapa de votación. En caso de detectarse falla en alguno de ellos, este se reemplaza en el núcleo por un repuesto. Una ventaja de este esquema es que la confiabilidad del sistema se mantiene mientras aún existan repuestos que puedan sustituir a los módulos con fallas. [6].

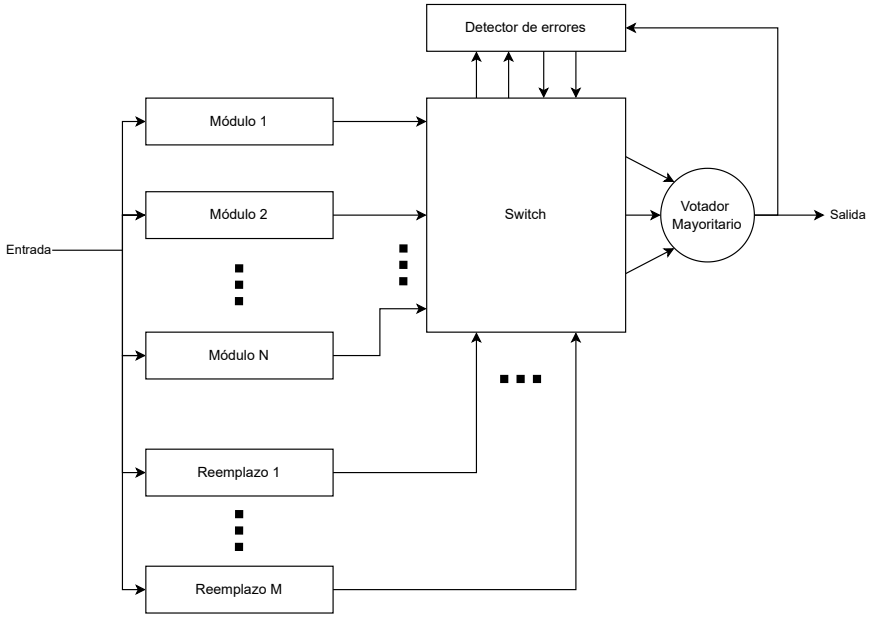


Figura 2.20: Arquitectura de una Redundancia de N Módulos con Reemplazos. [6]

Redundancia de Auto-Purgado

Consiste en una arquitectura en la que todos sus módulos trabajan en paralelo y pasan por una etapa de votación mayoritaria, con la cual se selecciona lo que se mostrará a la

salida. Esta misma salida se realimenta al sistema por medio de unos bloques de *switcheo* con los que, en caso de diferir a la respuesta mayoritaria, el módulo defectuoso se excluye a sí mismo del proceso de votación.

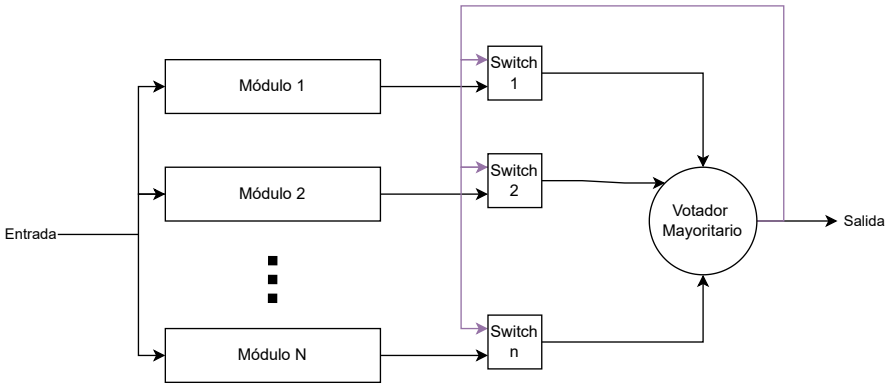


Figura 2.21: Arquitectura de una redundancia Auto-Purgada. [7]

Capítulo 3

Estado del Arte

A través de este capítulo, se presentan distintas arquitecturas de módulos I2C, como los que vienen embebidos en microprocesadores o los que se muestran como parte de*IPs*.

3.1. Periférico I2C de TM4C1294NCPDT

La TM4C1294NCPDT de Texas Instruments incluye un periférico de I2C, cuyo diagrama de bloques es mostrado en la Figura 3.1

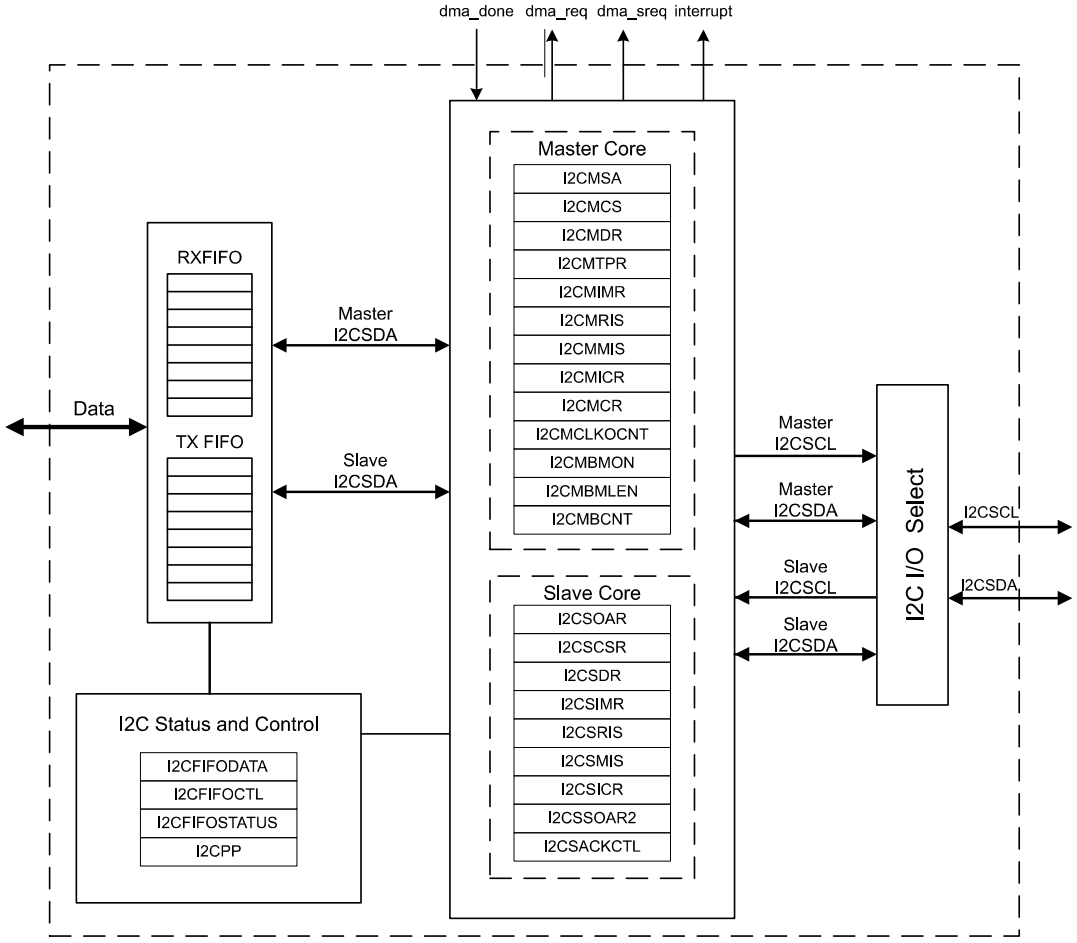


Figura 3.1: Diagrama de bloques del I2C de un TM4C1294NCPDT. [8]

Este diseño muestra arquitecturas independientes para el manejo del maestro y del esclavo, así como dos unidades de almacenamiento temporal tipo FIFO, una para los datos de recepción y otra para los de transmisión.

Sus principales características son:

- Cuatro modos de operación
  - Maestro transmisión
  - Maestro recepción
  - Esclavo transmisión
  - Esclavo recepción

- Velocidad de transmisión
  - 100 [kbits/s]
  - 400 [kbits/s]
  - 1 [Mbits/s]
  - 3.4 [Mbits/s]
- Memorias FIFO de transmisión y de recepción de 8 niveles
- Módulo de Acceso Directo a Memoria
- Supresión de glitches
- Soporte multimaestro
- Sincronización de reloj
- No responde a las direcciones de 10 bits

3.2. Periférico I2C del STM32l45

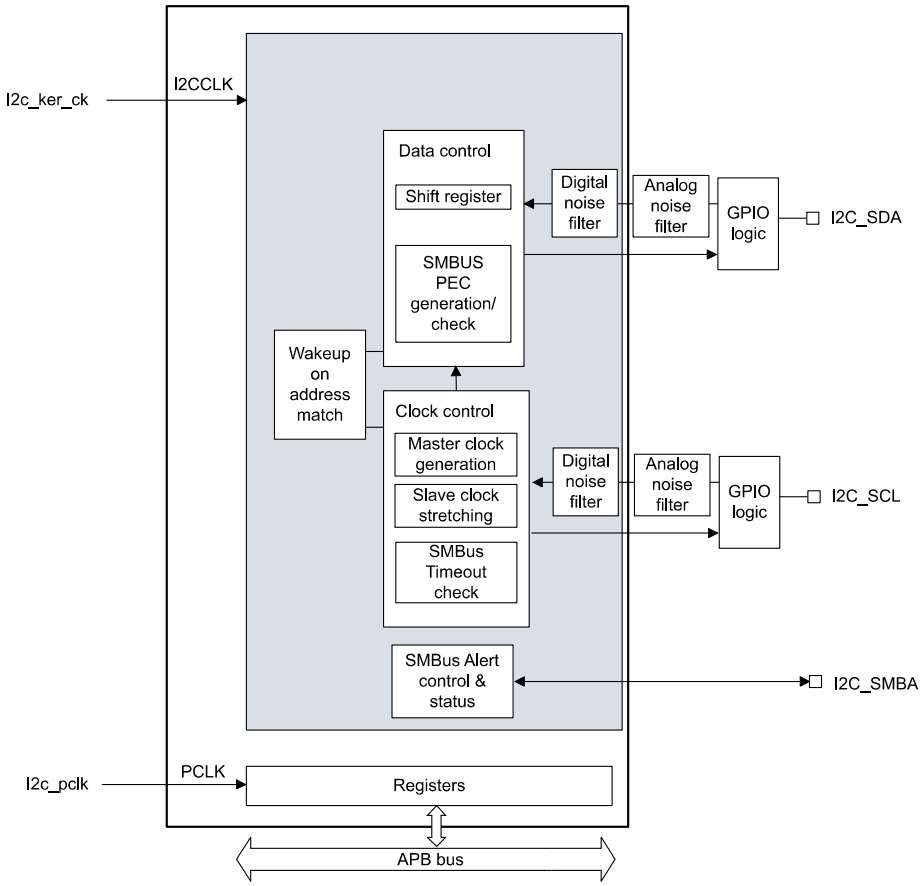


Figura 3.2: Diagrama de bloques del I2C de un STM32l45. [9]

El diagrama de bloques difiere bastante de lo visto en el caso del microprocesador de Texas Instruments, ya que este no tiene núcleos dedicados para el modo maestro y el esclavo. Por lo que centra su funcionamiento en un núcleo para el control de la línea de datos y otro para la línea de reloj. Además, cuenta con registros de datos independientes para la recepción de datos y otro para su transmisión. La principal diferencia con el módulo de Texas Instruments reside en que este no cuenta con una FIFO. En su lugar directamente se tiene que configurar una DMA. Su funcionamiento base es como esclavo, por lo que sólo actúa como maestro cuando se le indica que haga una condición de Start y pierde el modo maestro una vez que se marca la condición de stop o que pierde el arbitraje.

En el manual de referencia proporcionado por STM, indica que está diseñado con las pautas de la Revisión 3.0 del protocolo I2C. En el manual del microprocesador marca que sus principales características consisten en:

- Cuatro modos de operación
  - Maestro transmisión
  - Maestro recepción
  - Esclavo transmisión
  - Esclavo recepción
- Velocidad de transmisión
  - 100 [kbits/s]
  - 400 [kbits/s]
  - 1 [Mbits/s]
- Módulo de Acceso Directo a Memoria
- Supresión de glitches
- Soporte multimaestro
- Sincronización de reloj
- Funciona con direcciones de 10 bits

3.3. IP de I2C de la NAND LOGIC CORPORATION

Ya dentro del campo de FPGAs se pueden encontrar lo módulos de propiedad intelectual o *IP* por sus siglas en inglés. Estos son diseños que ofrecen ciertos fabricantes, proporcionando diseños ya probados que permiten ser usados en forma de simples bloques o cajas negras. Entre ellos, se pueden encontrar interfaces seriales como los que ofrece la compañía *NAND LOGIC*. En su página no se detalla una arquitectura de bloques definida, pero entre sus características se encuentra:

- Dos modos de operación

  - Maestro transmisión
  - Maestro recepción
- Velocidad de transmisión
  - No detallada
- Bit de Acknowledge
- FIFO de tamaño configurable
- Tiempo de espera configurable
- Funciona con direcciones de hasta 7 bits

A través de la página `https://www.xilinx.com/products/intellectual-property/1-1ncih7t.html#productspecs` se puede ver la tabla de utilización de recursos.

Family	Device	Speed Grade	Tool Version	HW Validated?	Slice	LUT	BRAM	DSP48	CMT	GT_x	FMAX (Mhz)
ARTIX-7 Family	XC7A175T	-3	Vivado 2019.2		313	271	0	0	0	0	200

Figura 3.3: Reporte de utilización del módulo de NAND LOGIC

3.4. Controlador I2C-SMBUS de CAST

Otra módulo de propiedad intelectual es el proporcionado por *CAST*. El diagrama de bloques es similar a lo que se puede encontrar en el periférico de la STM32l45, ya que ninguno cuenta con núcleos dedicados para el control como maestro o esclavo, si no que se enfoca en un bloque para control del reloj y otro para el de datos.

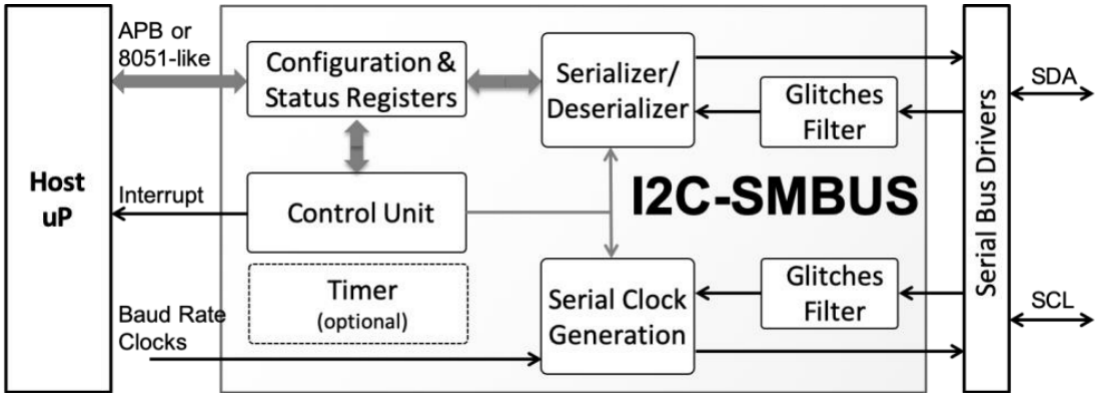


Figura 3.4: Diagrama de bloques de la IP I2C-SMBUS de CAST. [10]

- Cuatro modos de operación

  - Maestro transmisión
  - Maestro recepción
  - Esclavo transmisión
  - Esclavo recepción
- Velocidad de transmisión
  - No se indica
- Cuenta con arbitraje
- Generación de SCL y sincronización de datos
- Clock Stretching
- Filtro de glitches configurable
- Funciona con direcciones 7 bits

Family	Device	LUTs	BRAM	Clock Freq.
Spartan-7	xc7s75-1IL	589	0	100 MHz
Artix-7	xc7a12t-1	588	0	100 MHz
Kintex UltraScale	xcku025-2	566	0	100 MHz
Kintex UltraScale+	xcku3p-3	566	0	100 MHz

Figura 3.5: Reporte de utilización del módulo I2C-SMBUS de CAST.

3.5. Controlador I2C-Master/Slave de CAST

Se muestra igualmente otra *IP* de CAST, esta muestra un mayor enfoque en cumplir con el estándar de *I2C* en el aspecto de que no busca usar *SMBus* o similares, además de que cuenta con la posibilidad de operar con todas las velocidades que maneja el protocolo *I2C* hasta su Revisión 7.0, abordada en el Capítulo **??** del Marco Teórico.

- Cuatro modos de operación
  - Maestro transmisión
  - Maestro recepción
  - Esclavo transmisión
  - Esclavo recepción
- Velocidad de transmisión
  - 100 [kbit/s]
  - 400 [kbit/s]

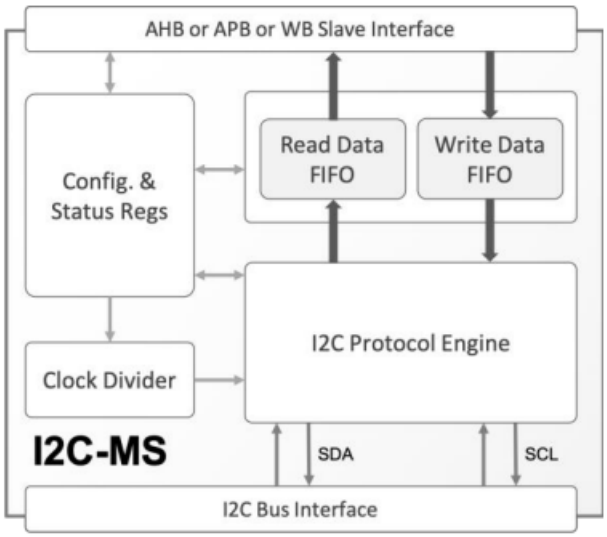


Figura 3.6: Diagrama de bloques de la IP I2C-Master/Slave de CAST. [10]

  - 1 [Mbit/s]
  - 3.4 [Mbit/s]
  - 5 [Mbit/s]
- Cuenta con arbitraje
- Clock Stretching
- Soporte de arquitectura multimaestro
- FIFO de lectura y escritura
- Número de transferencias configurable
- Funciona con direcciones 7 bits

Configuration	FPGA Family / Device	Logic (LUTs)	Memory (BRAM)	Clock Freq. (MHz)
Slave only, 2-bytes FIFOs	Kintex UltraScale / xku085-1	251	-	500
Master only, 2-bytes FIFOs	Kintex UltraScale / xku085-1	345	-	600
Master-Slave, 2-bytes FIFOs	Kintex UltraScale / xku085-1	395	-	600

Figura 3.7: Reporte de utilización del módulo I2C-Master/Slave de CAST.

Capítulo 4

Diseño

4.1. Diseño del concepto

Con lo visto en el Capítulo **??**, en la Sección 2.1 referente a la última revisión del estándar I2C y acotándolo con lo visto en el Capítulo 3, donde se analizaron distintos módulos que cumplen con dicha especificación se definen los requerimientos de un sistema para comunicarse completamente por medio del protocolo I2C.

4.1.1. Requerimientos y especificaciones

Velocidades

La especificación de I2C, abordada en la Sección 2.1 del Capítulo **??**, en su última revisión del 2021, define hasta 5 velocidades, de las cuales las más recientes y rápidas requieren de una configuración de trama inicial diferente, y aunque hay retrocompatibilidad para el modo *Hs* (Revisar Tabla 2.2) con módulos que operan a velocidades menores, este se pierde para el modo *UFm*, en el que sus líneas son unidireccionales. Debido a esto se entiende que los modos *esenciales* o velocidades *F/S* funcionan en la mayoría de módulos sin necesidad de cambiar características de las líneas o las tramas. Por lo que este trabajo contempla únicamente estas velocidades.

Almacenamiento de datos

El diseño debe contemplar una forma de almacenar temporalmente la información recibida o que está por enviarse. Esto debido a que en la Sección 1.3 del Capítulo 1 se indica que el diseño está contemplado para trabajar en conjunto con un microprocesador, por lo que con este sistema de almacenamiento temporal ya no se necesitará hacer un monitoreo constante del módulo. Asimismo, se permite que la operación sea de forma fluida, puesto que no necesita de esperar datos para seguir operando, logrando menores tiempos muertos.

Configuración

Debido a que existen diversos modos de operación y herramientas como la FIFO, se vuelve necesario dar al usuario el poder de activar, desactivar funcionalidades o de establecer el modo de trabajo, por lo que en el diseño se han de considerar líneas de control que permitan configurar y establecer el funcionamiento requerido

Banderas

Es necesario que el sistema proporcione información de la actividad que se encuentra realizando, así como reportar si existe un error debido a una mala configuración, para ello se emplean banderas que se encargan de proporcionar detalles de los procesos que está realizando el sistema.

Consideración de diseño

Como consideración de diseño, se requiere que los submódulos tengan bien identificado el comportamiento de sus estados, de manera que, en caso de un SEE como un SEU se pueda identificar que se encuentra dentro de un estado *no permitido* y regrese al estado inactivo o inicial, con el fin de no generar errores a los demás procesos del sistema, asimismo, deben indicar la falla levantando una bandera de error.

4.1.2. Propuesta

En la Figura 4.1 se propone un módulo básico que muestra las entradas y salidas necesarias acotadas a lo visto en el apartado 4.1.1

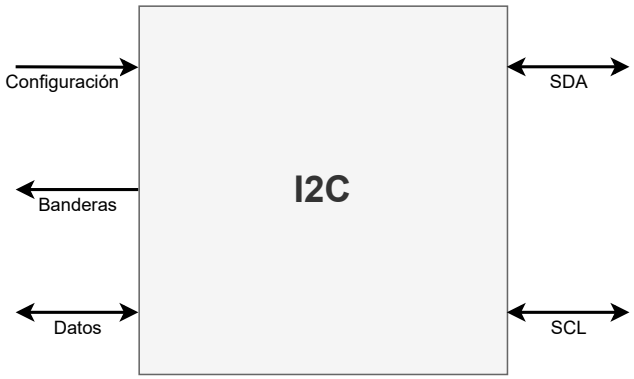


Figura 4.1: Concepto básico del módulo.

4.2. Diseño del Módulo

Para lograr obtener un módulo con las características mostradas en la Figura 4.1 se propone un sistema con los siguientes submódulos, que a su vez, está inspirado en la arquitectura de bloques que muestra el TM4C1294 de Texas Instruments, revisado en el Capítulo 3.

Primeramente, en la Sección de Generales se muestra el principio de diseño de los componentes fundamentales que se utilizarán posteriormente en los subsistemas. Después, se muestra el propósito y diseño de cada uno de los subsistemas que componen al módulo.

4.3. Generales

4.3.1. Flip-Flop D

Los Flip Flop tipo Data son un elemento de memoria y de entre otros tipos de Flip Flops, resultan ser los más económicos y eficientes, además de que permiten generar el comportamiento de otros Flip Flops a través lógica externa [21]. Su funcionamiento consiste en que al momento de la llegada de una señal de reloj, el Flip Flop retiene el

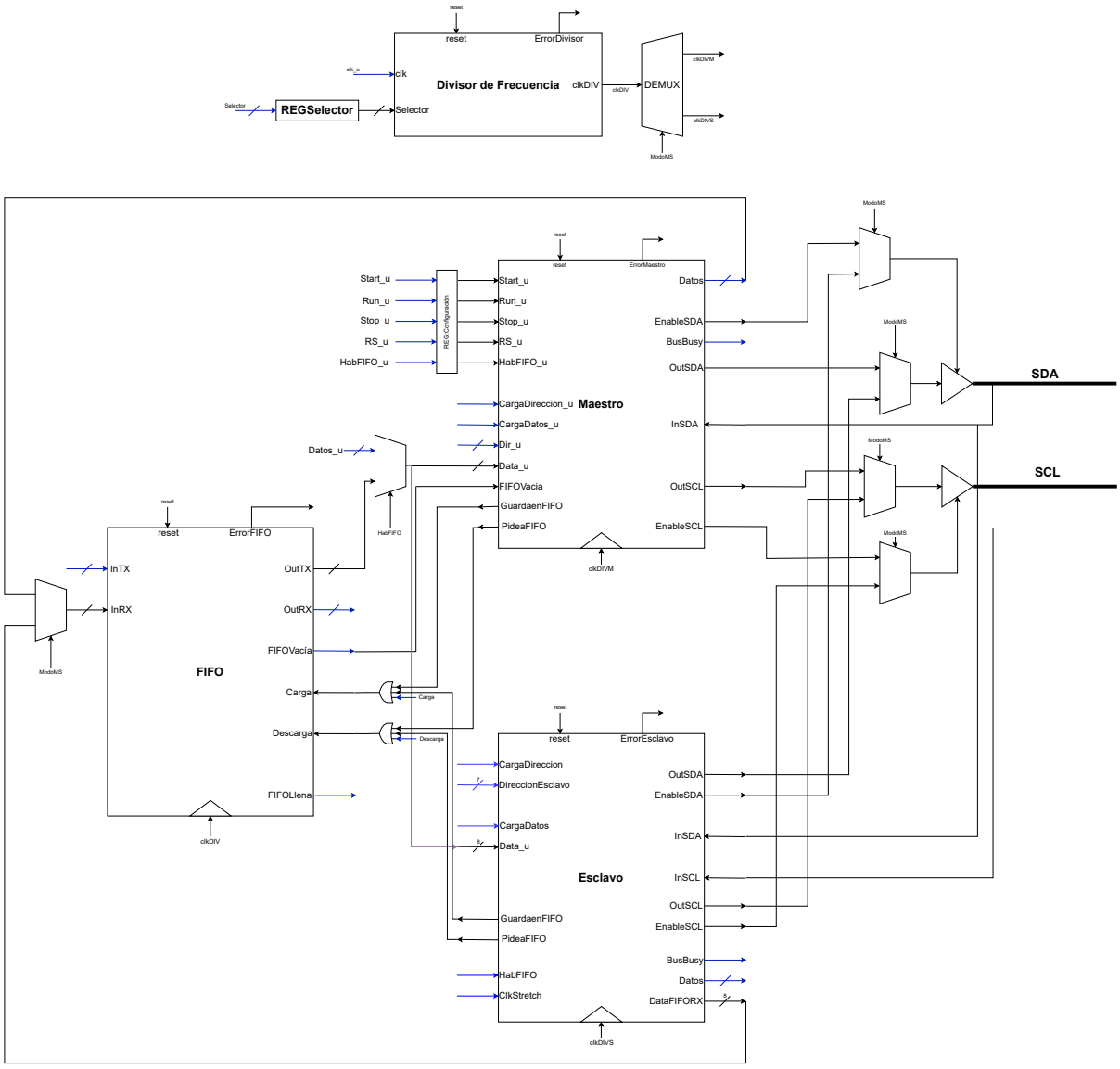


Figura 4.2: Módulo I2C completo a nivel de sistema.

valor que se presenta a la entrada en ese instante y el valor permanece hasta la llegada de un nuevo pulso de reloj.

Con esto de base, para el diseño de un Flip Flop D, de ahora en adelante simplemente FFD, se tomó la libertad de hacer una descripción en VHDL por método algorítmico, considerando la Tabla 4.1 ya que este se constituye como la mínima unidad de memoria que esteramos manejando durante el transcurso de este trabajo y requerimos que el sintetizador lo considere como tal para que lo implemente correctamente dentro del Bloque Lógico Programable. Asimismo, necesitamos que reconozca el comando de reset, por lo que igual se describirá el comportamiento necesario para ello.

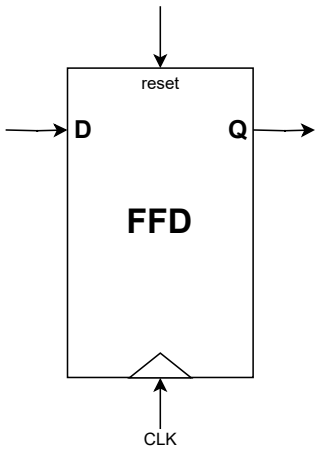


Figura 4.3: Flip Flop tipo D a alto nivel de abstracción.

D	$Q(t+1)$
0	0
1	1

Tabla 4.1: Tabla característica del FFD.

```
library ieee;
use ieee.std_logic_1164.all;

entity flip_flopD is
port(clk, reset, d: in std_logic;
q: out std_logic
);
end entity;

architecture behavioral of flip_flopD is
begin
process(clk, reset)
begin
if (reset='1') then
        q<='0';
elsif(rising_edge(clk)) then
        q<=d;
end if;
end process;

end behavioral;
```

Figura 4.4: Código en VHDL del FFD descrito de forma algorítmica.

4.3.2. FFD Keep

Un Flip-Flop D Keep es un Flip Flop tipo D que tiene la propiedad de mantener su estado en alto por más de un ciclo, similar a un Flip Flop RS el cual al activar su línea de Set se cambia el estado a '1' y se mantiene hasta la señal de Reset. Se implementó de esta manera ya que dentro de la mayoría de FPGAs se encuentra un FFD y en lugar de simular completamente un FFRS por un método algorítmico se prefiere utilizar esta implementación *dedicada* que permite optimizar el diseño de nuestro Flip Flop.

Para esto, se definió la Tabla 4.2 y mediante la resolución por mapas de Karnaugh, se obtiene que el circuito que permite este comportamiento es una compuerta *or* con señales de entrada provenientes de la salida del FFD y de la propia entrada del sistema, como se muestra en la Figura 4.5

Reset	$Q(t)$	D	$Q(t+1)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Tabla 4.2: Tabla característica del FFD Keep.

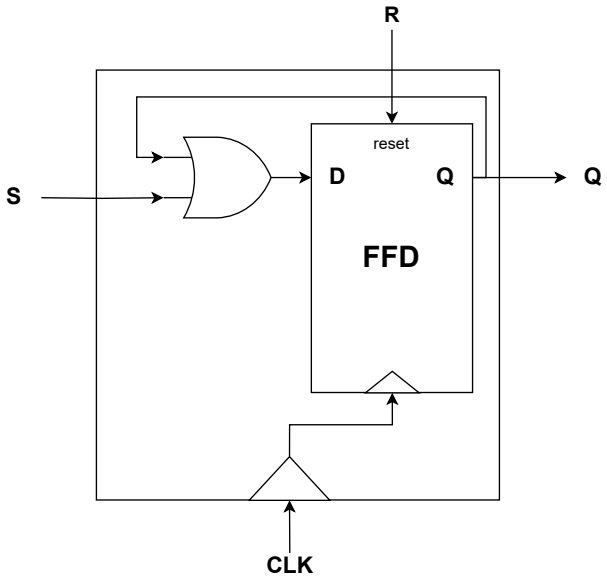


Figura 4.5: Flip Flop D con adaptación para trabajar como un FFD Keep.

4.3.3. Registro Paralelo-Paralelo

El almacenamiento de los datos estará dado por un registro paralelo-paralelo de 8 bits, como el mostrado en la Figura 4.6.

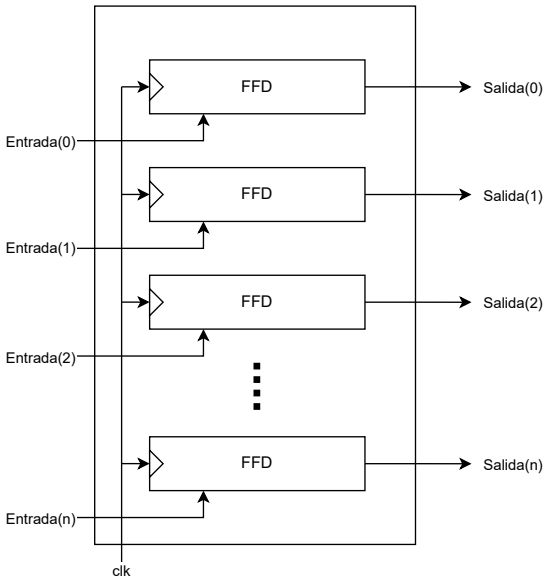


Figura 4.6: Registro Paralelo-Paralelo.

Este consiste en dos buses de datos de ocho bits, uno de entrada y otro de salida. La información es almacenada con un flanco positivo de la señal de reloj, dicha señal de reloj es síncrona para todos los FFD del registro.

4.3.4. Registro Paralelo y de Corrimiento

Para el diseño del registro paralelo y de corrimiento se necesitó primeramente de una modificación a la descripción del FFD visto en la Subsección 4.3.1, ya que este no tiene la línea de preset contemplada dentro de su descripción, en este caso se requirió modificar la descripción del original ya que el comportamiento del preset en este diseño no se puede emular con lógica externa. Por otra parte, el FFD interno a los Bloques Lógicos Configurables del FPGA ya contemplan esta señal, de tal forma que a partir de él se puede generar una versión de FFD con preset. Este diseño se mantendrá independiente al FFD original de la Subsección 4.3.1 ya que no siempre se necesitará la línea de preset y resulta más fácil omitirla con el diseño base que conectarla a un *GND* cada que se requiera usar un FFD.

Posterior a esta modificación, se establece la arquitectura del registro de corrimiento que permitirá que actúe de cuatro modos:

- Registro Paralelo-Paralelo
- Registro Serie-Serie
- Registro Paralelo-Serie
- Registro Serie-Paralelo

Para esto, se configuró el bloque básico visto en la Figura 4.8 que permite cargar datos en paralelo mediante el empleo del preset y el reset, mientras que la carga en serie se da por la señal de reloj.

```
library ieee;
use ieee.std_logic_1164.all;

entity FFDP is
port(clk, reset,preset, d: in std_logic;
q: out std_logic
);
end entity;

architecture behavioral of FFDP is
begin
process(clk, reset,preset)
begin
if (reset='1') then
        q<='0';
elsif (preset = '1') then
        q<='1';
elsif(rising_edge(clk)) then
        q<=d;
end if;
end process;

end behavioral;
```

Figura 4.7: Código en VHDL del FFD con línea de Preset descrita de forma algorítmica.

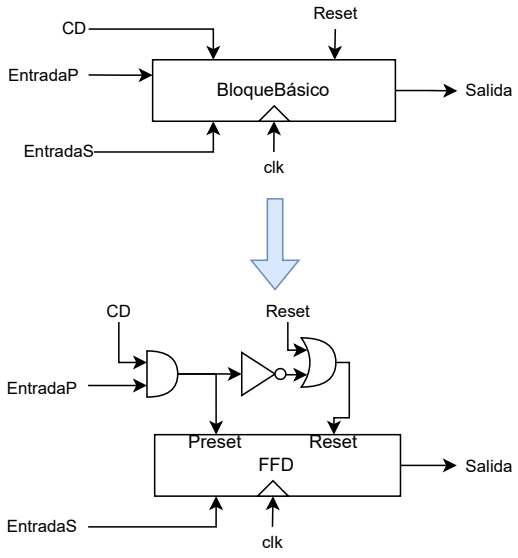


Figura 4.8: Bloque básico del Registro Paralelo y de Corrimiento.

El motivo para diseñar un registro Paralelo-Paralelo como el de la Subsección 4.3.3 y un registro Paralelo y de Corrimiento se hizo con el fin de que para cada caso sólo se use el hardware mínimo necesario.

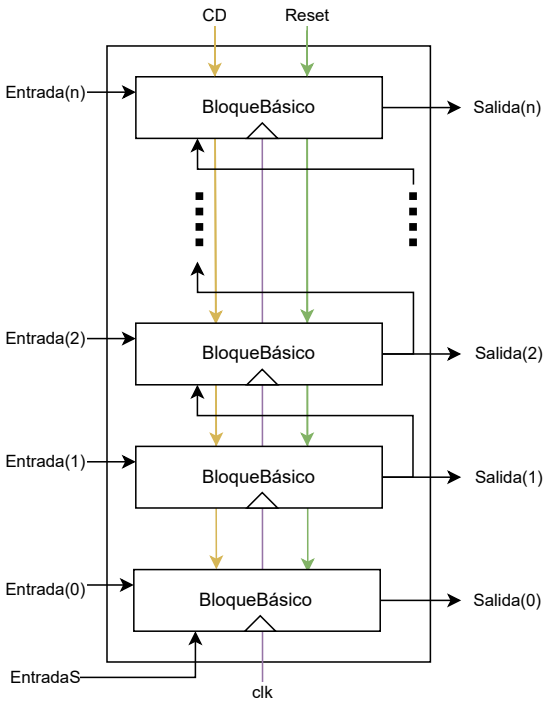


Figura 4.9: Diseño interno del Registro Paralelo y de Corrimiento.

Registro de Dirección Este es un caso especial del registro de corrimiento, puesto que este contiene la misma estructura pero con un registro menos, ya que la dirección de esclavo es de siete bits, mientras que los datos contienen ocho. Esta modificación se hizo con la finalidad de ahorrar lo más posible en cuanto a recursos lógicos.

4.4. Divisor de frecuencia

4.4.1. Diseño a nivel sistema

Debido a que esta arquitectura está considerada para trabajar en conjunto con un microprocesador como parte de sus periféricos, operará con el reloj que se le asigne, que normalmente es el mismo con el que opera el microprocesador y dado que este funciona a frecuencias mucho más altas de lo que requerimos $(0 - 1[Mbits/s])$ es necesario contar con un divisor de frecuencia que a la salida entregue un valor de reloj adecuado a los rangos de velocidad de transmisión dados por la especificación.

A continuación, se muestra la descripción de los elementos que se requieren para operar al Divisor de Frecuencia propuesto:

Líneas de configuración

- Selector: Mediante este bus se escoge la velocidad de operación interna del módulo, que está directamente ligada con la tasa de transmisión del bus SCL, con las consideraciones que se muestran más adelante.

Las banderas:

- Error: Se activa por algún error en el proceso de configuración del divisor.

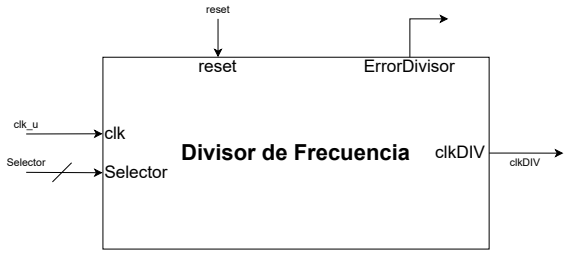


Figura 4.10: Divisor de frecuencia a alto nivel propuesto.

4.4.2. Diseño a nivel detalle

La especificación indica valores de operación definidos para el reloj, mostrados en la Tabla 2.2; sin embargo es posible operar a frecuencias menores a las que indica; es decir, si trabajamos con una frecuencia de 100$[kbits/s]$ idealmente podemos operar a una frecuencia menor. Por ello, el diseño que se propone es un divisor básico, constituido por un contador basado en Flip Flops tipo Toggle. Con un análisis básico del funcionamiento de un Flip Flop T en la configuración mostrada en la Figura 4.11 podemos corroborar que la frecuencia de la señal a la salida es de la mitad que a la entrada, como referencia está la Figura 4.12.

La propuesta de este contador asíncrono se da debido a que su salida es directa de los Flip-Flops, por lo que al no depender de circuitos combinacionales a la salida, se evitan mayores tiempos de propagación que en conjunto pueden generar señales transitorias no deseadas. Esto es crítico ya que su salida controlará las señales de reloj de las demás máquinas de estado del sistema; por lo tanto, un simple glitch afectaría todos los demás relojes. Entonces, la limpieza de la señal de salida es crítica para un correcto funcionamiento.

Asimismo, es importante aclarar que una máquina síncrona que tenga salida puramente secuencial puede ser contemplada, pero debido a que cada señal de salida es un Flip-Flop, el diseño se vuelve más complicado y difícil de escalar. Igualmente, la consideración de un Flip-Flop a la salida que filtre la señal se vuelve obsoleta al considerar que el reloj de este debe ser mucho mayor al de la señal que estará filtrando y dado que el reloj lo asigna un microprocesador *genérico*, no es factible que este pueda mandar más de una línea de reloj para un solo periférico.

Con todo lo anteriormente mencionado, el diseño de un contador asíncrono con base en Flip Flop tipo T, se vuelve la mejor opción debido a su diseño sencillo, a la fácil escalabilidad que tiene, y a la limpieza en su señal de salida.

Como consideración de diseño, tenemos que contemplar que internamente en la Artix 7 y en la mayoría de FPGAs contamos con Flip Flops D dentro de sus Bloques Lógicos Configurables, por lo que el primer paso es describir un circuito combinacional que en conjunto funcione como una máquina de estados simulando el funcionamiento de un Flip Flop T con un D interno, a través de su diagrama de estados (Figura 4.13).

Con la reducción por medio de Mapas de Karnaugh se obtiene que el circuito combinacional requerido es una compuerta XOR, quedando el diseño general como se muestra en la Figura 4.15

Consecuentemente, si tomamos que a la salida de la configuración de la Figura 4.11 tenemos un nuevo reloj, este se puede conectar en cascada a la entrada de reloj del bloque

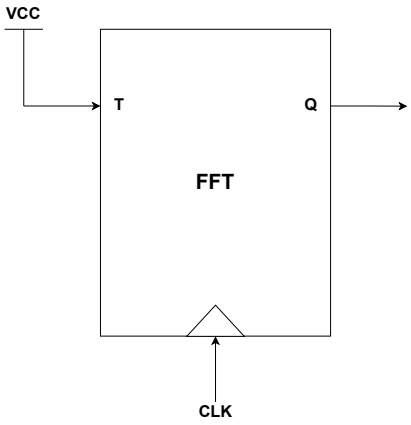


Figura 4.11: Flip-Flop Toggle en configuración divisora de frecuencia.

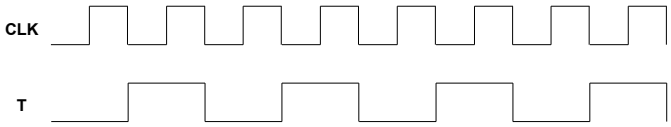


Figura 4.12: Gráfica de salida del Flip-Flip T.

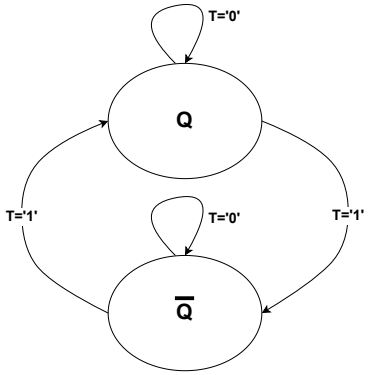


Figura 4.13: Diagrama de Estados de un FFT.

Estado presente	Entradas	Estado Futuro
D	T	D(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

Figura 4.14: Tabla de verdad de un FFT.

siguiente, obtendremos una nueva señal de reloj pero ahora dividida entre 4 con respecto a la original.

Por ejemplo, si se usa una tarjeta de desarrollo Basys 3 la cual trabaja con un reloj a $50[MHz]$, podemos hacer un cálculo rápido para ver cuántos elementos en cascada son necesarios para reducir la frecuencia del reloj a la requerida.

$$Frecuencia = \frac{FrecuenciaBasys}{2^N}, \tag{4.1}$$

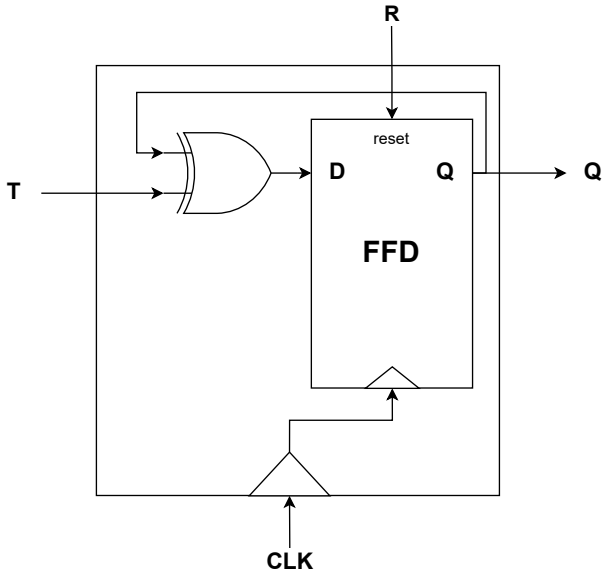


Figura 4.15: Flip Flop D con adaptación para trabajar como un Flip Flop T.

donde N es el número de módulos conectados en serie.

$$N = \log_2 \left(\frac{FrecuenciaBasys}{Frecuencia} \right) \tag{4.2}$$

$$N = \log_2 \left(\frac{50[MHz]}{100[kHz]} \right) = 8{,}96,$$

Ya que sólo se consideran los números enteros para N, se redondea hacia arriba para garantizar que se tiene una frecuencia menor a la máxima que indique el estándar. Podemos ver que para operar a frecuencias cercanas a $100[kHz]$ con un reloj de 50[MHz] como el de la Basys 3, es necesario tener 9 módulos conectados en serie. Es importante resaltar que posteriormente esta frecuencia se verá más reducida por el generador de señal SCL del maestro, por lo que la velocidad de transmisión final depende del factor de reducción que también tendrá ese componente, que se verá en el apartado 4.7.2. Con este factor en consideración, la fórmula que se deberá emplear es la ecuación 4.4. Una vez obtenido este dato se deberá convertir a formato binario para poderlo cargar a los registros del selector.

$$VelocidaddeTransmision = \frac{FrecuenciaBasys}{3 * 2^N}, \tag{4.3}$$

$$N = \log_2 \left(\frac{FrecuenciaBasys}{3 * VelocidaddeTransmision} \right) \tag{4.4}$$

Aunque una de las ventajas de un FPGA reside en la facilidad de cambiar su arquitectura interna, este diseño está contemplado para trabajar en condiciones espaciales; por lo tanto, aunque existan formas de reconfigurar la arquitectura de forma dinámica aún en esas condiciones, es ideal y más fácil establecer una cantidad base de divisores en cascada y que a su vez se permita seleccionar por cuántas fases de reducción tendrá que pasar la señal original. Esto se consigue mediante un multiplexor *N a 1* que se encarga de

seleccionar la salida de alguno de los bloques previamente establecidos, como se muestra en la Figura 4.16.

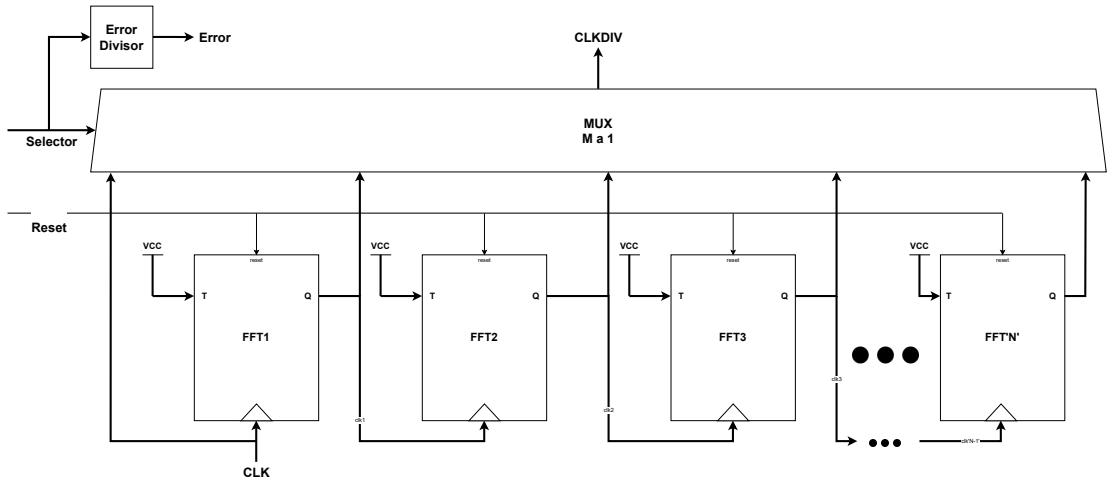


Figura 4.16: Diseño del Divisor de frecuencia.

El bloque de Error Divisor se destino a que, conforme crece la cantidad de bloques que se necesitan, se vuelve más probable que la cantidad de entradas del multiplexor exceda las que son realmente necesarias, entonces este exceso de entradas son consideradas como un error. Idealmente la cantidad de entradas M del multiplexor es igual a los N bloques divisores de frecuencia.

4.5. FIFO - First Input First Output

4.5.1. Diseño a nivel sistema

Como se mencionó en el apartado 4.1.1, esta arquitectura está enfocada a trabajar como periférico de un microprocesador, por ende es necesario garantizar que aún si el procesador no está disponible para atender la información manejada por el módulo, ya sea para ser enviada o guardar los datos recibidos este pueda seguir operando con normalidad. Entonces se propone el bloque de FIFO, que actúa como una memoria temporal para la información.

Banderas consideradas para la FIFO.

- FIFO Vacía
  Informa que la FIFO se encuentra vacía.
- FIFO Llena
  Informa que la FIFO está en su máxima capacidad.

4.5.2. Diseño a nivel detalle

El diseño a detalle del módulo encargado de almacenar temporalmente la información se desarrolló con base en una arquitectura FIFO donde el almacenamiento es a través

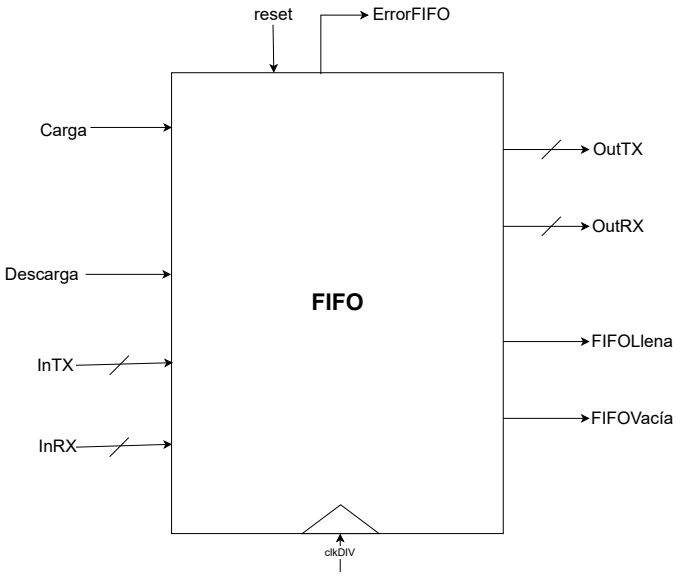


Figura 4.17: FIFO a alto nivel propuesto.

de una serie de ocho registros paralelo-paralelo de 8 bits cada uno en configuración de cascada, por lo que el primer dato que se almacena es el que se muestra a la salida, y los datos que vayan llegando posteriormente se apilan uno tras otro. Estos registros son controlados a través de dos tipos de acciones: carga o descarga. Con la operación de carga se realiza el almacenamiento del dato que se muestra a la entrada de la FIFO, dicho contenido se almacena en la localidad que está habilitada por el apuntador; mientras que para la operación de descarga se genera un desplazamiento de la información de todos los registros, desechando el primer dato almacenado que se muestra a la salida y sustituyéndolo por el consecuente en la pila.

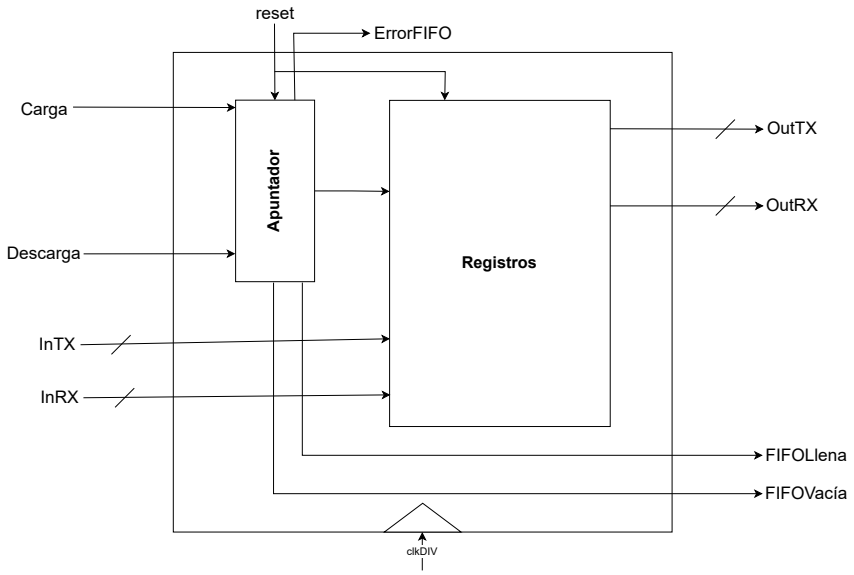


Figura 4.18: FIFO propuesta.

Esta propuesta está subdividida en un apuntador y la sección de registros paralelo paralelo con dos apartados de almacenamiento. La primera es la de datos a transmitir,

con datos cargados por el usuario y la segunda es el almacenamiento de datos recibidos, que son cargados a través de los módulos maestro o esclavo.

Apuntador

Primeramente, para el diseño del apuntador, se define que su función es ser el encargado de seleccionar en qué localidad se dará la carga de información a los registros; es decir, si el apuntador indica la posición '0', cuando se dé la señal de carga, el valor a la entrada de *InTX* e *InRX* se cargará al primer registro y el valor del apuntador cambia a '1' y así consecuentemente hasta llenar los 8 registros de la FIFO.

Es entonces que el diseño consiste en un contador de 9 estados: El primero apunta al primer registro, además de que indica que la FIFO está vacía; consecuentemente, los otros siete estados son dedicados a habilitar la carga de datos en las localidades de los demás registros; el último y más alto se encarga de indicar que la FIFO está llena y no apunta a una localidad de almacenamiento, sirviendo únicamente como bandera.

Para cumplir con lo anterior, se generó el diagrama de estados del contador mostrado en la Figura 4.19

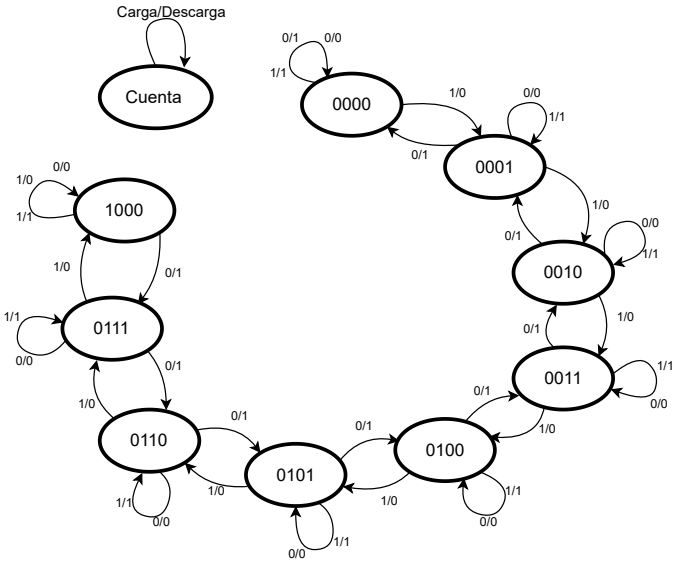


Figura 4.19: Diagrama de estados del apuntador.

Una vez detallado su diagrama de estado, se muestra en la Tabla 4.6 las salidas para cada uno de los estados.

Con el apuntador y la sección de registros paralelo paralelo, vista en la Subsección 4.3.3, se genera la interconexión mostrada en la Figura 4.20. Como se dijo anteriormente, el valor del contador nos sirve para establecer la localidad en la que se carga el valor a la entrada, por lo que del valor de la cuenta se sacan líneas de habilitación que enmascara la señal de carga para los registros a los que no se apunta, provocando que sólo se actualice el valor del registro en la localidad a la que apunta el contador.

Estas líneas alimentan a la señal de control que en conjunto con la señal de descarga maneja el reloj de los registros, aunque la señal de descarga es independiente de cualquier habilitación puesto que sirve para realizar un desplazamiento sobre de todos los registros,

Estados Q		Salidas		
Binario	Decimal	Llena	Vacío	Error
0000	0	0	1	0
0001	1	0	0	0
0010	2	0	0	0
0011	3	0	0	0
0100	4	0	0	0
0101	5	0	0	0
0110	6	0	0	0
0111	7	0	0	0
1000	8	1	0	0
1***	9-15	0	0	1

Tabla 4.3: Salidas para cada estado del apuntador.

desplazando la información en cadena, donde cada registro transmitirá la información contenida al registro inmediato anterior.

De forma más detallada y visual, la arquitectura quedaría como la Figura 4.20.

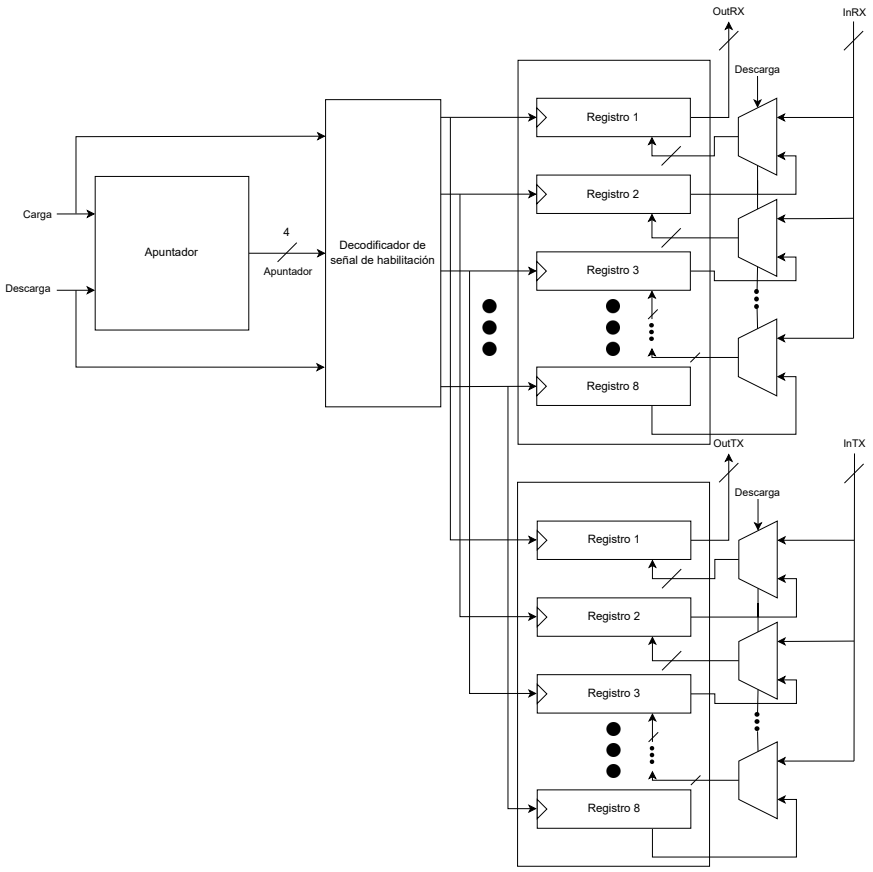


Figura 4.20: FIFO Bajo Nivel Detallada.

4.6. Esclavo - Objetivo

4.6.1. Diseño nivel sistema

El esclavo tiene la tarea de reconocer la condición de inicio para posteriormente comparar si la dirección propia coincide con la que envía el maestro. Asimismo, necesita identificar el bit de sentido en el que se desarrollará la comunicación, ya sea actuando como el receptor o el transmisor y por último, deberá reconocer la condición de parada. Para ello es necesario que cuente con los siguientes registros de configuración

Registros de configuración

- Carga de Dirección: Se utiliza como señal de carga para la dirección de esclavo
- Dirección de esclavo: Es la dirección propia del módulo
- Carga de Datos: Sirve para que el usuario cargue los datos a enviar
- Datos: A través de este se lee la información recibida o se carga la que se quiere enviar
- Habilitación de FIFO: Con este se indica si los datos que se transmitirán se tomarán de lo que contiene la FIFO y no de lo que cargue el usuario directamente, también indica que los datos recibidos pasarán directamente del registro de datos a la FIFO.
- Clock Stretch: Se usa para provocar que el maestro no use la línea de SCL, haciendo que tenga que esperar a que el esclavo esté disponible nuevamente. Se usa para dar tiempo de que el procesador lea o cargue la información al registro de datos para que pueda continuar la comunicación

Banderas

- Bus Ocupado: Indica que las líneas están siendo ocupadas, independientemente de si el esclavo es parte de dicha comunicación
- Error: Indica que el esclavo no obtuvo un ACK.

Los submódulos que componen al esclavo se muestran en la Figura 4.22, a continuación se detalla la función de cada uno de los bloques mostrados, iniciando por el detector de Start que como su nombre indica, está a la espera de que las líneas de SDA y SCL generen la condición de inicio, posterior a la detección de dicha señal, se genera un pulso que valida el bit de un registro, este indica al módulo que está habilitado y mantiene dicho estado hasta que llega el pulso de la condición de *Stop*, generada por el bloque detector de Stop. Mientras se encuentra habilitado, el esclavo espera la recepción de la dirección de esclavo enviada por el maestro, almacenando los datos recibidos en un registro operando de forma serial con la línea de SDA, para conocer el momento en el que se puede verificar que la dirección es correcta, se emplea un contador, que empieza a contar cada flanco de bajada de la señal de SCL. Una vez pasados ocho pulsos de reloj, activa la señal que carga el valor del comparador para identificar si el valor coincide con el del registro de dirección, en caso de que coincidan, activa la señal de validado. Al mismo tiempo, se encarga de identificar el valor del bit de lectura/escritura. Posterior a esto, envía el ACK y continúa

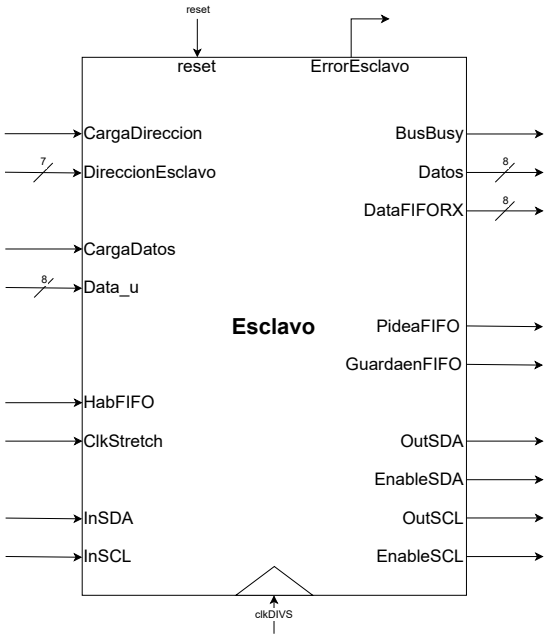


Figura 4.21: Esclavo a alto nivel.

su funcionamiento dependiendo del bit de R/S. En caso de ser una escritura por parte del maestro, se libera la línea de SDA y procede a leer los datos que se reciben durante los ocho pulsos de reloj y en el momento de enviar nuevamente un ACK, carga los datos del registro serial al registro de datos para ponerlos a disposición del dispositivo asociado. En ese momento se emplea el módulo de ClkStretch, que permite suprimir la línea de SCL para provocar que el maestro espere al esclavo. Esta función es opcional, y se debe considerar que el maestro debe tener capacidad de entender este comportamiento. Una vez generada esta secuencia, se puede seguir enviando datos de 8 bits indefinidamente, hasta que el maestro envíe una condición de stop.

Para el caso de que el bit de RS indique una operación de lectura, después del ACK del esclavo, el esclavo tiene la tarea de enviar los datos contenidos en el registro serial, cuya operación es ahora de forma paralela-serial, ya que el usuario tiene disponible un bus de 8 bits para cargar los datos que planea enviar y a través de un Multiplexor envía de forma serial cada uno de los bits de información. En este modo es el maestro el que tiene que enviar el ACK después de cada trama de datos, dicha función también sirve para indicar si el maestro quiere terminar la comunicación, ya que al enviar un No Acknowledge, el esclavo entiende que debe soltar la línea. Con esto se evita que haya un cruce entre la condición de stop que vaya a enviar el maestro y la información que esté enviando el esclavo, esta operación de soltar la línea la ejecuta un registro que detecta si el maestro envió un ACK o un NACK. Al igual que la operación de escritura, se pueden enviar muchas tramas de datos hasta que se dé la condición de stop y nuevamente se tenga que ejecutar la trama como se muestra en la Figura 2.5

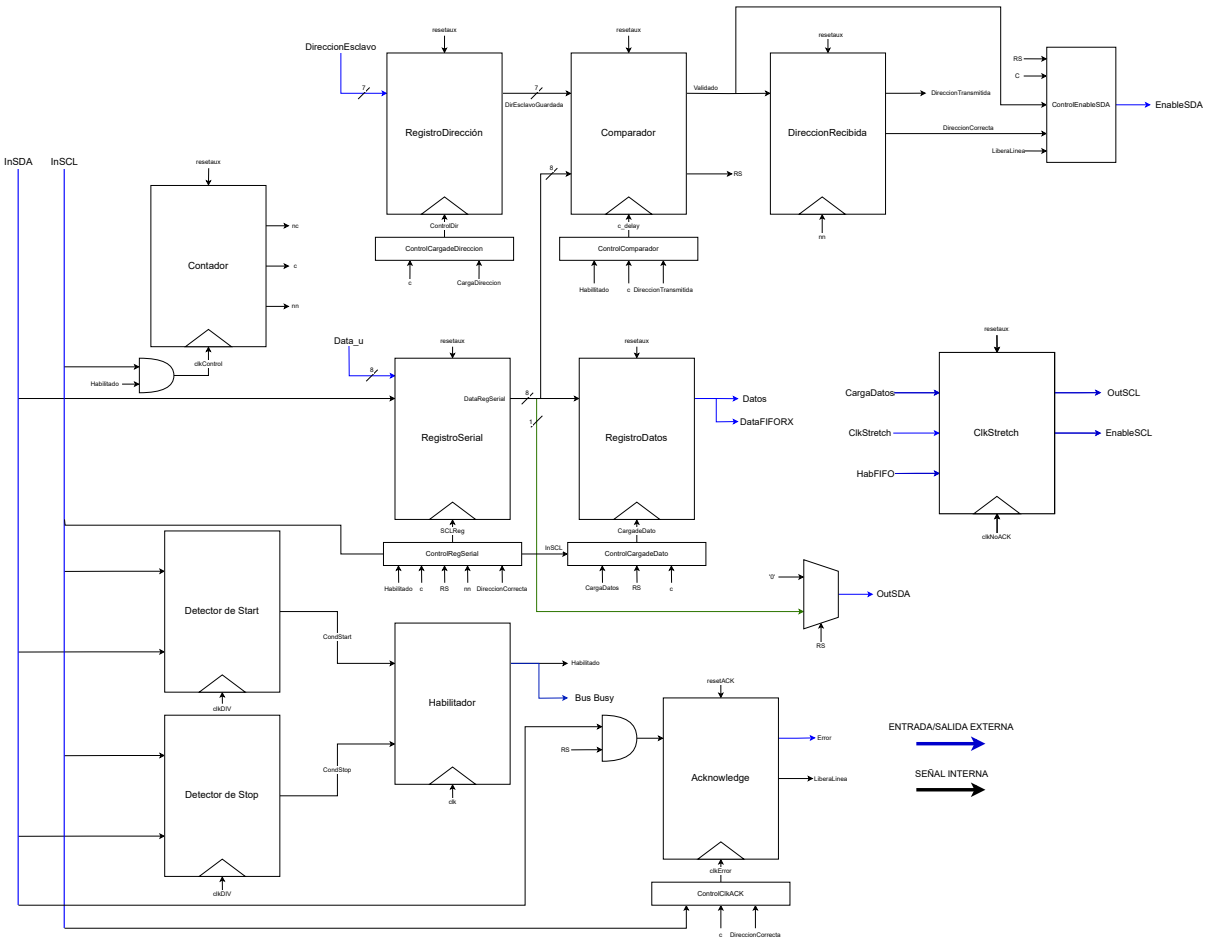


Figura 4.22: Esclavo a bajo nivel.

4.6.2. Diseño nivel detalle

Detector de Start

Para el diseño del detector de Start se partió de la propuesta presentada en la Figura 4.23. Sus dos entradas son las líneas de datos y reloj del bus, teniendo como señal de reloj la señal proveniente del divisor de frecuencia. Consecuentemente, se generó la propuesta de diagrama de estado mostrada en la Figura 4.23.

La decisión de asignar como reloj del detector a la señal proveniente del divisor de frecuencia y no directamente la señal del reloj base se tomó después de que al realizar pruebas físicas, el módulo esclavo soltaba el bus de datos unos instantes antes de lo que debería, lo que provocaba una condición de Stop indeseada; esta condición se podía evitar mediante una reducción en el reloj del detector descrito, ya que al ser tan rápida, si se reduce la velocidad de muestreo es posible que no se alcance a captar la condición. Esta solución funcionó más como un parche ya que no solucionaba el problema de fondo que era un tema de sincronización de señales entre el esclavo y el maestro, pero funcionaba igualmente como protección para que no se capten los transitorios provenientes de la línea. Aunque en esta situación se dio para una condición de Stop, se aplicó la misma idea al detector de Start ya que igualmente podría ocurrir una condición de start debido a transitorios en el bus aún cuando el bus se encuentre inactivo, por lo que dicho efecto

activaría el esclavo innecesariamente. Igualmente la especificación contempla un filtro de glitches para velocidades mayores a 400[$kbits/s$], por lo que esta reducción podría actuar como un complemento a este filtro y en caso de no ser necesitada o de querer usar el reloj base únicamente se deja sin modificar el registro selector de velocidad del divisor, lo que deja preseleccionado la frecuencia del reloj base.

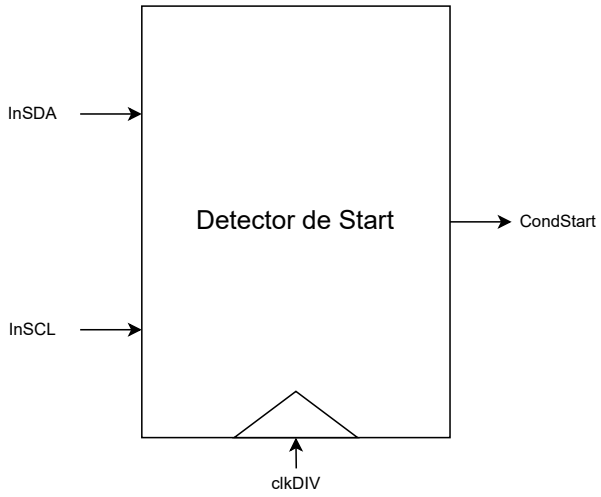


Figura 4.23: Detector de Start.

Una vez explicado lo anterior, se propone el diagrama de estados de la Figura 4.24. Este diagrama se implementó usando el código Gray, para que los estados siguientes al presente sólo cambien en un bit, lo que genera un cambio entre estados más sutil, reduciendo las transiciones entre un estado y otro, las cuales pueden activar momentáneamente las compuertas y mostrar errores a la salida del sistema. Esto fue evidente dentro de las primeras pruebas, en las cuales se activaba la señal de *Stop* por instantes muy cortos en el detector de *Stop*, ya que en un primer diseño al pasar del estado "00" al "11" pasaba momentáneamente por el estado "10" el cual era el estado que activaba dicha señal de *Stop*. Esto mismo se decidió implementar en el diseño del detector de start con el fin de *blindar* al diseño contra este tipo de fallas y a que inicialmente se contempló hacer el esclavo sensible a la condición de *Restart*, lo que podría generar que en alguna transición de estados se activara esta señal de *Restart*, aunque después se decidió retirar debido a que esta condición no tiene una funcionalidad específica dentro de este esclavo, como sí lo es en el caso de los módulos sensores con esclavo I2C que tienen registros internos configurables.

Estados Q	Salidas
	CondStart
00	0
01	0
11	1
10	0

Tabla 4.4: Salidas para cada estado del detector de Start.

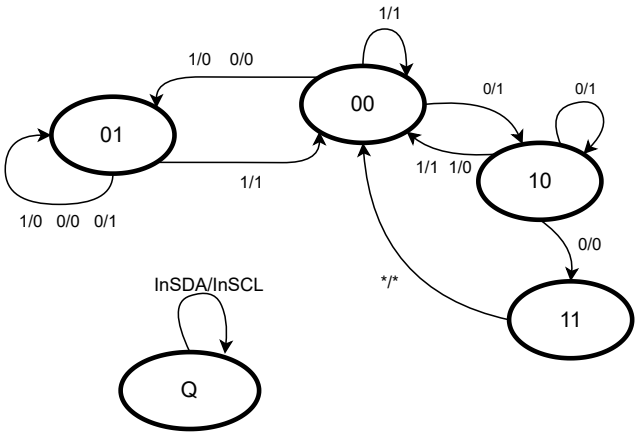


Figura 4.24: Diagrama de estados del Detector de Start.

Detector de Stop

La concepción del detector de Stop es muy similar a la del detector de Start, incluso comparten el reloj proveniente del divisor de frecuencia, situación explicada en el apartado anterior.

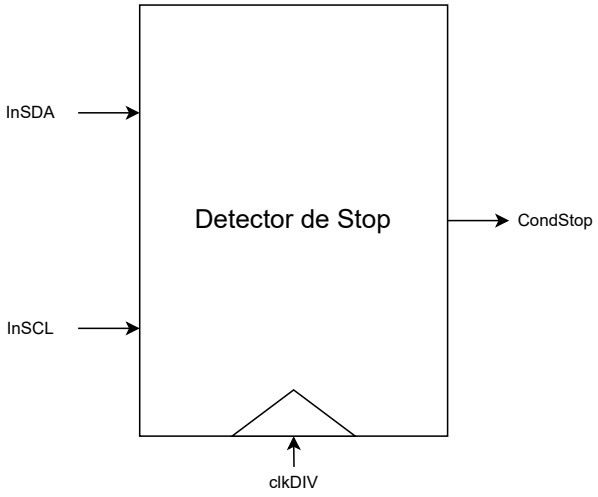


Figura 4.25: Detector de Stop.

Continuando, el diseño se produjo a través del diagrama de estado de la Figura 4.26, siguiendo una formulación similar a la propuesta para el apartado 4.6.2, referente al detector de start, en cuanto al código Gray. Además, igual que en el diseño anterior, se integra el estado "01" como estado para contener la ejecución de un patrón no reconocido, ya que sin este se podía dar la condición de Stop a partir de cualquier patrón, y no forzosamente de tener ambas líneas en un cero lógico, por lo que el agregar este estado fuerza a que ambas líneas tengan que estar abajo para iniciar con la detección de *Stop*.

Asimismo, aunque en la especificación se indica que una condición de start no debería estar inmediatamente seguida de una de stop, hay ciertos dispositivos que pueden operar con dicha acción y este esclavo es uno de ellos, ya que está diseñado para aceptar la condición de stop en cualquier momento. Esta decisión se tomó al momento de la realización de pruebas, puesto que en ciertos momentos el maestro de la Tiva enviaba un

pulso transitorio a través de la línea de *SCL* previo a la condición de stop, generando una cuenta *parásita* en el contador debido a que el hardware del FPGA es lo suficientemente rápido como para captar dicho pulso. Entonces, con la finalidad de evitar una mala sincronización se decidió que no sea el contador el que habilite el detector de stop por medio de la señal de cuenta en "1" que es el momento en el que se da la condición de stop, si no que permita que se dé en cualquier momento.

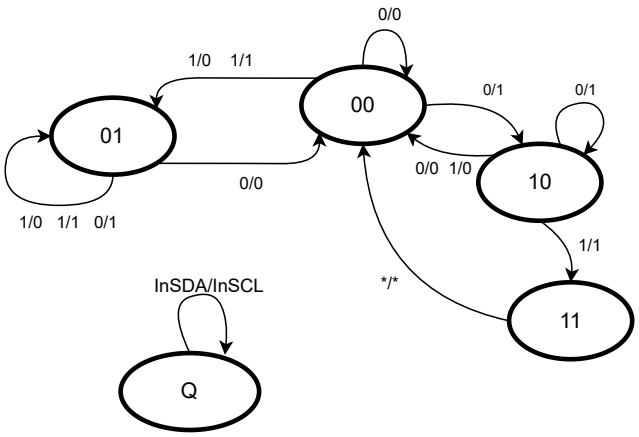


Figura 4.26: Diagrama de estados del Detector de Stop.

Estados Q	Salidas
	CondStop
00	0
01	0
11	1
10	0

Tabla 4.5: Salidas para cada estado del detector de stop.

Contador

El contador del esclavo es uno de los elementos esenciales en el funcionamiento del módulo, ya que este es el que genera la mayoría de señales de control que se verán en el apartado 4.6.2, donde se abordan las señales de control del esclavo.

Este elemento se encarga de recibir la señal de reloj proporcionada por la línea de *SCL* e incrementar su cuenta con cada flanco descendente del pulso. Por ello, el sistema se mantiene en cero hasta después de la condición de start y una vez recibidos los ocho pulsos con la dirección de esclavo correcta, es en el noveno que la cuenta regresa a cero y se indica que es momento de dar el ACK y durante las tramas de datos, esa misma señal funge como carga de datos para los registros serial y paralelo. Es decir, este contador sirve para sincronizar señales de control para que no se modifiquen los registros internos durante la operación del módulo.

Para su concepción, se estableció un diseño síncrono, ya que a diferencia del divisor de frecuencia, este no actuará como reloj del sistema, si no más bien como señal de control en conjunto con otras señales. En cambio, el ser un diseño síncrono nos permite analizar

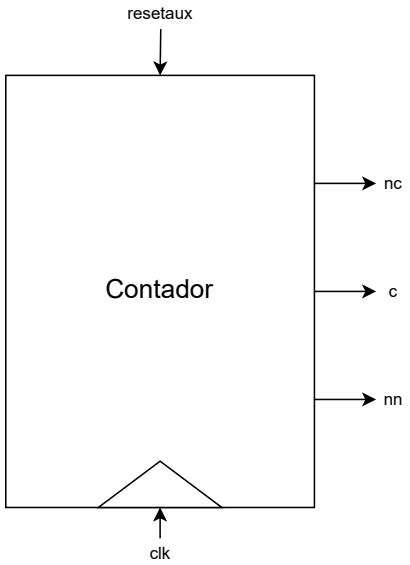


Figura 4.27: Contador del esclavo.

todos sus estados de forma sencilla, por medio del diagrama de estados de la Figura 4.28 y en caso de encontrarse en un estado prohibido, regresar al estado base.

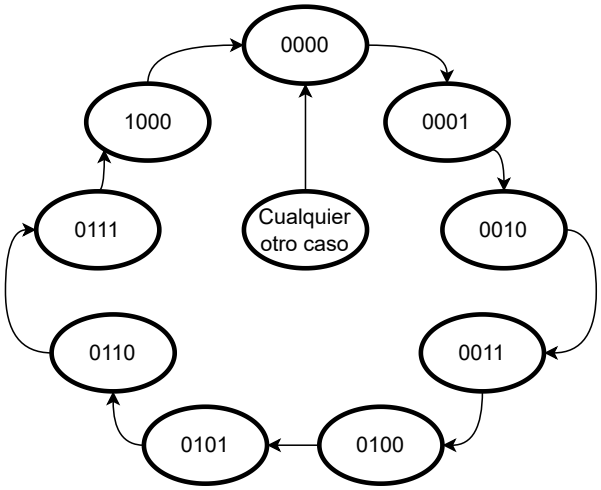


Figura 4.28: Diagrama de estados del Contador esclavo.

La razón para que el contador trabaje con el flanco negativo del reloj se muestra en la Figura 4.29. Ahí se muestra que de esta manera da tiempo para que la señal en *SDA* se estabilice antes de que llegue el reloj de *SCL*, al contrario de trabajar con flancos positivos. Este ejemplo muestra el caso con el acknowledge, pero esto mismo sucede con el envío de datos cuando el esclavo funciona como transmisor.

Comparador

El comparador en esencia tiene dos trabajos, el primero de ellos se encarga de validar la dirección de esclavo. Para ello, hace un cotejo a través de compuertas xnor entre cada uno de los bits almacenados en el registro de dirección y los que recibió en el registro serial que recibe información de la línea de datos; si todos coinciden el resultado de este

Estados Q		Salidas		
Binario	Decimal	nc	c	nn
0000	0	0	1	0
0001	1	0	0	1
0010	2	0	0	0
0011	3	0	0	0
0100	4	0	0	0
0101	5	0	0	0
0110	6	0	0	0
0111	7	0	0	0
1000	8	1	0	0
1***	9-15	0	0	0

Tabla 4.6: Salidas para cada estado del contador esclavo.

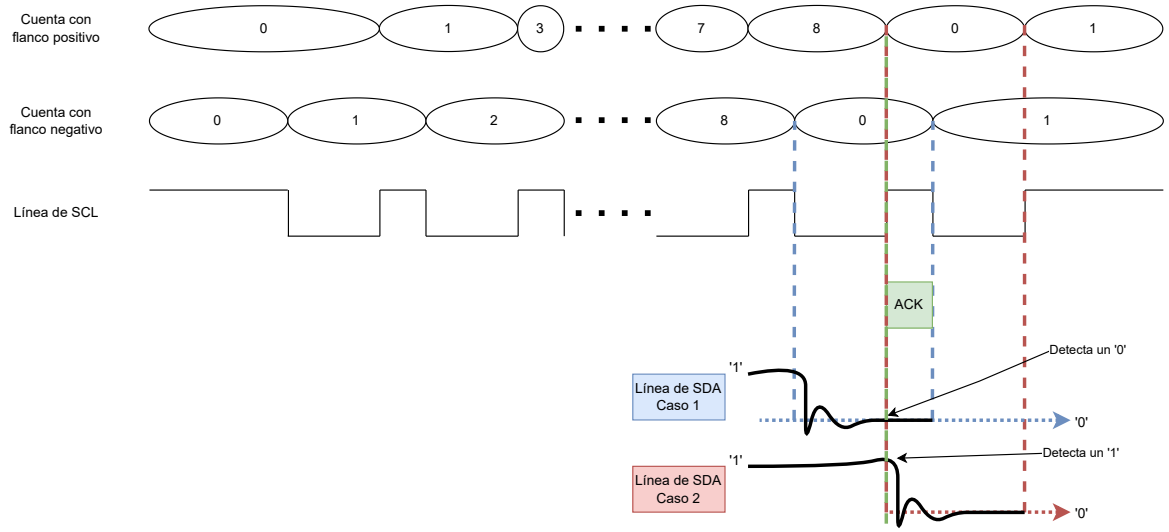


Figura 4.29: Comparación de utilización de flanco positivo y negativo.

proceso se almacena en un FFD Keep por el resto de la comunicación hasta la llegada de la condición de parada. En paralelo, se almacena el valor del bit de escritura/lectura leyendo el bit menos significativo del mismo registro serial; este valor igualmente se almacena en un Flip Flop dedicado y se mantiene durante toda la comunicación.

En la Figura 4.31 se puede observar el comparador internamente.

Detector de Acknowledge

Cuando el esclavo actúa como transmisor, debe recibir el acknowledge por parte del maestro después de cada trama de datos enviada, por eso a la entrada del detector se habilita por medio del bit de *RS* puesto que sólo en ese modo tiene un propósito específico.

La máquina dedicada a detectar el reconocimiento por parte del maestro, está constituida por dos bloques que se muestran en la Figura 4.33. El primero de izquierda a derecha es el que detecta el estado del bit en el momento del flanco positivo y el segundo es el que, en caso de obtener un NACK en el primero, se encarga de liberar la línea para

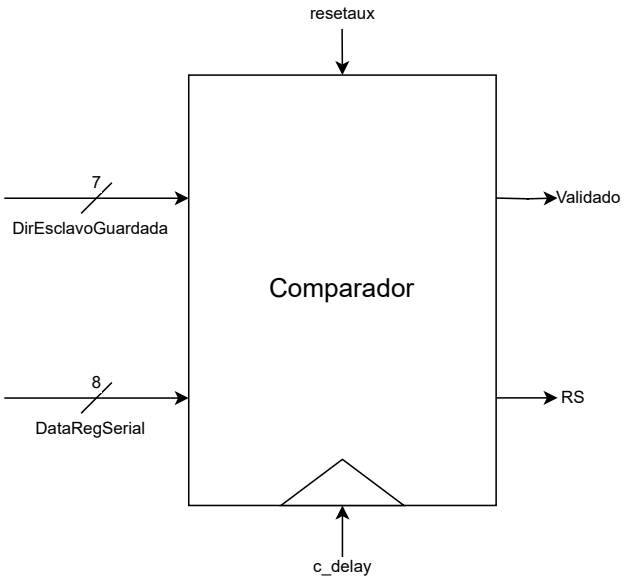


Figura 4.30: Comparador de direcciones.

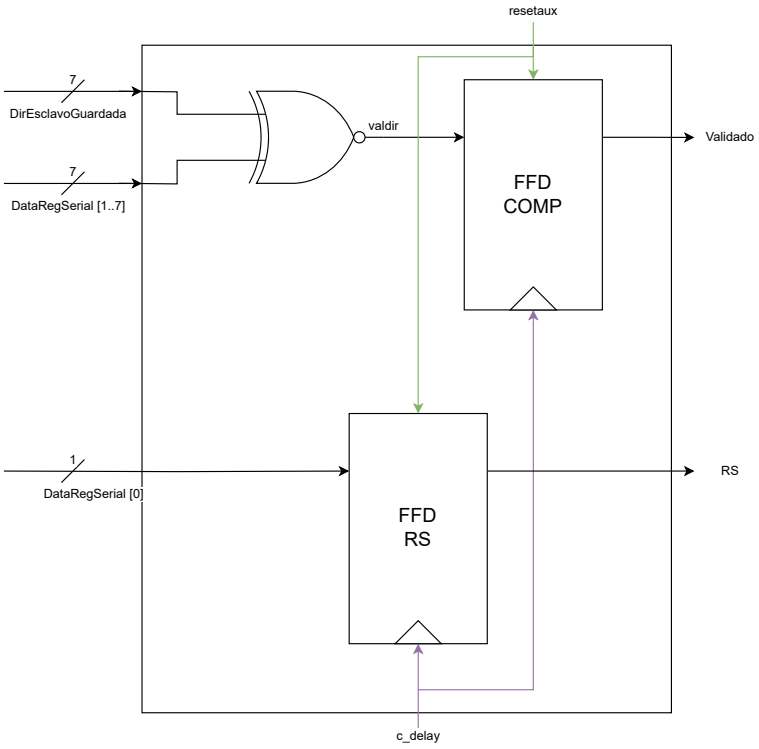


Figura 4.31: Componentes del comparador.

que el maestro pueda enviar la condición de Stop, por lo que aprovecha el flanco negativo de la señal de reloj para actualizar el estado y liberar la línea en el momento en el que el maestro tomará el bus, justo después del tiempo de acknowledge.

Aunque en teoría la máquina marca un error para el caso de un NACK, este no se debe considerar tal cual como uno, es más bien una bandera para indicar que el maestro no dio su reconocimiento, pero se deja como una bandera de error porque en caso de que se envíe el NACK y el maestro no envíe la condición de stop, efectivamente estaría

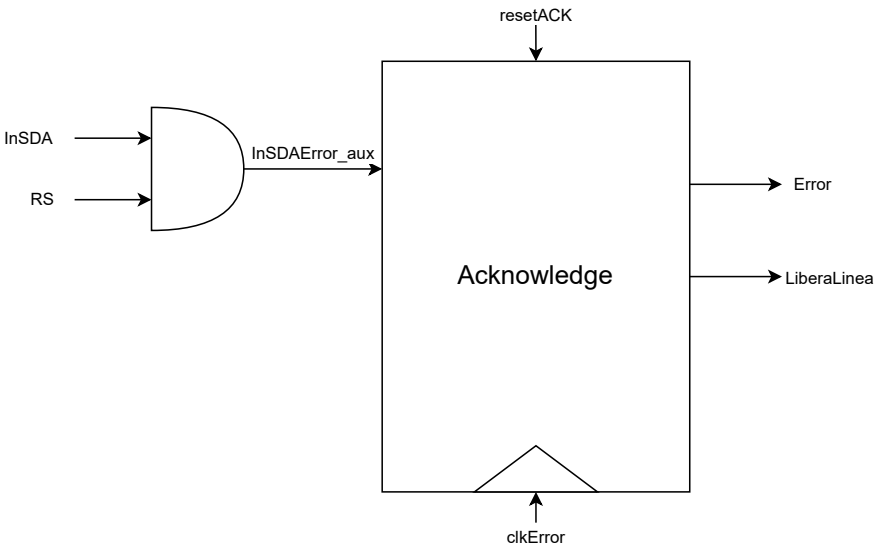


Figura 4.32: Detector de acknowledge.

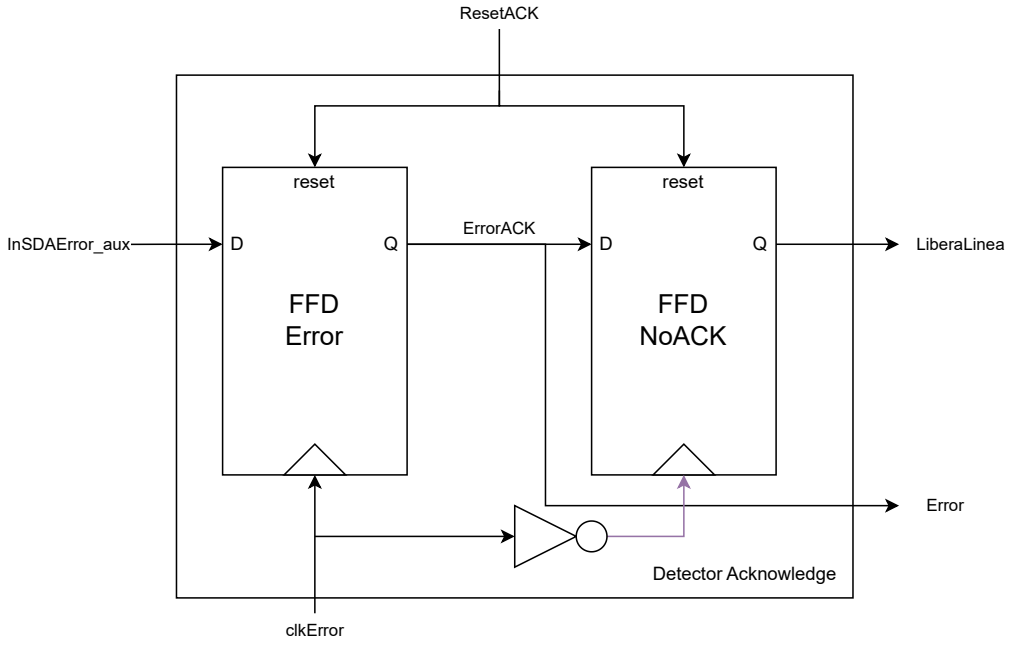


Figura 4.33: Componentes del detector de acknowledge.

actuando erróneamente.

Dirección Recibida

La constitución de esta máquina es similar a la anterior, ya que su concepción es por medio de dos bloques. Dentro del proceso de diseño se establecieron dos propuestas, donde al final se decidió optar por el diseño asíncrono, ya que el primer acercamiento fue por medio de diagramas de estado como el de la Figura 4.35, pero este diseño requeriría de emplear más compuertas lógicas y al ser una máquina que sólo se necesita para almacenar un estado y no para generar algún comportamiento en concreto, se optó por un diseño asíncrono que requiere de menos elementos lógicos, como el mostrado en la Figura 4.36.

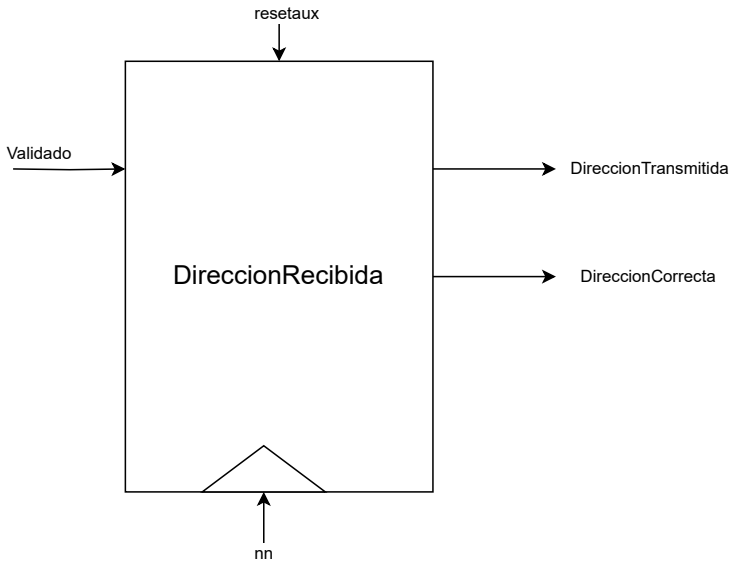


Figura 4.34: Indicador de dirección recibida.

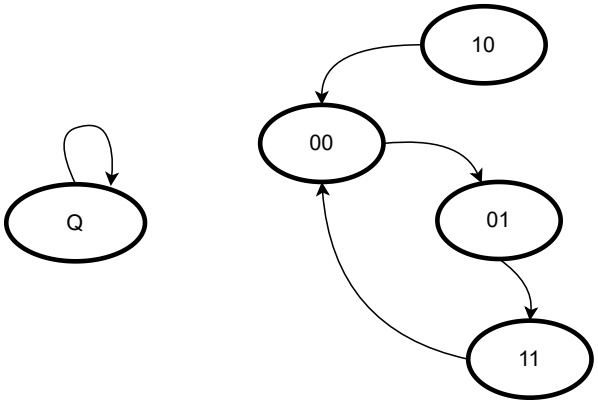


Figura 4.35: Propuesta de diagrama de estados del indicador de dirección recibida.

Estados Q	Entradas	Salidas		
	Validado	Error	DireccionTransmitida	DireccionCorrecta
00	0	0	0	0
01	0	0	0	0
11	0	0	1	0
10	0	1	0	0
00	1	0	0	0
01	1	0	0	0
11	1	0	1	1
10	1	1	0	0

Tabla 4.7: Salidas para cada estado del detector de stop.

En el diseño se usan dos FFD Keep en cascada, el primero de ellos se usa para indicar que recibió el primer pulso de *nn* proveniente del contador y que indica que la cuenta asciende a uno. El segundo se utiliza al recibir el segundo *nn* este segundo implica que la parte de la dirección ya fue transmitida puesto que el *nn* se activa primero después

de estar inactivo y después de cada ACK, por lo que se activa dos veces al recibir la dirección.

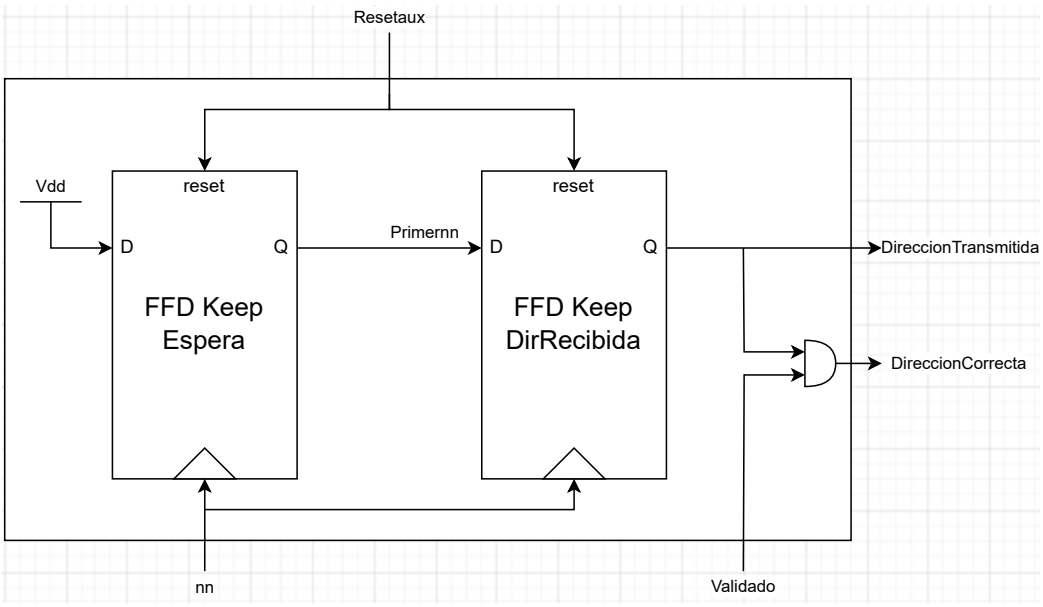


Figura 4.36: Componentes internos del indicador de dirección recibida.

ClockStretch

La funcionalidad de *estirar* el reloj se define dentro del protocolo, no es obligatorio que el esclavo lo posea pero si se desea dar tiempo al microprocesador de que pueda leer los datos recibidos o de cargar nuevos datos, es una función de bastante ayuda, aunque no todos los maestros son capaces de reconocer esta condición, por lo que se tiene que tener contemplado a la hora de activar esta opción.

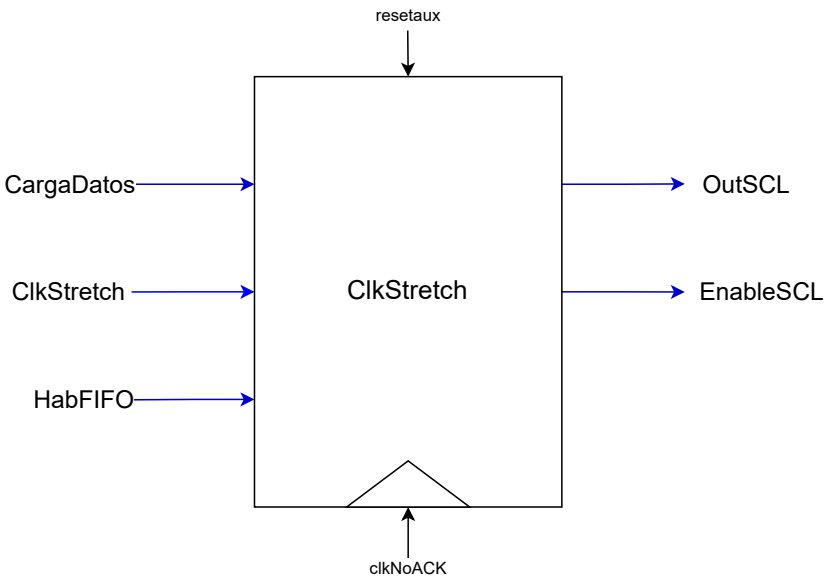


Figura 4.37: Máquina de *Clock Stretching*.

El funcionamiento consiste en que mientras exista una señal de carga de datos o esté habilitada la FIFO, no se accionará el *estiramiento*, pues estas dos señales están continuamente reseteando al FFD interno provocando que no pueda activarse. A la entrada de ese mismo FFD se tiene la señal *clkStretch* que activa el usuario en caso de que desee usar el módulo. El módulo buscará renovar su estado cada que pase el flanco de bajada del reloj *SCL* cuando es el momento del acknowledge, con la finalidad de que se active justo después de recibir el reconocimiento pero sin bloquearlo. Para desactivar el *estiramiento* del reloj, se necesita activar la FIFO o la Carga de datos, ya que si sólo se desactiva la señal de *clkStretching* esta no se actualizará internamente ya que necesita los pulsos del reloj que justo está manteniendo bajo.

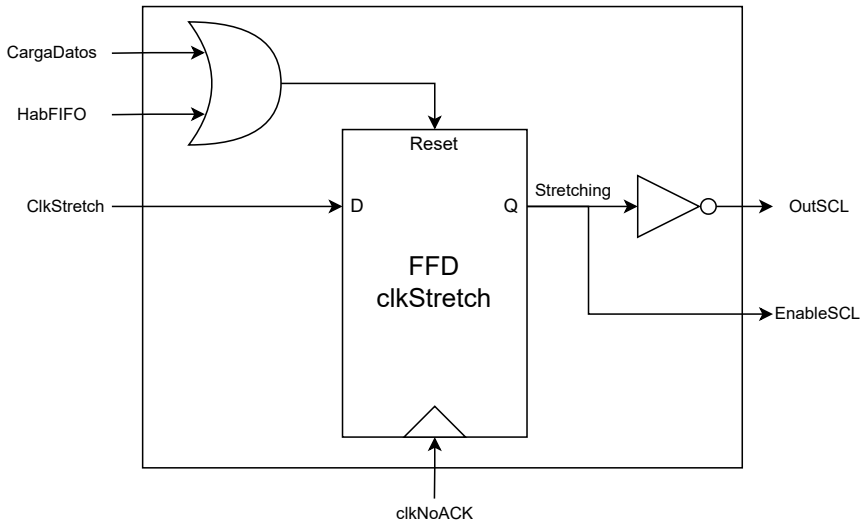


Figura 4.38: Componentes internos del módulo de *Clock Stretching*.

Controles

En este apartado se discute y analiza la necesidad de cada uno de los controles que se pueden ver en la Figura 4.22, los cuales normalmente consisten en circuitos combinacionales que, según el estado de sus señales de entrada, generan una acción casi inmediata en el sistema. La mayoría usan la señal de *Habilitado*, ya que para la mayoría de acciones tiene que estar activada la comunicación, por lo que en algunos controles no se menciona su aplicación ya que es la misma en todos.

Control de EnableSDA Este control se dedica a permitir que el esclavo pueda escribir en la línea de datos y tiene dos casos específicos: En recepción cuando necesita enviar el ACK, es decir, el momento en el que el contador habilite la señal *c*, y asegurando con la señal *validado* que la dirección coincide con la del esclavo; en el caso de la transmisión, se habilita para dar un ACK después de recibir la dirección de esclavo correcta, posterior a eso, toma el bus de datos para enviar información, y permanece tomando el bus hasta que recibe un NACK,que se ve reflejado en la señal *LiberaLinea* la indica que el maestro dejará de recibir datos y procederá a enviar una condición de stop.

Control de Comparador Este combinacional se emplea para cargar el resultado de la comparación y almacenarlo por el resto de la comunicación. Esto lo hace en el momento del ACK porque el registro serial está cargado con todos los datos de dirección y el bit de escritura/lectura, para esto hace uso de la señal *c* del contador, además de la señal *Habilitado* y para asegurarse de que este proceso no se haga después de la recepción de la dirección, usa la señal de *DireccionTransmitida* con el fin de que una trama de datos no pueda ser confundida con la de dirección.

Este control es distinto a los demás porque involucra un filtro por medio de un FFD. Esto se hizo debido a que en la fase de pruebas al inicio de la comunicación se daba un transitorio provocado por retrasos en el contador y la señal de habilitado, lo que cargaba un valor erróneo al Flip Flop del comparador. Este transitorio provocaba que desde antes de recibir la dirección, esta se validara y se almacenara el bit de escritura/lectura en '0'. Una vez establecido este filtro dependiente del reloj del divisor de frecuencia, se eliminó ese transitorio.

Control de Carga de Dirección El control de carga de dirección se usa para que la dirección del esclavo sólo se pueda modificar durante el estado inactivo del módulo o en el tiempo de acknowledge.

Control de Carga de Dato El control de carga de dato es empleado cuando el esclavo está configurado con el bit RS en $'0'$ para recibir datos del maestro. Con la finalidad de que los datos del registro serial se carguen al registro paralelo y no se vean afectados por el reloj de SCL como pasaría con el registro serial, lo que podría causar que no de tiempo para que sea leída la información recibida. Esta carga de datos se da en el momento de activación de la señal *c*, una vez que se haya transmitido la dirección del esclavo.

Control de Registro Serial Este control es el más complejo, ya que contiene una multiplexación según el bit de RS. El comportamiento para una recepción consiste en que el reloj habilitará la carga de datos seriales cuando llegue un flanco positivo de *SCL*, exceptuando el pulso del acknowledge, ya que a este registro no le interesa la información de acknowledge.

Para el caso de una transmisión, el registro carga datos de forma paralela y usa el reloj de *SCL* para escribir en la línea de *SDA*, para ello, sigue una lógica similar a la expuesta en el diseño del contador, del apartado 4.6.2 ya que para cambiar de un dato a otro emplea el flanco negativo del reloj con el fin de permitir que la señal no se vea afectada por los retrasos de tiempo internos y por el tiempo de respuesta de la línea.

Control de reloj de ACK El control del acknowledge se ve regido por la señal proveniente de la línea *SCL*, pero habilita únicamente el pulso que se da cuando el contador está en cero, es decir, la señal *c*. Para tener un buen funcionamiento, se complementa con la señal de *Dirección Correcta* lo que permite que sólo si la dirección enviada al inicio corresponde con la del esclavo, se lea el estado del acknowledge.

Resetaux Por último, la señal más importante de toda la máquina. El *resetaux* es una señal que se activa ya sea mediante el *reset* externo activado por el usuario o mediante

la condición de stop que recibe del maestro y se encarga de devolver todo el módulo a su estado base.

4.7. Maestro - Controlador

4.7.1. Diseño a nivel sistema

El maestro desenvuelve distintas actividades como las de generar las condiciones de inicio y paro, ser el encargado de la generación del reloj de la línea de *SCL*, y de enviar señales de *acknowledge* cuando está recibiendo datos. Para realizar estas tareas que permiten la recepción o envío de información se propone la arquitectura de la Figura 4.39

Los comandos necesarios para configurar el maestro son:

- Start: Se encarga de iniciar la comunicación mediante la condición de Start.
- Run: Se utiliza para indicar que la carga de datos y otras configuraciones están listas.
- Stop: Se usa para provocar que la trama que se envía sea la última y genera la condición de Stop.
- RS: Bit de Receive/Send; indica si será maestro transmisor o receptor.
- Habilitación de FIFO: Indica que la ruta de los datos se hará pasar por la FIFO y la información ahora se maneja a través de esta.
- Carga de datos: Señal que se manda al registro de datos para cargar los bits de datos.
- Carga de dirección: Señal que se manda al registro de dirección para cargar los bits de la dirección de esclavo con el que se quiere comunicar.

Las banderas necesarias:

- Modulo Busy: Se activa cuando el módulo está utilizando el bus.
- Fin: Indica que la comunicación por parte del módulo terminó.
- Error: Se activa por algún error en el proceso de configuración o un comportamiento erróneo del módulo.

La arquitectura del Maestro se puede dividir en dos subsistemas principales: los registros y la controladora. Como se puede ver en la Figura 4.40

Esencialmente, los registros únicamente almacenan los datos y la dirección del esclavo con el que se quiere establecer la comunicación; mientras que la controladora es quien dicta qué se escribe en las líneas de salida. La controladora está compuesta por dos unidades principales, el generador de reloj SCL, que simultáneamente genera el reloj para los registros RPS y el generador de Start/Stop, que como su nombre indica, toma

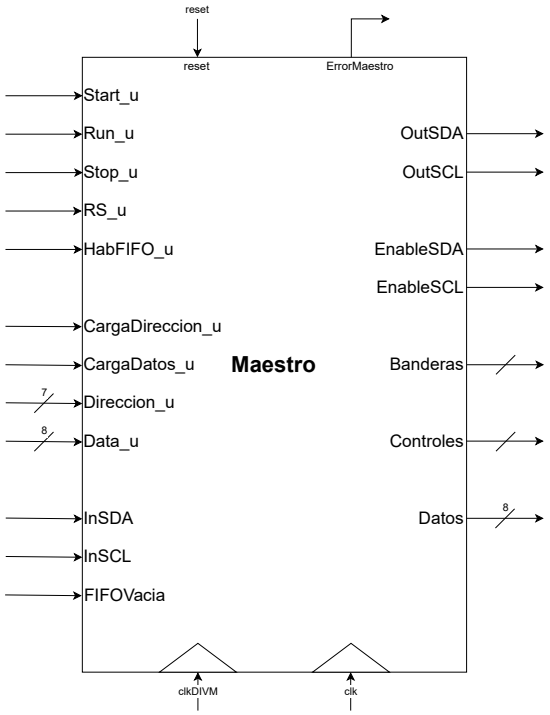


Figura 4.39: Maestro a alto nivel.

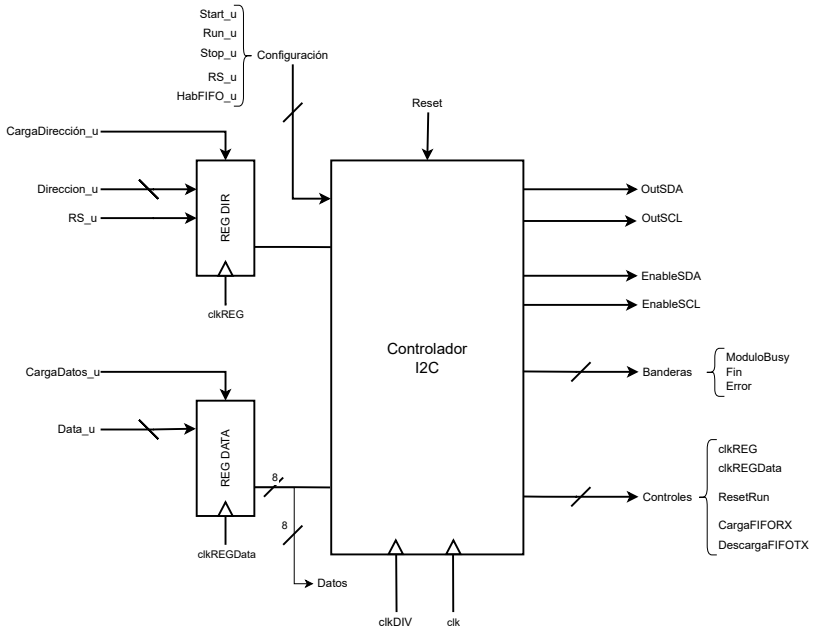


Figura 4.40: Maestro a bajo nivel.

el control del bus cuando se necesita generar la condición de Start o Stop; asimismo, cuenta con otras tres unidades auxiliares: la generadora de la señal de Stop, que saca la señal para indicar al generador de Start/Stop el momento para dar la Condición de Stop; el secuenciador que se encarga de indicar en qué momento se envía la dirección, en qué momento los datos y el acknowledge, así como la indicación de que se envió la primera trama compuesta por dirección y datos. Finalmente, el último submódulo es el que detecta el *Acknowledge*.

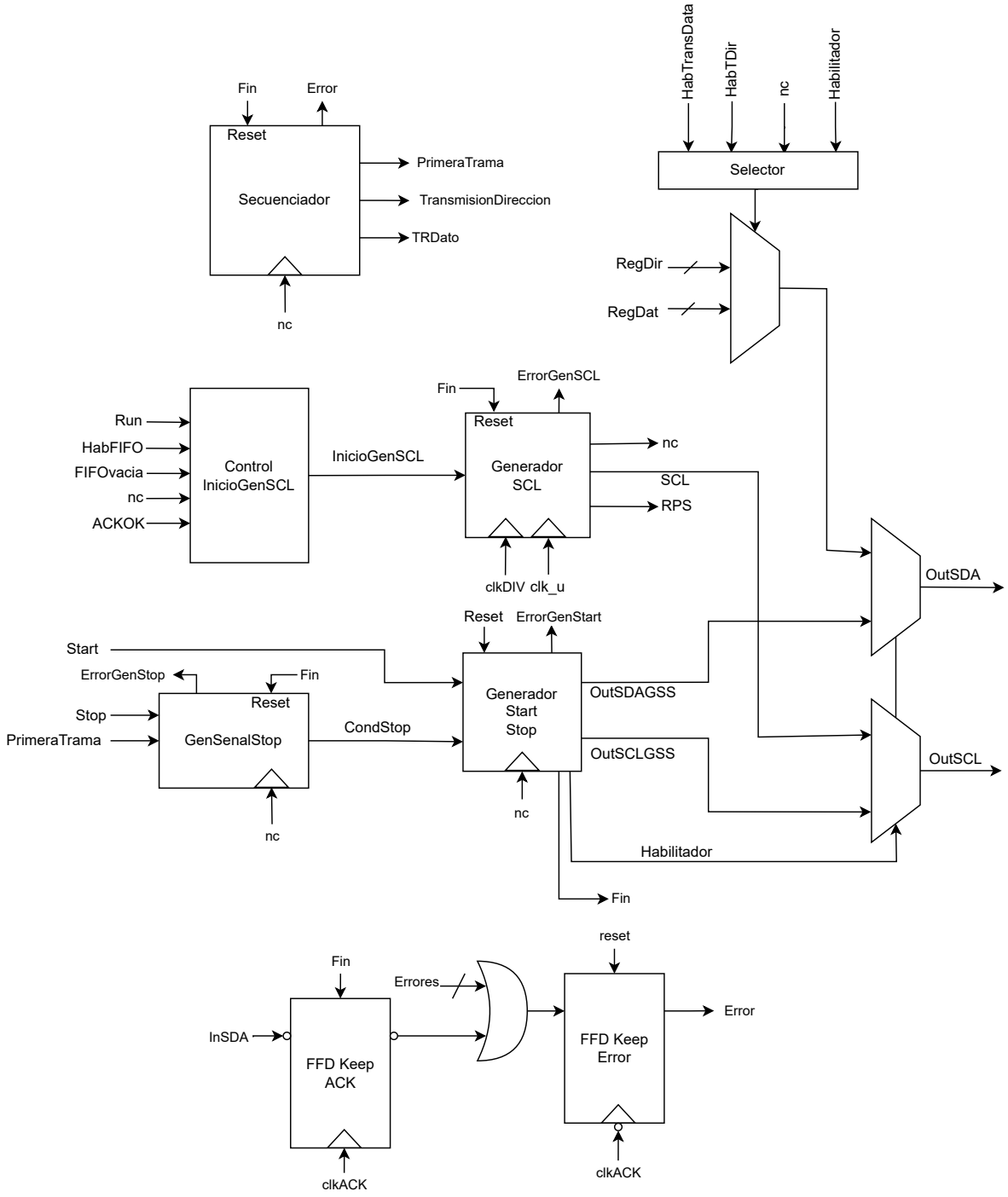


Figura 4.41: Componentes de la Controladora del Maestro.

4.7.2. Diseño a nivel detalle

A continuación, se presenta el diseño de los submódulos que componen al maestro, más específicamente los que se encuentran dentro de la controladora mostrada en la Figura 4.41 puesto que los registros mostrados en la Figura 4.40 son analizados en la Subsección 4.3.4.

Generador Start/Stop

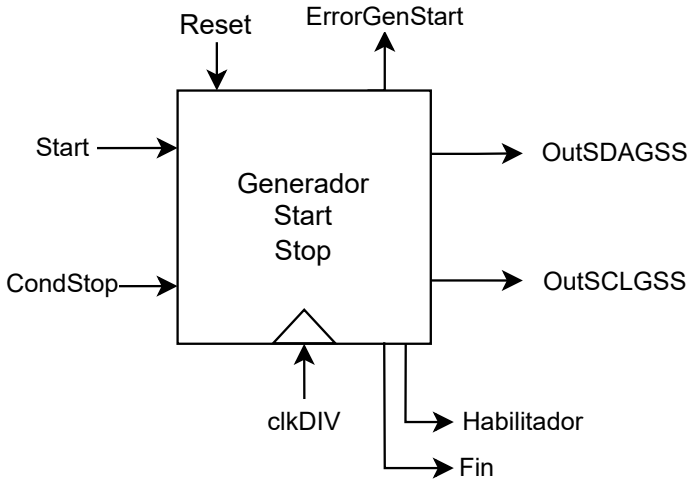


Figura 4.42: Generador de las condiciones de Start y Stop.

Como se ha venido mencionando, el generador de Start/Stop es el encargado de ejecutar las acciones que indican el inicio y fin de la comunicación, por lo que su comportamiento es como el que se muestra en la Figura 4.43. Para ello, toma en cuenta las señales de start, proveniente del registro de control y la señal de stop que es generada por el generador de la señal de stop, abordado más adelante.

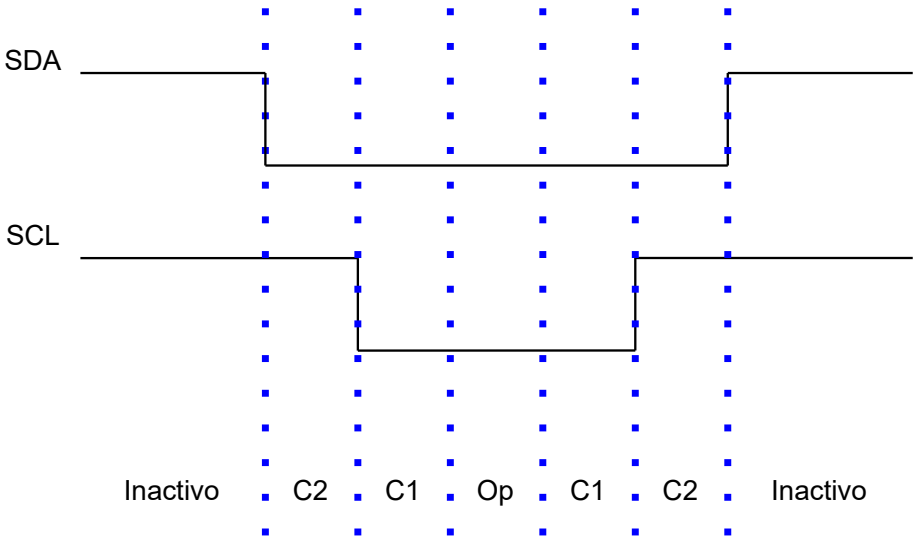


Figura 4.43: Muestra del funcionamiento de las líneas.

Para su diseño, se propuso el diagrama de estados de la Figura 4.44, en él se muestran las salidas que se contemplan para cada estado, para ello, se colocó una tabla en la Figura 4.43 que indica el estado de las salidas mostradas en dicho diagrama de estados.

Esta máquina igualmente se encarga de generar la condición de Re-Start, para ello es necesario que el usuario no active el bit de Stop y mantenga la línea de Start levantada mientras se transmite la primera trama.

Inactivo	SDA = '1' SCL = '1'
C2	SDA = '0' SCL = '1'
C1	SDA = '0' SCL = '0'
Operando	SDA = '0' SCL = '0'

Tabla 4.8: Comportamiento de las líneas de salida en cada estado para el secuenciador.

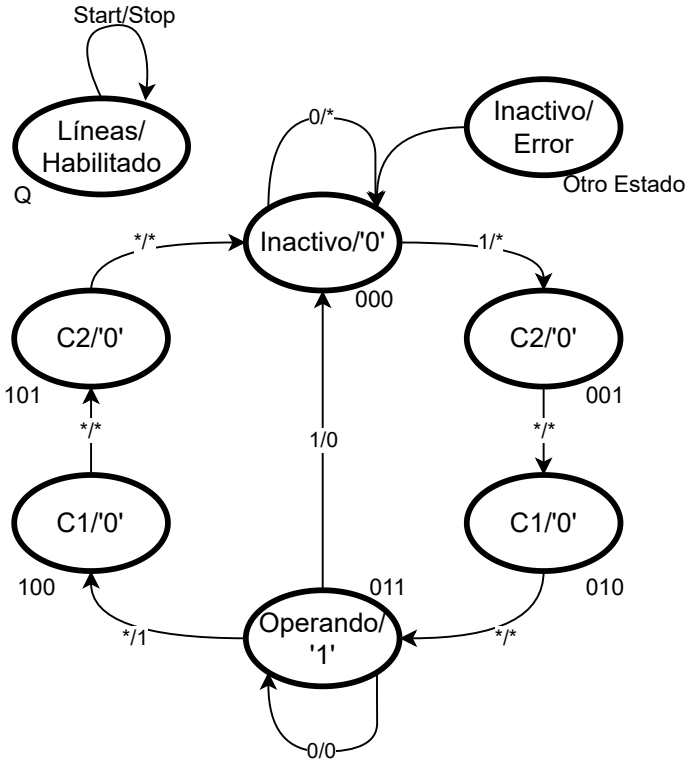


Figura 4.44: Diagrama de estados del generador de condiciones de Start y Stop.

Generador SCL

Este módulo se encarga de generar los relojes de la línea de *SCL* y el reloj *RPS* de los registros de desplazamiento. Para ello también cuenta con un contador interno de 0 a 9, que como en el contador del esclavo, sirve para sincronizar los momentos de carga de información, de acknowledge o de término de comunicación.

El diseño del contador y del generador de reloj estuvo dentro del mismo diagrama de estados, mostrado en la Figura 4.47, esto debido a que en un inicio se buscó tener módulos de cuenta y de reloj por separado, pero más adelante en las simulaciones se notó que había desplazamientos de más por la señal de *RPS* que generaba pulsos en la cuenta de uno y nueve, lo que cargaba datos sin querer. Si bien se trató de contener esta falla mediante una compuerta *AND* que contuviera dicho pulso, esto no serviría debido a los tiempos de propagación que no permitían encapsular debidamente el pulso, por lo que

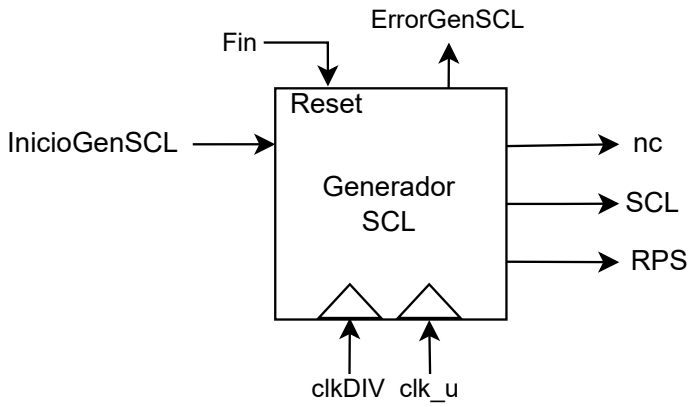


Figura 4.45: Generador del reloj de SCL.

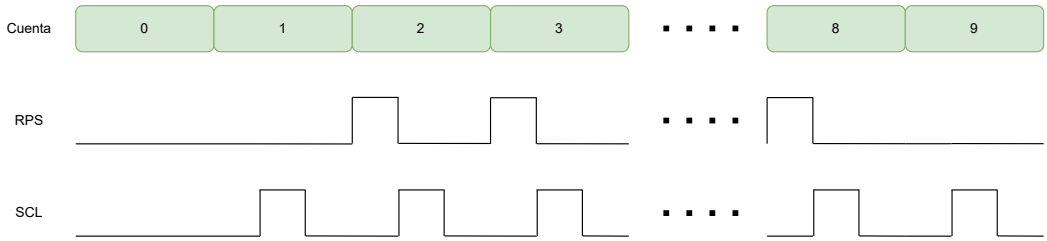


Figura 4.46: Comportamiento del reloj de SCL.

los registros seguían captando la señal de reloj.

Al mismo tiempo, con ese primer diseño se notó que existía un tiempo de ejecución muerto ya que al ser máquinas Moore, la señal de inicio del módulo de relojes tomaba todo un ciclo de reloj después de la señal de inicio para comenzar a actuar. Con el fin de corregir estos errores y de hacer más fácil el rediseño, se optó por una sola máquina que realice ambas acciones de cuenta y generación de reloj; así en caso de querer cambiar el valor de la señal de reloj en un momento que no se necesite, simplemente se modificaba el valor del mintérmino en la tabla de estados y no toda la máquina, ya que la cantidad de estados iba a ser la misma, caso contrario de querer realizar esta modificación en un sólo módulo, lo que exigía elevar la cantidad de estados del bloque de relojes. El comportamiento deseado se muestra en la Figura 4.46

Durante la fase de pruebas, se detectaron varios glitches, por lo que al tratarse de líneas de reloj esto se volvió un error crítico. Para solucionar esto, se decidió implementar FFD que operen a la frecuencia del reloj base, con tal de que permitieran filtrar las señales de SCL y RPS para que lleguen *limpias* a las entradas de los registros y la línea.

Cuando se realizó el diseño del divisor de frecuencia, se indicó que este no daba directamente la tasa de transmisión que indica el protocolo *I2C*, esto se debe a que como este generador de SCL trabaja con la frecuencia dividida tenemos que por cada 3 estados, vamos a tener un pulso de *SCL* por lo tanto, si en el divisor de frecuencia disponemos de un reloj a la salida de *3[MHz]*, vamos a obtener una velocidad de transmisión *real* de *1[MHz]*.

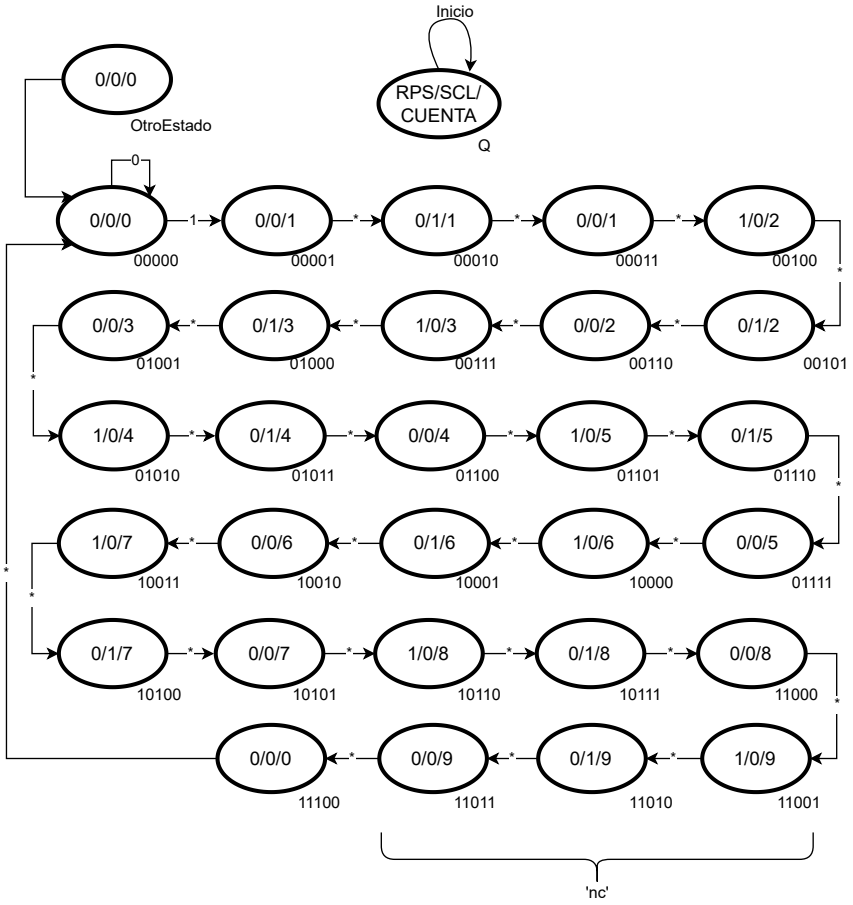


Figura 4.47: Diagrama de estados del generador de SCL.

Generador de la señal de Stop

Este módulo tiene su propósito en que debe dar la indicación de en qué momento se generará la señal de stop que llega al generador de la condición de stop, para esto tiene como entradas la señal de *Stop* original que proviene de los registros de control y la señal de *Primera Trama* que indica si ya se envió o recibió el primer dato. Además de que el reloj será manejado por el flanco negativo de la señal *nc* para que el estado de esta máquina se actualice después del momento de *acknowledge*.

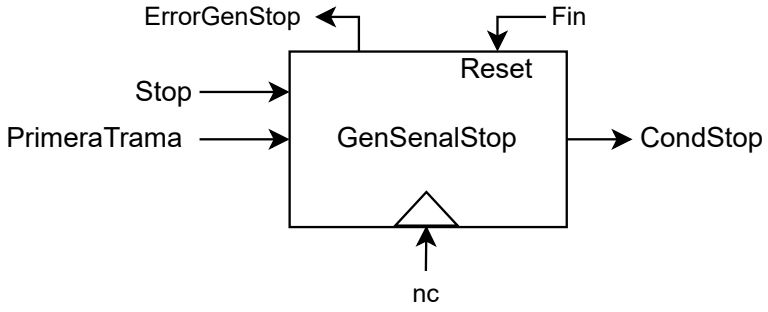


Figura 4.48: Generador de la señal de *Stop*.

Con las señales previamente mencionadas se generó el diagrama de estados mostrado

en la Figura 4.49, el cual restringe la generación de la señal de stop hasta para después de la primera trama, después de esto, si el usuario da la señal externa de stop, esta máquina generará la señal de stop interna hasta que sea el momento, es decir, después del tiempo de acknowledge. Una vez que la máquina se encuentra en el estado $'1'$ este mismo estado indica la señal de *Stop* que será enviada al generador de la condición de *Stop*

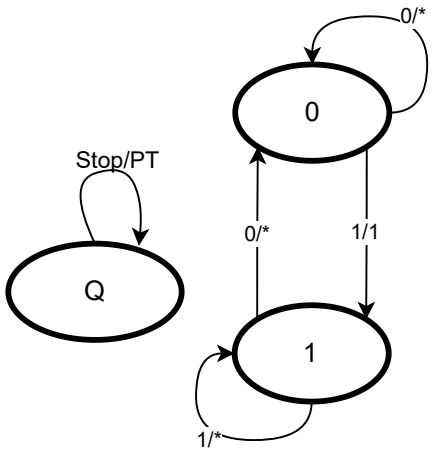


Figura 4.49: Diagrama de estados del generador de la señal de Stop.

Secuenciador

El secuenciador es el encargado de darle la estructura a las tramas. Es decir, tiene la tarea de indicar en qué momento se envía la dirección de esclavo (*TransmisionDireccion*) y cuándo se reciben o envían los datos (*TRDato*). También tiene una señal dedicada a indicar cuando ya fue enviada la primera trama compuesta de dirección y datos (*PrimeraTrama*).

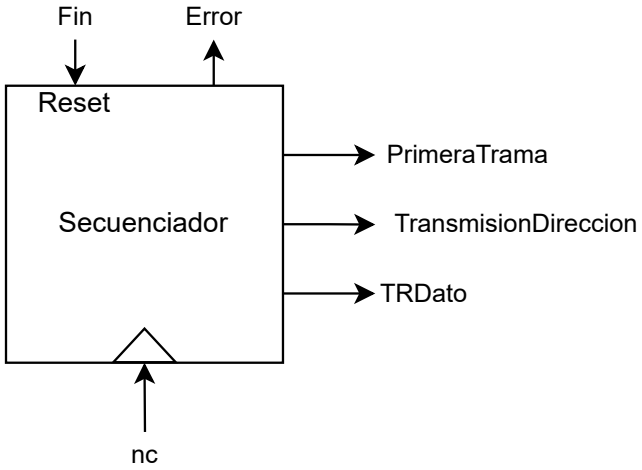


Figura 4.50: Secuenciador.

Para su diseño se contempló como única entrada a la señal de control *nc*, para este diseño se trabaja con el flanco negativo de la señal, lo que en esencia permite tener un contador de dos bits, con cuenta de cero a dos, como se muestra en el diagrama de estados

de la Figura 4.51, cada una de las cuentas activa distintas salidas, que son mostradas en la Tabla 4.9

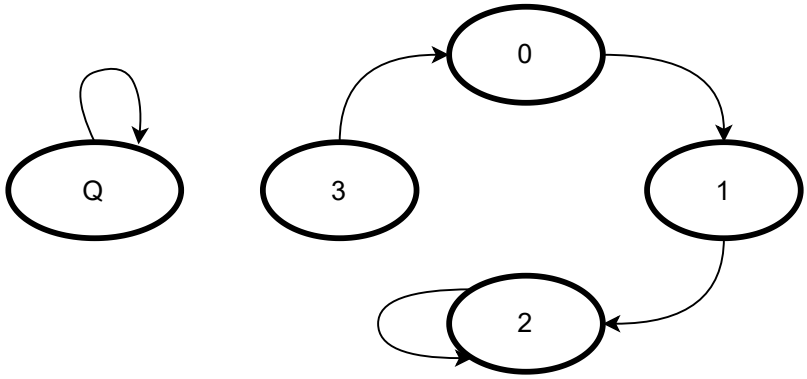


Figura 4.51: Diagrama de estados del secuenciador.

Estados Q	Salidas			
	PrimeraTrama	TransmisionDirección	TRDato	Error
0	0	1	0	0
1	0	0	1	0
2	1	0	1	0
3	0	0	0	1

Tabla 4.9: Salidas para cada estado del secuenciador.

Registro de Acknowledge

Este registro tiene la función de obtener el valor de la línea de datos *SDA* justo en el momento que se da el pulso de reloj *SCL* después de fin de trama.

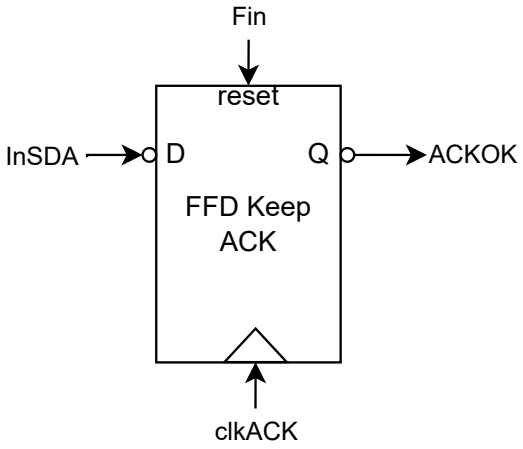


Figura 4.52: Registro de acknowledge.

Este registro está activo en todo momento, es decir, ya sea que el maestro reciba o transmita datos. Esta decisión se tomó ya que el registro de errores depende de este, por

lo que aún en el caso de que el Maestro sea quien envíe el acknowledge es igualmente necesario saber qué valor se está imprimiendo en la línea al momento del acknowledge, entendiéndose de que de no presentarse un acknowledge en el momento se estaría indicando un error.

Registro de Errores

Por último, tenemos al registro que se encarga de almacenar si existe algún error dentro de los módulos previamente descritos.

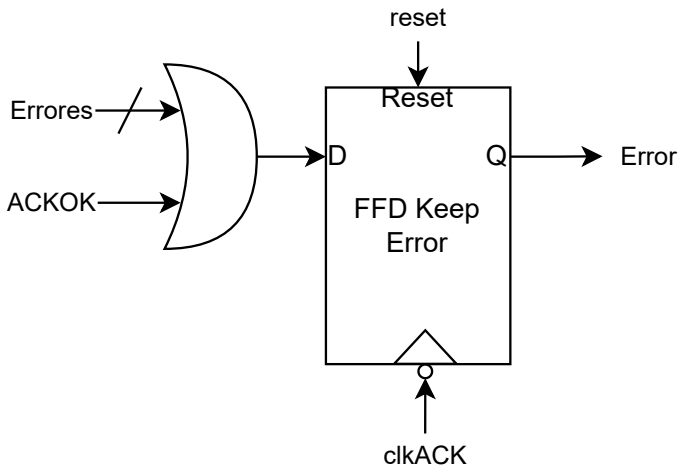


Figura 4.53: Registro de Errores.

Para ello, consta de una compuerta *OR* con una línea dedicada para las señales de error proveniente de cada módulo interno del maestro, más precisamente de los componentes de la controladora. Su valor interno se actualiza con el flanco de bajada de la línea de *SCL* después del momento de cada acknowledge, por lo que se actualiza su valor instantes después de conocer el estado del acknowledge.

4.8. Implementación de tolerancia a fallas

Para la implementación de mecanismos de tolerancia a fallas, se propuso un esquema de Redundancia Modular Triple, ya que este es un mecanismo preferentemente utilizado para misiones de corto periodo de duración [7], además de que también es empleado para aumentar la tolerancia a fallas transitorias causadas por radiación [22].

Como consideración inicial, se tiene tomar en cuenta que los métodos de redundancia modular obtienen parámetros de confiabilidad mayores si se implementan al más bajo nivel que si se hace a un nivel mayor [7]. Es por ello, que se tomó la decisión de reforzar los elementos más críticos a bajo nivel, considerando a los Flip Flop como dichos elementos, ya que estos además de ser los que almacenan la información generada a partir de la comunicación del sistema, son la base de los estados de las máquinas internas, por lo que resultan ser más relevantes dentro del sistema que está primordialmente construido con máquinas de estado finito tipo Moore, por lo que al solo depender las salidas del estado presente las fallas transitorias son menos importantes para ser tomadas en cuenta.

Interesándonos más por fallas del tipo *SEU* que pueden cambiar el valor de los bits internos de los dispositivos que almacenan memoria.

Igualmente, sería necesario realizar procesos que ayuden a discernir y obtener los puntos más débiles del sistema, pero para ello se necesitan procesos de inyección de fallas que exceden los alcances de este trabajo.

4.8.1. Diseño de un FFD con TMR

Una vez discutida la propuesta, se genera el diseño necesario para trabajar con el esquema de Redundancia Modular Triple sobre el hardware previamente diseñado. Para ello, se busca sustituir cada uno de los FFD, mostrados anteriormente en la Subsección 4.3.1, en cada una de las máquinas por un módulo con tres FFD trabajando en paralelo y con una etapa de dictaminación que escoja la salida que compartan la mayoría de Flip Flops. La arquitectura del módulo quedaría como la que se muestra en la Figura 4.54

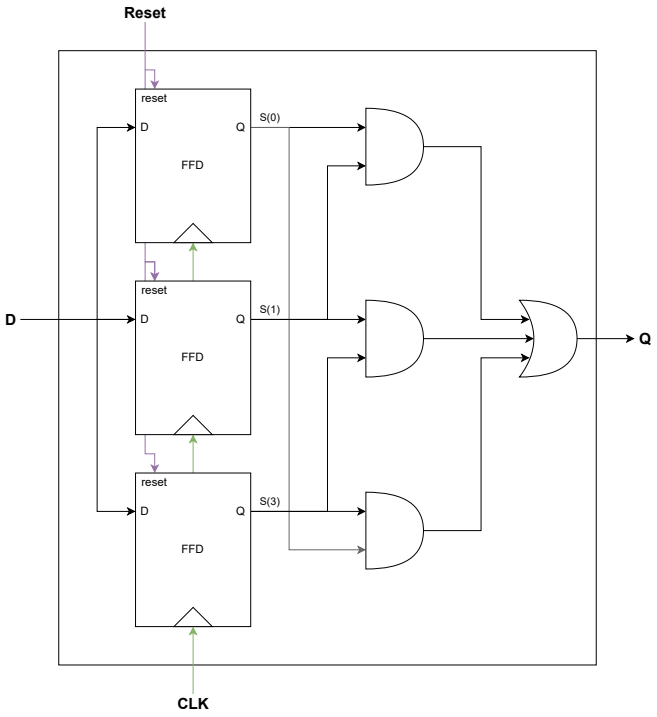


Figura 4.54: Implementación de un FFD con tolerancia a fallas.

La generación del votador mayoritario o *Dictaminador* se obtuvo a partir de la Tabla 4.10.

Entradas			Salidas
S0	S1	S2	Elección
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Tabla 4.10: Tabla de verdad para el votador mayoritario.

Si analizamos las entradas y salidas del sistema con *TMR*, son las mismas en número y características que las de un FFD normal, por lo que la sustitución de un FFD sencillo por uno con arquitectura *TMR* es directa dentro de la descripción, simplificando todo el proceso de rediseño.

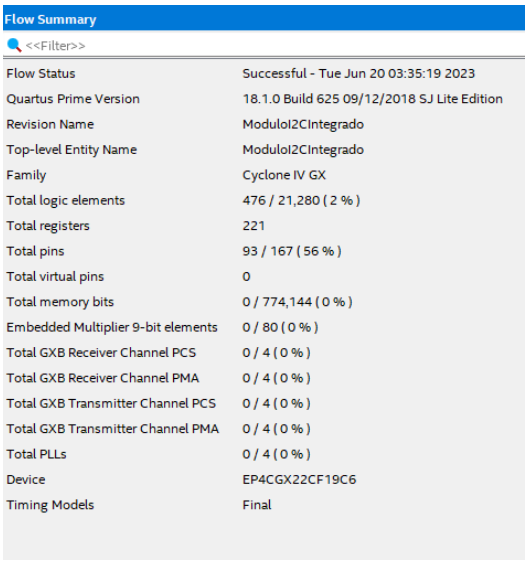
Este proceso de implementación de un esquema de redundancia modular triple, es igualmente fácil de extrapolar a cualquier grado de redundancia pasiva, conservando la misma cantidad de entradas y salidas con las mismas características. Por lo tanto, resulta fácil generalizar este concepto con arquitecturas de redundancia modular de N elementos.

Capítulo 5

Resultados

5.1. Reporte de recursos lógicos

A través de la plataforma de *Quartus* se obtuvieron los reportes de los recursos empleados para la generación del hardware, tanto para la versión sin tolerancia a fallas como para la versión con la implementación de *TMR*.

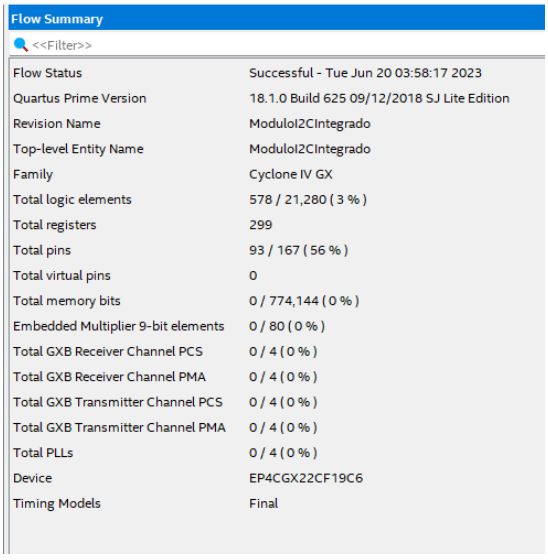


Flow Summary

<<Filter>>

Flow Status	Successful - Tue Jun 20 03:35:19 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	ModuloI2CIntegrado
Top-level Entity Name	ModuloI2CIntegrado
Family	Cyclone IV GX
Total logic elements	476 / 21,280 (2 %)
Total registers	221
Total pins	93 / 167 (56 %)
Total virtual pins	0
Total memory bits	0 / 774,144 (0 %)
Embedded Multiplier 9-bit elements	0 / 80 (0 %)
Total GXB Receiver Channel PCS	0 / 4 (0 %)
Total GXB Receiver Channel PMA	0 / 4 (0 %)
Total GXB Transmitter Channel PCS	0 / 4 (0 %)
Total GXB Transmitter Channel PMA	0 / 4 (0 %)
Total PLLs	0 / 4 (0 %)
Device	EP4CGX22CF19C6
Timing Models	Final

Figura 5.1: Reporte de recursos lógicos utilizados por el módulo sin aplicar TMR.



Flow Summary

<<Filter>>

Flow Status	Successful - Tue Jun 20 03:58:17 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	ModuloI2CIntegrado
Top-level Entity Name	ModuloI2CIntegrado
Family	Cyclone IV GX
Total logic elements	578 / 21,280 (3 %)
Total registers	299
Total pins	93 / 167 (56 %)
Total virtual pins	0
Total memory bits	0 / 774,144 (0 %)
Embedded Multiplier 9-bit elements	0 / 80 (0 %)
Total GXB Receiver Channel PCS	0 / 4 (0 %)
Total GXB Receiver Channel PMA	0 / 4 (0 %)
Total GXB Transmitter Channel PCS	0 / 4 (0 %)
Total GXB Transmitter Channel PMA	0 / 4 (0 %)
Total PLLs	0 / 4 (0 %)
Device	EP4CGX22CF19C6
Timing Models	Final

Figura 5.2: Reporte de recursos lógicos utilizados por el módulo aplicando TMR con optimización.

Se observa que la cantidad de recursos lógicos no es proporcional a la cantidad de elementos lógicos que se agregaron por la implementación de *TMR*. Esto es debido a que el sintetizador busca optimizar la cantidad de elementos lógicos y detecta la redundancia como elementos repetidos, por lo que fusiona la mayoría de los Flip Flops con redundancia triple en uno. Para solucionar esto en Quartus, es necesario ir a los ajustes de *Analysis &*

Synthesis, dentro de esta sección se buscan los ajustes avanzados del sintetizador y en la nueva ventana que se abre, buscamos la opción de *"Allow Register Mergin"* y si aparece en *"on"* la cambiamos a *"off"*.

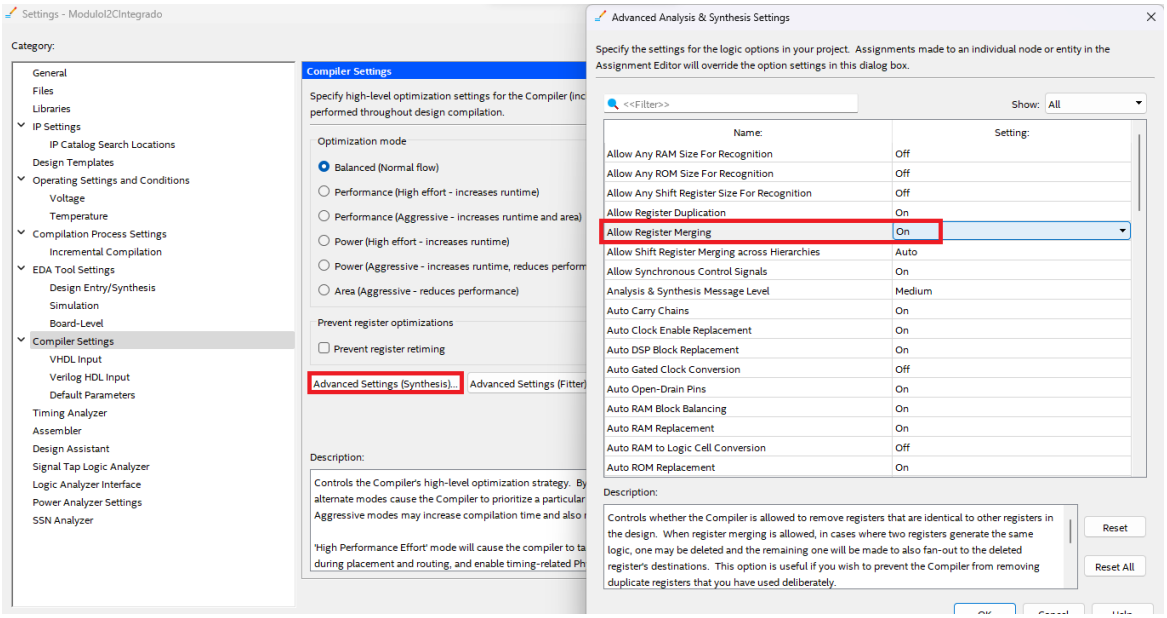


Figura 5.3: Ventanas de configuración del sintetizador.

Con los ajustes anteriores, se obtiene que ahora la cantidad de registros utilizados empata perfectamente con lo esperado, es decir, en la Figura 5.1 se observa que el número de registros empleados es de 221; mientras que en la Figura 5.4 referente al reporte de recursos lógicos empleados, aunque esta vez sin la optimización por parte del sintetizador en los registros, en ella se puede observar que el número asciende a 663, que corresponde con lo esperado de un aumento de tres veces la cantidad de registros.

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Jul 26 12:53:28 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	ModuloI2CIntegrado
Top-level Entity Name	ModuloI2CIntegrado
Family	Cyclone IV GX
Total logic elements	1,235 / 21,280 (6 %)
Total registers	663
Total pins	93 / 167 (56 %)
Total virtual pins	0
Total memory bits	0 / 774,144 (0 %)
Embedded Multiplier 9-bit elements	0 / 80 (0 %)
Total GXB Receiver Channel PCS	0 / 4 (0 %)
Total GXB Receiver Channel PMA	0 / 4 (0 %)
Total GXB Transmitter Channel PCS	0 / 4 (0 %)
Total GXB Transmitter Channel PMA	0 / 4 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 5.4: Reporte de recursos lógicos utilizados por el módulo aplicando TMR sin optimización.

5.2. Equipo

Para la fase de pruebas se empleó un analizador de estados lógicos, este monitorizaba las líneas SDA y SCL, así como algunas señales internas de la máquina que ayudaron a observar el correcto funcionamiento de cada una de las partes. Así mismo, se empleó la tarjeta de desarrollo Basys 3, conectada a un bus simple montado sobre protoboard. Para completar la comunicación, se utilizó una tarjeta de desarrollo Tiva EK-TM4C1294XL.

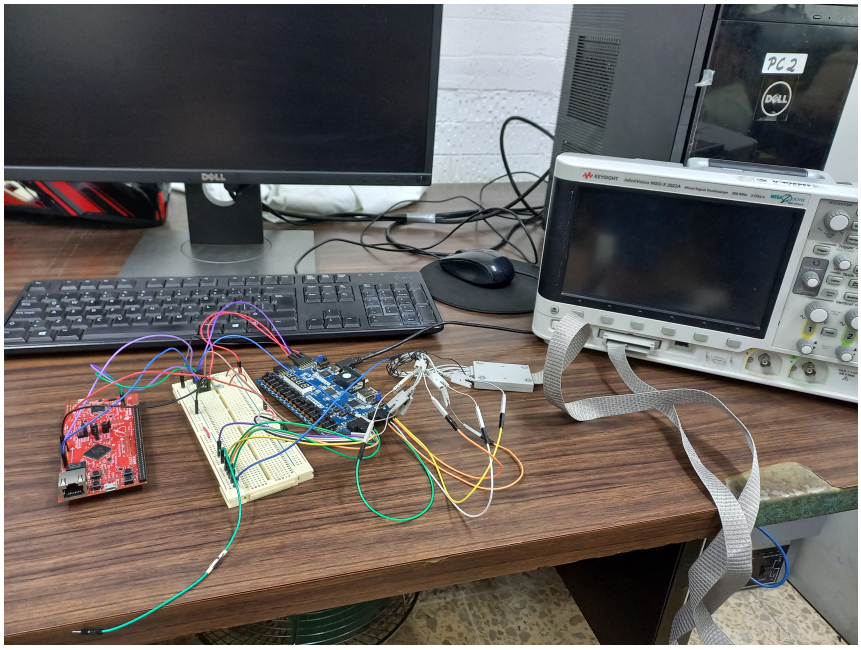


Figura 5.5: Muestra del ambiente de trabajo.

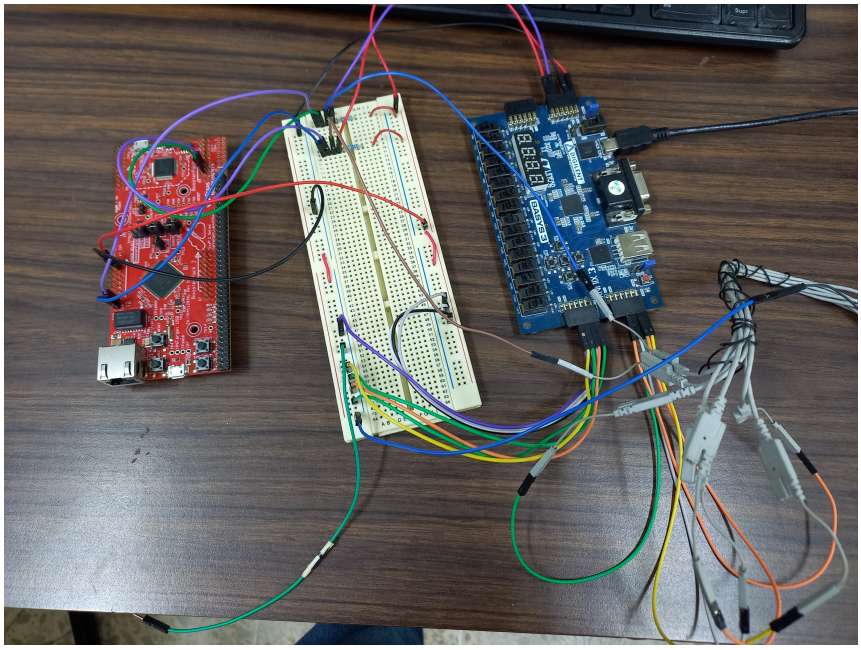


Figura 5.6: Conexiones al bus entre Basys 3 y la Tiva.

5.3. Comunicación con módulo externo

5.3.1. Maestro-Basys 3 con Esclavo-TIVA

Maestro Transmisor

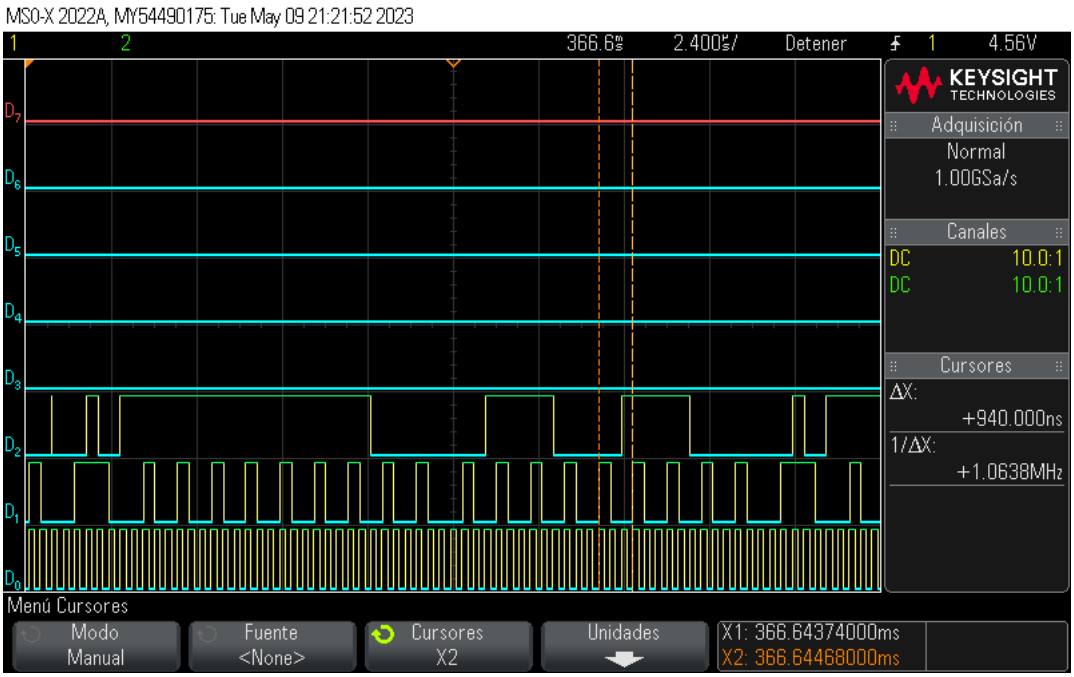


Figura 5.7: Trama de envío de un dato generada por el maestro a 1$[Mbit/s]$.

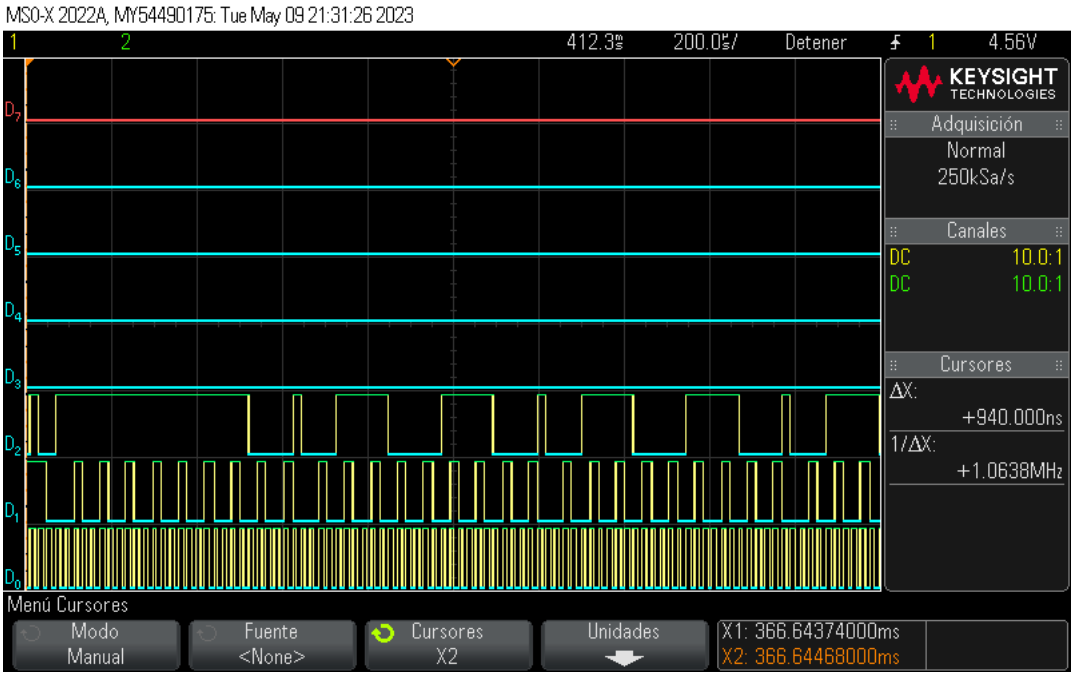


Figura 5.8: Inicio de trama de envío de varios datos generados por el maestro a 1$[Mbit/s]$.

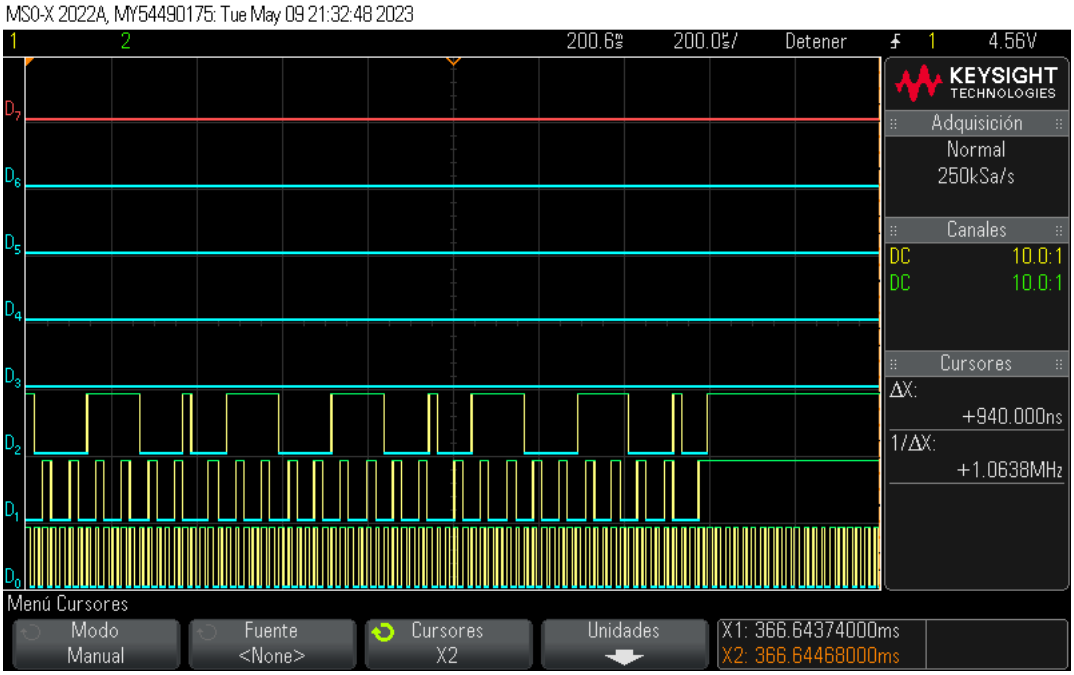


Figura 5.9: Fin de trama de envío de varios datos generados por el maestro a 1$[Mbit/s]$.

Para ambos casos de envío de un dato y varias tramas, se puede observar la estabilidad y sincronicidad de las señales que se imprimen en el bus por parte del maestro, esto a velocidades de 1$[MHz]$ que es la más rápida para los objetivos planteados. Se verifica la correcta generación de condiciones de inicio y parada.

Maestro Receptor

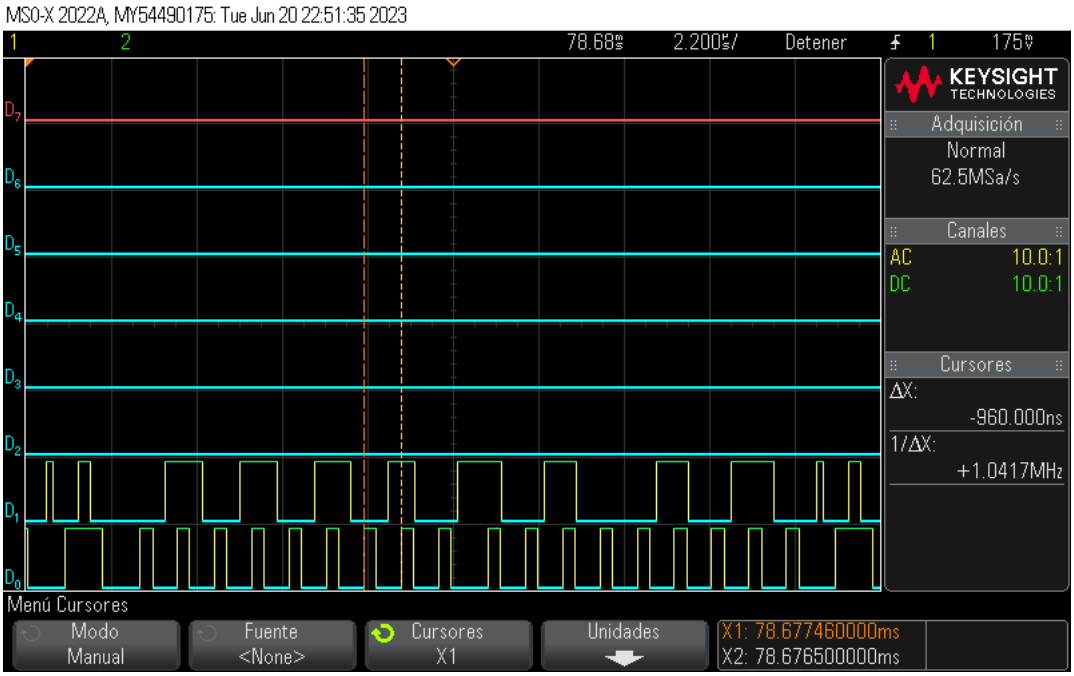


Figura 5.10: Trama de recepción de un dato generada por el maestro a 1$[Mbit/s]$.

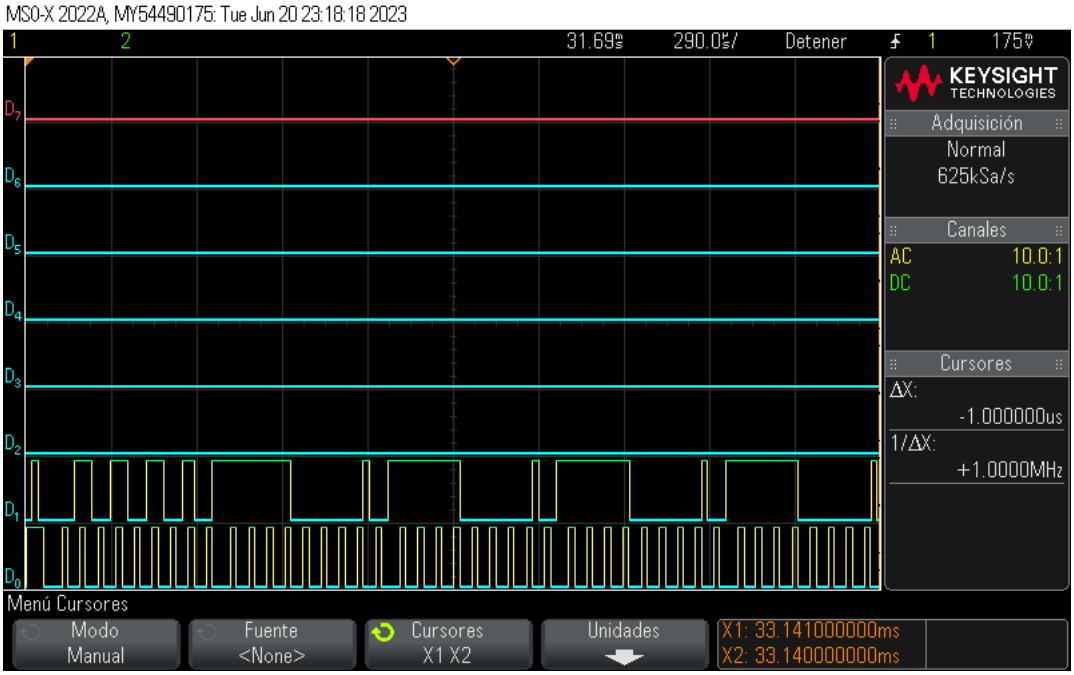


Figura 5.11: Inicio de trama de recepción de un dato generada por el maestro.

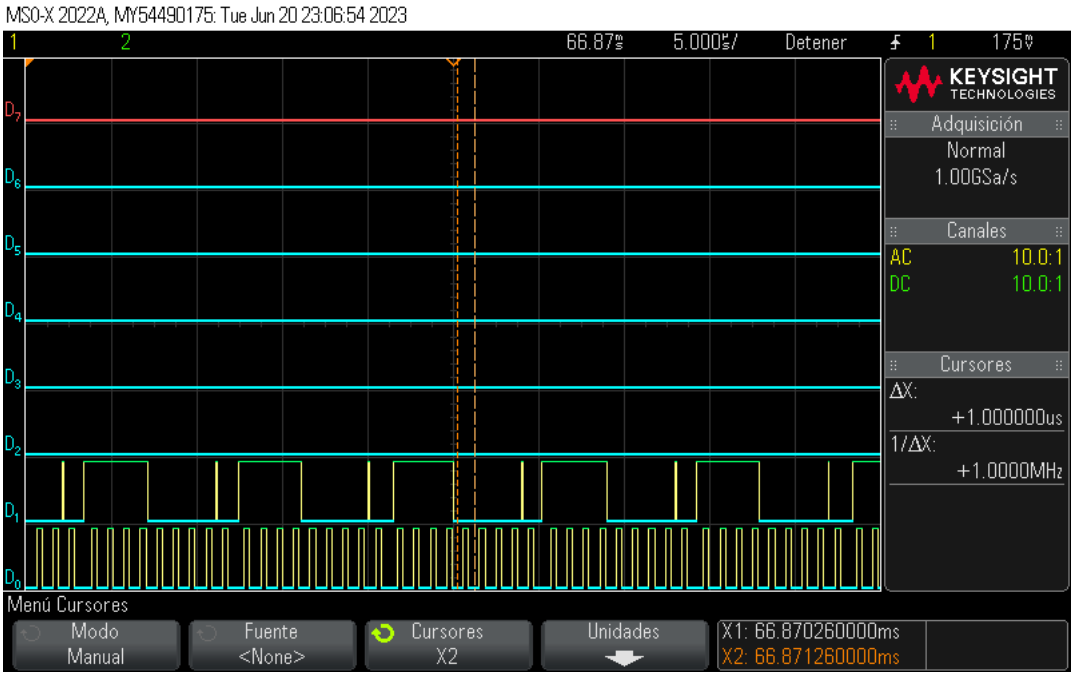


Figura 5.12: Trama de recepción de varios datos generada por el maestro a 1$[Mbit/s]$.

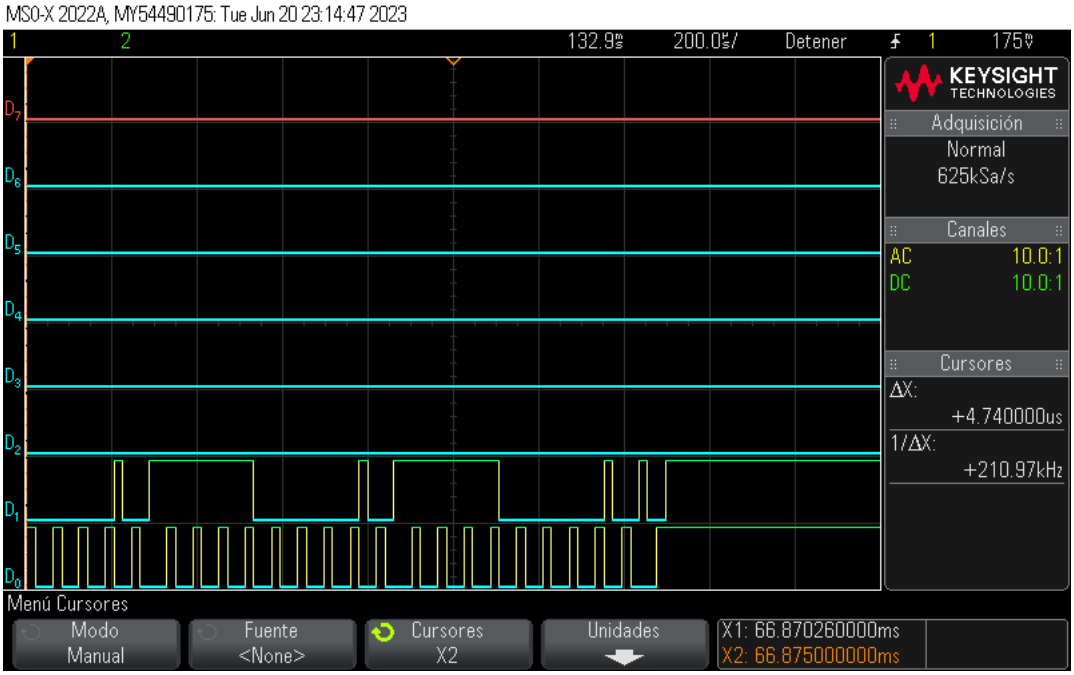


Figura 5.13: Final de trama de recepción de un dato generada por el maestro.

En el caso del maestro trabajando como receptor, se observa el mismo comportamiento estable sin glitches, con generaciones correctas de Start y Stop. Además de que da el acknowledge en el momento correcto.

5.3.2. Esclavo-Basys 3 con Maestro-TIVA

Esclavo Transmisor

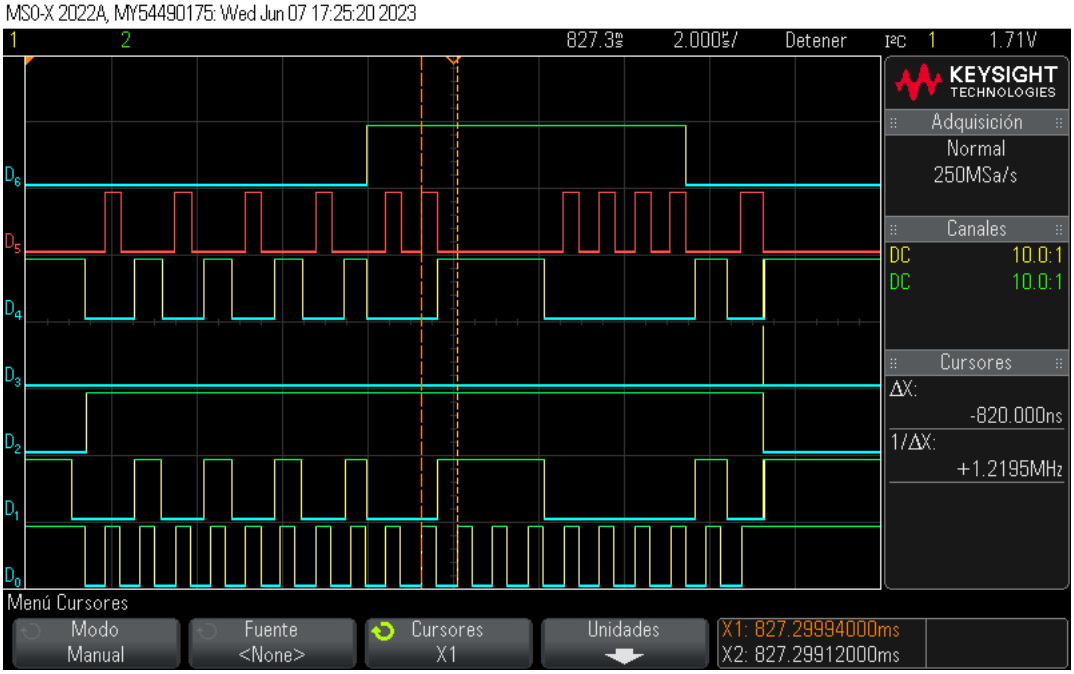


Figura 5.14: Trama de un dato generada a 1$[Mbit/s]$.

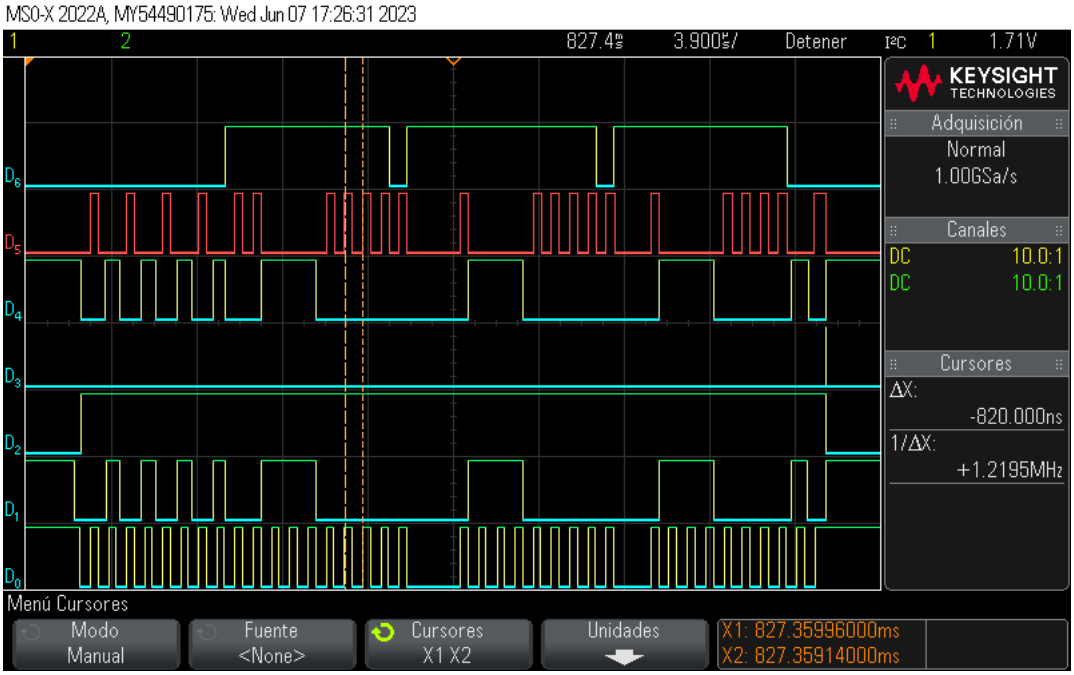


Figura 5.15: Trama de un dato generada a 1$[Mbit/s]$.

Se puede observar que da el acknowledge en el momento de la recepción de dirección y procede a escribir los datos en la línea de SDA, cambiándolos en el flanco negativo.

Esclavo Receptor

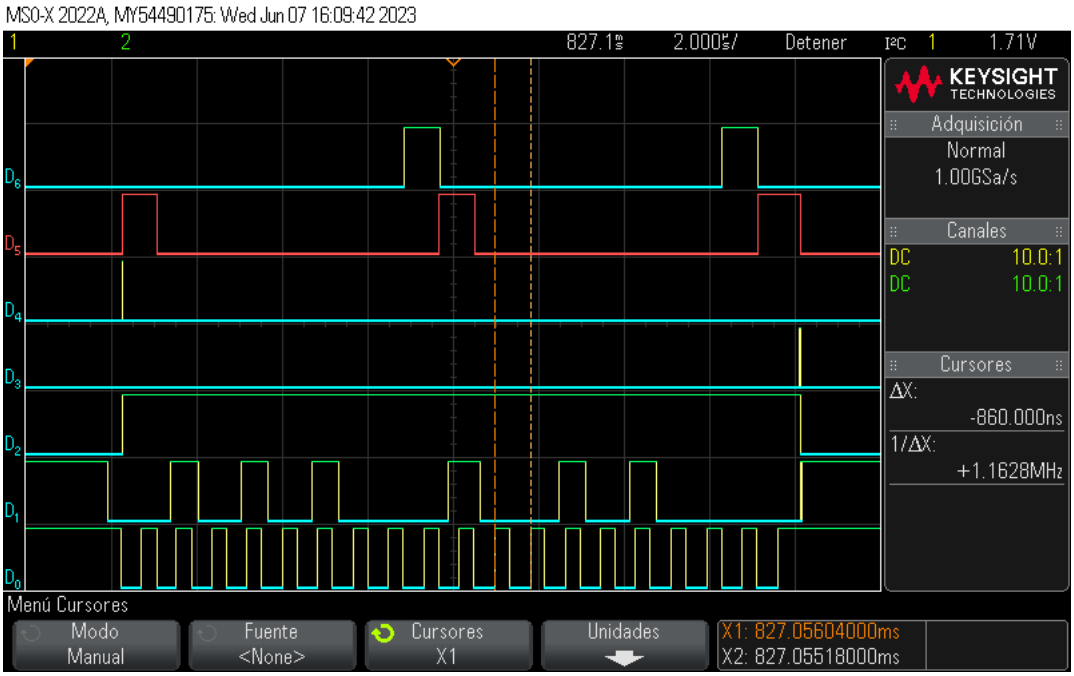


Figura 5.16: Recepción de un dato a 1$[Mbit/s]$.

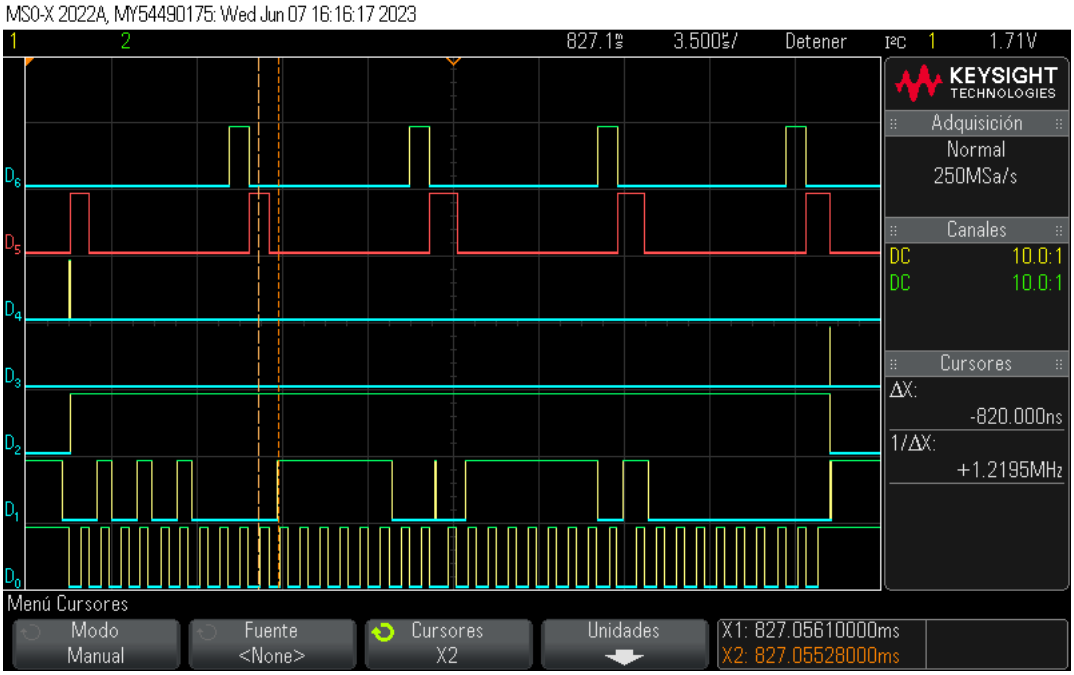


Figura 5.17: Recepción de varios datos a 1$[Mbit/s]$.

Se puede ver que en este modo el esclavo responde correctamente con el acknowledge en el pulso indicado. Se observa que no hay problemas de sincronización debidos a retrasos.

Capítulo 6

Conclusiones

En el presente trabajo se realizó el diseño e implementación en *FPGA* de un módulo de comunicación I2C capaz de funcionar como periférico de un microprocesador. El módulo desarrollado logró establecer comunicación con un microprocesador externo, de forma que se pudo recibir y enviar información probándose a distintas velocidades dentro del rango *F/S* (Revisar Tabla 2.2) obteniendo una respuesta correcta aún con la tasa de transmisión más rápida contemplada en los objetivos (*Fast-Plus*), como se puede ver en el Capítulo 5 de resultados.

Para su desarrollo primeramente se consideraron distintas arquitecturas, de la que resultó más funcional la propuesta de un esquema similar al manejado por *Texas Instruments* debido a su enfoque en la modularidad, es decir, que cada bloque interno se centra en una tarea específica como el almacenamiento temporal que es asignado a la FIFO o la formación de las tramas que es producto del núcleo maestro, por poner ejemplos.

Desde su concepción se buscó utilizar la menor cantidad de recursos lógicos mediante un diseño a bajo nivel, por lo que al revisar el reporte de sintetización del Capítulo 5 se observa por la cantidad de registros y elementos lógicos que se logró obtener un ahorro de componentes considerable comparado con lo que hoy se encuentra en módulos comerciales vistos en el Capítulo 3. Por supuesto que esta comparación no puede ser del todo directa, puesto que algunas de las *IPs* vistas contienen funciones que no se incluyen dentro de este trabajo, pero este cotejo sirve para dar una idea del ahorro de recursos acorde a la funcionalidad obtenida.

A través de la construcción de este sistema digital se ahondó en temas relacionados con la construcción de máquinas de estado finito, donde se abordaron problemáticas como la selección de sistemas síncronos y asíncronos, considerando los primeros para las aplicaciones *críticas* en los que se necesita conocer cada uno de los estados de la máquina y que tengan la capacidad de restablecerse a su estado base en caso de errores como cambios a estados no permitidos, lo que mejora el desempeño en términos de la estabilidad que brindan, con la desventaja de que en la mayoría de casos estas máquinas emplean más elementos lógicos. Para los sistemas asíncronos, se les consideró en máquinas donde sus estados no sean *críticos* para la operación, como en un divisor es decir, en casos donde si existe un error en el estado presente este no influya en la operación de los demás sistemas, pues en el ejemplo del divisor, si se llegase a cambiar a un estado *erróneo* únicamente afectaría al adelantar o atrasar una señal de reloj, esto sólo modificaría brevemente la operación *normal* y correcta del sistema sin alterar su rutina. Igualmente este tipo de sistemas asíncronos normalmente conllevan una menor cantidad de elementos lógicos que un divisor o contador síncrono, lo que puede ayudar a obtener un menor efectos de *jitter*.

Asimismo, se buscaron más formas de obtener señales lo más estable posibles, donde encontré técnicas como la implementación de máquinas sin circuitos combinacionales a la salida, lo que permite evitar glitches por los tiempos de propagación debidos a las compuertas, aunque hacer este procedimiento conlleva una mayor de cantidad de Flip Flops básicamente necesitando uno por cada bit de salida en el que se quiera mantener una señal lo más estable posible. Igualmente, se encontró que a través de codificaciones como el código *Gray* se puede ayudar a obtener menores transitorios, a través de hacer que las transiciones entre un estado y otro sean menos *bruscas* lo que se logra con este código es que sólo cambie un bit de un estado a otro, evitando pasar por varios estados intermedios en los cuales se pueden activar momentáneamente ciertas señales *parásitas* en la etapa de transición , que para el caso de señales de reset, resulta crítico ya que no dependen de una señal de reloj y pueden ser activadas en cualquier momento por un

transitorio.

Además, se tomo en cuenta el tipo de *FSM* para el tipo de fallas que más nos interesan, ya que, por ejemplo, una máquina tipo Mealy es más susceptible a fallas del tipo transitorio, al depender sus salidas del estado presente y de las entradas que pueden cambiar en cualquier momento. Por ello, se tomó la decisión de usar primordialmente máquinas tipo Moore, para que en caso de un fallo transitorio en las entradas este no afecte a las salidas mostradas a las demás partes del sistema, lo que nos enfoca en prevenir los errores en elementos con memoria ante una eventual falla como las causadas por los *SEU* que causen un cambio en la información contenida.

De igual forma, confirmé la necesidad de un paradigma de diseño como el método *Top-Down* que permite conformar sistemas con alto nivel de abstracción e ir desarrollando hasta su mínima expresión en bajo nivel como máquinas que trabajan en conjunto. Esto permite eficientar el proceso de diseño, considerando desde el primer momento las necesidades y requerimientos de cada desarrollo; permitiendo además, de que en caso de obtener un mal comportamiento en un subsistema, este pueda ser replanteado sin necesidad de rediseñar a los demás subsistemas.

Asimismo, al desarrollar este trabajo desde un nivel de abstracción tan bajo se permitió que la identificación y posterior protección de puntos críticos para la operación del sistema ante eventuales *SEE* fuera sencilla, ya que de haberse hecho con niveles más altos de abstracción se desconocería qué compone internamente a la máquina y aunque tanto en *Quartus* como *Vivado* se puede obtener una visualización con la herramienta de *RTL viewer* de la máquina sintetizada, esta máquina dependerá del sintetizador que se esté usando y las consideraciones que este tenga para agregar u optimizar en cuanto a recursos lógicos. Además de depender de la implementación, es decir del *FPGA* en el que se cargue la configuración, puesto que cambia la arquitectura incluso entre familias, lo que genera de que para cada una pueda existir una variante del diseño interno acorde a los parámetros que se fijen en el *Fitter*, provocando que se tenga que realizar aplicar un mecanismo de tolerancia a fallas para cada *FPGA* que se quiera usar. Es entonces que este diseño al más bajo nivel satisface las problemáticas previamente mencionadas, ya que utilizando una descripción estructural el diseñador tiene el control de la descripción de hardware con mucha mayor independencia de las consideraciones del sintetizador, comparado con respecto a la descripción que generaría el sintetizador; además de que por su modularidad, permite sustituir al elemento más básico por uno con mecanismos de tolerancia a fallas, lo que permite obtener una mayor confiabilidad que si se implementan los mismos mecanismos pero con un grado de abstracción mayor.

6.1. Trabajo a futuro

Debido a que este módulo busca cumplir con la especificación de I2C, hay múltiples funcionalidades opcionales que se observan en la tabla 2.1. Además de que este desarrollo es parte de todo un proyecto para un *Sistema de Comando y Manejo de Información* en *FPGA*, por lo que aún hay múltiples tareas a realizar.

- Filtro para glitches.
- Implementar Interfaz *AXI*.

- Crear un módulo de *IP*.
- Probar como periférico del Núcleo *Microblaze*.
- Adaptar las tramas para velocidades mayores a *F/S*.
- Compatibilidad para direcciones de 10 bits.
- Soporte para una arquitectura multi-maestro (Sincronización y Arbitraje).
- Integrar procesos de inyección a fallas para identificar los puntos débiles del sistema.

Bibliografía

[1] N. Semiconductors, "I2C-bus specification and user manual UM10204. Rev. 7.0," October 2021, disponible en: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[2] A. Pang and P. Membrey, "Beginning FPGA: Programming Metal," *Apress Media LLC*, p. 3, 2017.

[3] Altera, "FPGA Architecture, *Rev. 1.0*," 2006.

[4] A. Pavlov and M. Sachdev, *CMOS SRAM circuit design and parametric test in nano-scaled technologies: process-aware SRAM design and test.* Springer Science & Business Media, 2008, vol. 40.

[5] M. Yang, G. Hua, Y. Feng, and J. Gong, *Fault-tolerance techniques for spacecraft control computers.* John wiley & sons, 2017.

[6] B. W. Johnson, *Design & analysis of fault tolerant digital systems.* Addison-Wesley Longman Publishing Co., Inc., 1988.

[7] E. Dubrova, *Fault-tolerant design.* Springer, 2013.

[8] T. Instruments, "Tiva™ tm4c1294ncpdt microcontroller," *TM4C1294NCPDT datasheet,[Revised June 2014]*, 2014.

[9] STMicroelectronics, "Rm0394 reference manual," *ST.LifeAugmented,[Rev. 4.0]*, 2018.

[10] I. CAST, "I2C-SMbus controller." disponible en: https://www.cast-inc.com/sites/default/files/pdfs/2023-06/cast_i2c-smbus-amd.pdf.

[11] S. Churiwala and I. Hyderabad, "Designing with Xilinx® FPGAs," in *Circuits & Systems.* Springer, 2017.

[12] N. Patil, D. Das, E. Scanff, and M. Pecht, "Long term storage reliability of antifuse field programmable gate arrays," *Microelectronics Reliability*, vol. 53, no. 12, pp. 2052–2056, 2013.

[13] N. Battezzati, L. Sterpone, and M. Violante, *Reconfigurable field programmable gate arrays for mission-critical applications.* Springer Science & Business Media, 2010.

[14] H. Haznedar, *Digital microelectronics.* Benjamin-Cummings Publishing Company, 1991.

[15] A. Gohel and R. Makwana, "Multi-layered shielding materials for high energy space radiation," *Radiation Physics and Chemistry*, vol. 197, p. 110131, 2022.

[16] H. Garrett, "Guide to modeling earth's trapped radiation environment," *AIAA*, 1999.

[17] M. M. McCormack, "Trade study and application of symbiotic software and hardware fault-tolerance on a microcontroller-based avionics system," Ph.D. dissertation, Massachusetts Institute of Technology, 2011.

[18] F. H. Schmidt Jr, "Fault tolerant design implementation on radiation hardened by design sram-based fpgas," Ph.D. dissertation, Massachusetts Institute of Technology, 2013.

[19] E. Petersen, *Single event effects in aerospace*. John Wiley & Sons, 2011.

[20] Y. Chiang, C. M. Tan, C.-J. Tung, C.-C. Lee, and T.-C. Chao, "Lineal energy of proton in silicon by a microdosimetry simulation," *Applied Sciences*, vol. 11, no. 3, p. 1113, 2021.

[21] M. M. Mano and J. F. Gonzalez, *Diseño digital*. Pearson Educación, 2003.

[22] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving fpga design robustness with partial tmr," in *2006 IEEE International Reliability Physics Symposium Proceedings*. IEEE, 2006, pp. 226–232.