



Universidad Tecnológica Nacional  
Facultad Regional Bahía Blanca

**Ingeniería Electrónica**

**Proyecto Final**

**Desarrollo de prototipo para ampliación de la  
memoria RAM de computadoras navales, de 16k  
palabras de 24 bits a 64k palabras**

**Autor**

**Diego Hernan Alvarez**

**Director/es**

**Mg. Guillermo Friedrich**

**Ing. Adrian Laiuppa**

**Codirector**

**Ing. Christian L. Galasso**

**Bahía Blanca | 10 de abril de 2024**

## Índice de contenidos

<b>1. Resumen.....</b>	<b>4</b>
<b>2. Palabras claves.....</b>	<b>5</b>
<b>3. Introducción.....</b>	<b>6</b>
<b>4. Cronograma de Tareas.....</b>	<b>9</b>
<b>5. Hardware.....</b>	<b>10</b>
5.1 Sistema de distribución de potencia.....	10
5.1.1 Fuentes de alimentación.....	10
5.1.2 Capacitores de desacople.....	15
5.2 FPGA y módulos asociados.....	17
5.2.1 Utilización de pines de la FPGA.....	17
5.2.2 Puerto de programación JTAG.....	20
5.2.3 Oscilador de referencia.....	21
5.2.4 Conexión con SRAM.....	22
5.2.5 LEDs indicadores de estado y puntos de prueba.....	23
5.3 Conexión con el G-PPAL-BUS.....	24
5.4 Características generales del PCB.....	27
<b>6. Diseño digital.....</b>	<b>28</b>
6.1 Verificación del diseño digital RAM-MU16.....	29
a) Módulo Registrar_entradas.vhd.....	29
b) Módulo Gen_v0.....	30
c) Módulo MAD.....	30
d) Módulo TAC.....	31
e) Módulo Paridad_v0.....	33
f) Módulo Prueba_FSM_y_MANEJASRAM.....	34
g) Módulo RAM_MU (archivo top).....	35
6.2 Cambios para el diseño digital RAM-MU64.....	41
6.3 Componentes del diseño digital.....	47
a) Registrar_entradas_v1.vhd.....	47
b) Filtro_glitches_v2.vhd.....	49
c) gen_v0.vhd.....	49
d) MAD_v0.vhd.....	49
e) TAC_v1.vhd.....	50
f) Paridad_v0.vhd.....	51
g) GEN_PAR_V0.....	51
6.4 Etapa de pruebas y corrección de errores del diseño digital en prototipo anterior.....	54
6.4.1 Configuración inicial del prototipo.....	54
6.4.2 Funcionamiento del modo 16k.....	55
6.4.3 Funcionamiento del modo 64k.....	57
6.4.4 Modificación y depuración del diseño digital.....	57
<b>7. Conclusiones.....</b>	<b>61</b>

**8. Bibliografías y referencias bibliográficas..... 62**

# 1. Resumen

El proyecto consiste en el desarrollo de la tercera versión de un prototipo que permita el reemplazo de las memorias RAM existentes en las computadoras de los buques del Comando de la Flota de Mar perteneciente a las F.F.A.A., teniendo como ámbito físico de desarrollo el Servicio de Análisis Operativo, Armas y Guerra Electrónica (SIAG).

Se analizará toda la investigación y diseño generado para la construcción del segundo modelo, realizando la validación de las diferentes etapas de desarrollo, detección de errores e introducción de modificaciones que resulten vitales para el correcto funcionamiento del bloque, dando lugar a una versión optimizada de la unidad.

A modo ilustrativo, en cuanto a especificaciones técnicas, el módulo a reemplazar es una memoria dinámica (DRAM) basada en circuitos del tipo CMOS, caracterizada principalmente por:

- Capacidad de almacenamiento de aproximadamente 16.000 palabras de 24 bits (16k24).
- Diferentes modos de operación: escritura en palabras enteras de 24 bits, en media palabra de 12 bits o como caracteres de 8 bits, mientras que la lectura solo puede realizarse en palabras de 24 bits.
- Ciclos de ejecución de alrededor de 750 ns.
- Control de paridad de cuatro bits por palabra (16k4).
- Capacidad “tri-state”, dado que se pueden conectar hasta cuatro unidades en un bus de datos común (G-PPAL-BUS), dando lugar a un espacio de almacenamiento total de 64k24 bits.

Al igual que la versión anterior, el circuito de reemplazo estará compuesto, básicamente, por una matriz de compuertas programables (FPGA) como núcleo principal, un circuito integrado de memoria (SRAM) para el almacenamiento de datos, diversos buffers y transceptores de adaptación de tensiones y sus respectivas fuentes de alimentación. La nueva característica que se agregará es la posibilidad de agrupar los 64k24bits del G-PPAL-BUS en una sola unidad. Por lo tanto, será posible seleccionar, mediante el hardware, si la unidad funciona como memoria de 16k24 bits o 64k24bits. Esto se debe a que en algunos sectores específicos de las computadoras navales se requiere, de manera obligatoria, un módulo de 16k24bits. Se partirá desde un prototipo de hardware existente, el cual tiene la capacidad de memoria para almacenar los nuevos datos. Por lo tanto, se diseñará en primera instancia la descripción de hardware para este prototipo y se obtendrá el código final con la nueva función. Luego de obtener el código depurado, se realizará la versión final de la placa de circuito impreso definitiva. Esto se debe a que algunos componentes de la placa prototipo se encuentran obsoletos y no son recomendados para nuevos diseños.

Luego del desarrollo del prototipo funcional del módulo de memoria RAM, este trabajo presenta su rediseño con la corrección de errores detectados en la versión anterior. Esta nueva versión, lista para la producción, incorpora la capacidad de agrupar los 64k24bits del G-PPAL-BUS en una sola unidad, permitiendo su uso como memoria de 16k24bits o 64k24bits mediante selección por hardware.

## 2. Palabras claves

Expansión de memoria, computadora naval, RAM, FPGA, diseño digital

### 3. Introducción

Según se muestra en el diagrama en bloques de la figura 1, el hardware debe contener la lógica implementada sobre una FPGA, un integrado de memoria SRAM en donde almacenar los datos, buffers tri-state bidireccionales para aislar y adaptar las tensiones de las señales del G-PPAL-BUS con las de la FPGA y un sistema de distribución de potencia capaz de otorgar las diferentes tensiones requeridas en el diseño. Además, debe tenerse en cuenta que se requiere un puerto de programación JTAG para la FPGA. Otro requisito indispensable es el agregado de un oscilador de referencia para la FPGA, el cual será de 50 MHz. Por último, el diseño también debe contener LEDs indicadores de estado de la FPGA (CRC/INIT y DONE) más un led indicador de fallo de paridad.

Entre los requerimientos funcionales del prototipo también está el hecho de que debe soportar condiciones de operación exigentes como la tolerancia a temperaturas elevadas, vibraciones de baja y alta frecuencia e interferencia electromagnética de diversas fuentes. Adicionalmente, el espacio físico y la forma de montaje de cada uno de los componentes del sistema introduce restricciones, por lo cual se requiere de un diseño robusto y con un factor de forma determinado.

Se utilizó un sistema de distribución de potencia basado en el implementado en el proyecto ETHER-TMIO, ya que se trata de un diseño probado y que permite estandarizar los componentes utilizados en los distintos desarrollos del grupo. Lo mismo sucede con el chip de FPGA (*Xilinx FPGA*, n.d.) que es un componente fiable y utilizado en todos los prototipos anteriores. En el caso de la memoria SRAM, se utilizó un integrado de 4 Mbit (*CY7C1049GN30-10ZSXI*, n.d.), capaz de albergar los necesarios 64 k palabras de 24 bits de datos más 4 bits de paridad, es decir, 1792 kbits. Los buffers de adaptación utilizados fueron los LVTH2245 (*SN74LVTH2245*, 1997) y en el caso de las señales que necesitan un buffer del tipo open collector el SN74LVC07A (*SN74LVC07A*, n.d.). Se optó por los integrados LVTH2245, respecto a los 74LVCH8T245 utilizados en las versiones anteriores, ya que estos poseen una mayor capacidad de corriente. En algunas pruebas realizadas con los prototipos anteriores se observó un comportamiento errático aleatorio y se le atribuyó a la diferencia de manejo de corriente que existe entre los integrados originales y el integrado 74LVCH8T245.

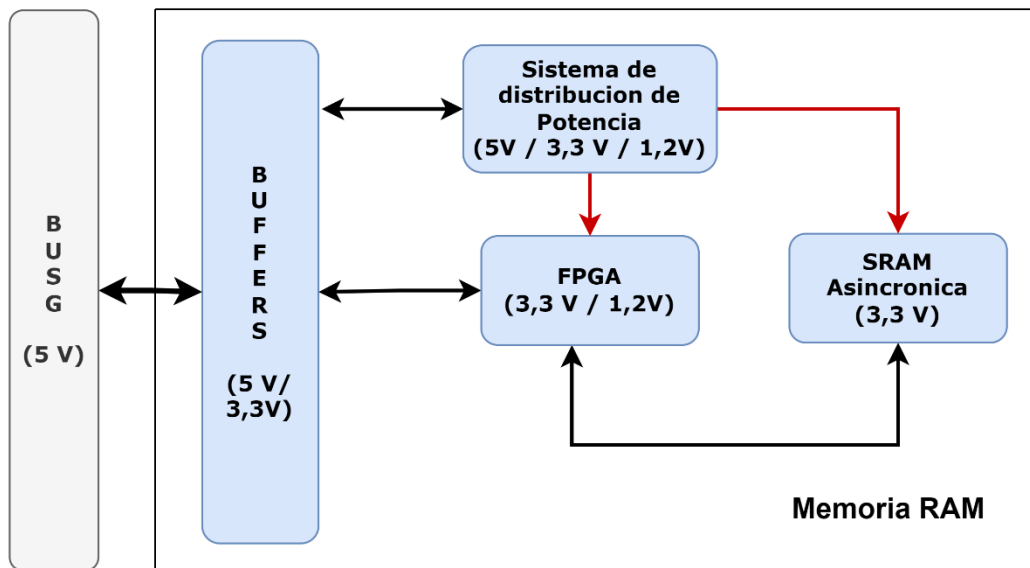


Figura 1 – Diagrama de bloques del hardware perteneciente al proyecto RAM-MU64.

De esta forma, tomando en cuenta las especificaciones definidas hasta el momento, se concibe la versión final del hardware del proyecto RAM-MU64, el cual se puede observar en la figura 2.

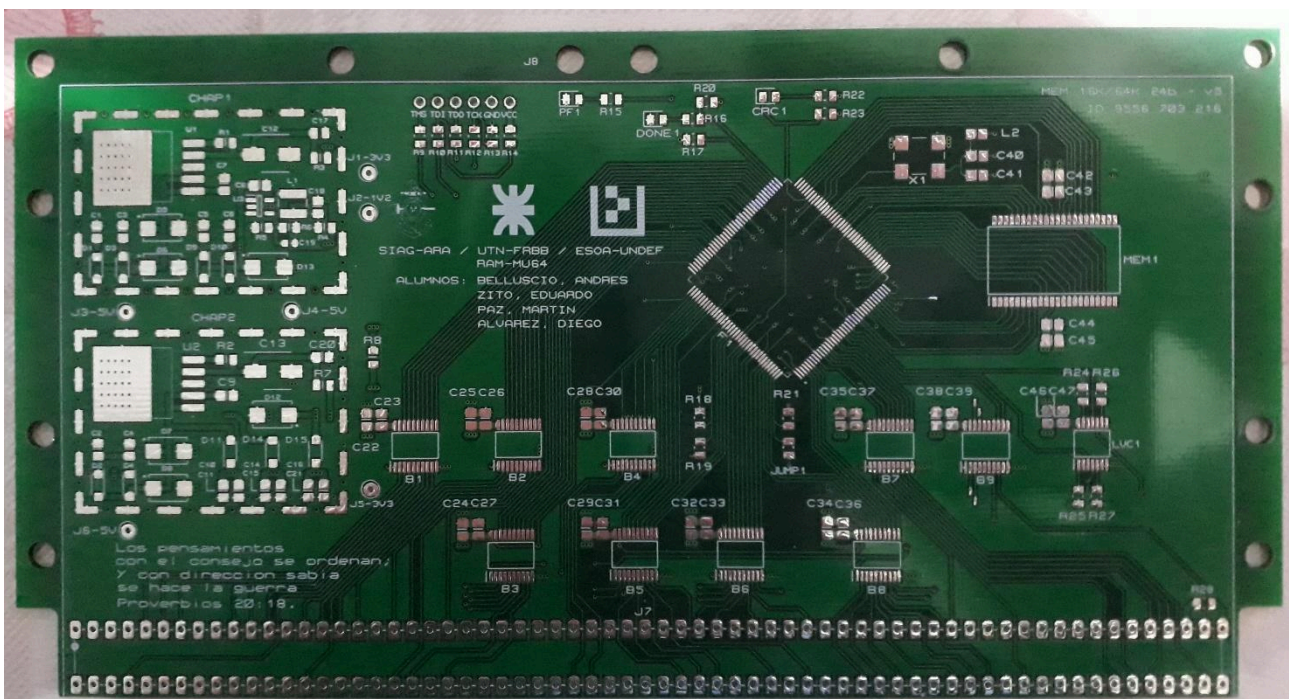


Figura 2 – Hardware final del proyecto RAM-MU64.

## 4. Cronograma de Tareas

En colaboración con el codirector, se estableció un cronograma detallado para planificar eficientemente la ejecución del proyecto final de carrera. Este detalla objetivos parciales a cumplir en un periodo máximo de seis meses, garantizando una gestión temporal óptima.

	Descripción	Mes							Resultado esperado al final de la etapa	Indicadores verificables de cumplimiento
		1/ 21	2/ 21	3/ 21	4/ 21	5/ 21	6/ 21	7/ 21		
Revisión general	Lectura y análisis de información existente y de documentos generados a partir de las versiones anteriores.	X							Conocimiento general del funcionamiento del módulo a reemplazar.	Completa disposición de documentación requerida
Verificación de HDL	Verificación funcional de la última versión del código VHDL en funcionamiento.	X							Depurar posibles errores existentes y validar el VHDL antiguo.	Simulaciones a través de bancos de prueba.
HDL	Diseño RTL de la nueva estructura de funcionamiento. Desarrollo del primer código VHDL adaptado a un prototipo anterior capaz de almacenar los nuevos datos.		X						VHDL corregido para el nuevo PCB.	Disposición del primer VHDL completo funcionando en bancos de prueba.
Ensayos del nuevo HDL en prototipo anterior	Pruebas en laboratorio (SIAG) con el nuevo HDL. Depuración de bugs en funcionamiento.				X				Eliminación de bugs en funcionamiento del VHDL del primer prototipo.	Equipamiento necesario para el análisis y detección de errores de funcionamiento de la primera versión
Depuración	Validación del VHDL.				X				VHDL definitivo	Errores encontrados durante el proceso previo de análisis
Pruebas finales	Pruebas del segundo prototipo. En laboratorio (SIAG) y a bordo.								Resultados de funcionamiento	Equipamiento necesario para funcionamiento "in situ" del módulo de memoria.
Rediseño del circuito esquemático y del circuito PCB para producción.	Búsqueda de los nuevos componentes necesarios y reemplazo de componentes obsoletos. Rediseño de los circuitos con los componentes.								Diseño definitivo de del circuito esquemático y del PCB.	Disposición del circuito PCB final.
Correcciones	Posibles modificaciones (de ser necesarias).							X	Detección final de errores	Resultados previos de pruebas y ensayos definitivos
Documentación	Originar información de pruebas y resultados.	X	X	X	X	X	X	X	Detalle del proceso de desarrollo del proyecto	Necesidad de figurar información generada

## 5. Hardware

### 5.1 Sistema de distribución de potencia

Un sistema de distribución de potencia (PDS) debe poder proveer la energía necesaria para el correcto funcionamiento de los componentes integrantes del circuito. De esta forma, se deben contemplar los requerimientos de potencia RMS y de pico. Adicionalmente, para los PDS de circuitos digitales, se exigen requisitos de bajo ruido y baja tolerancia para la tensión, a fin de garantizar el correcto desempeño de los circuitos integrados.

A fin de cumplir con los requisitos explicitados, un sistema de distribución de potencia se compone, de forma general, de dos elementos principales:

- Reguladores de tensión.
- Capacitores de desacople.

El regulador de tensión monitorea la tensión de salida y ajusta la corriente de salida a fin de mantener la tensión a valor constante. El regulador actúa ante cambios de baja frecuencia de la carga, pudiendo extender su rango hasta algunas centenas de kHz. Es por ello que, para reducir el ruido de alta frecuencia, el cual se debe a la conmutación de las salidas de la FPGA, las cuales generan un pico de corriente, se requiere de la presencia de una red de capacitores de desacople. Los mismos actúan como almacenadores de energía local para el dispositivo, y de acuerdo con su valor, podrán actuar rápidamente para responder a cambios de alta frecuencia de la demanda de corriente.

#### 5.1.1 Fuentes de alimentación

Como se mencionó anteriormente, el sistema de distribución de potencia utilizado en este diseño se basa en el diseño del ETHER-TMIO. Este esquema consiste en tres convertidores DC-DC, los cuales son alimentados a partir de una fuente de 5 V. Dos convertidores son del tipo lineal y regulan 3,3 V. El restante, es un convertidor por conmutación que se configura para regular una salida de 1,2 V. En el caso de este proyecto particular, la tensión de entrada se recibe directamente de la alimentación del sistema del buque. Además, puede ser tomada desde dos líneas distintas: XV5P1 o XV5P2.

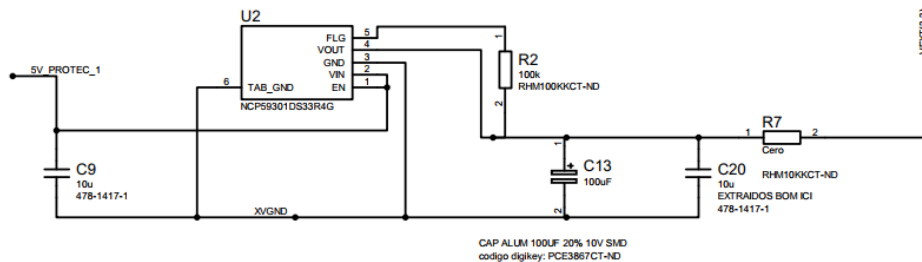
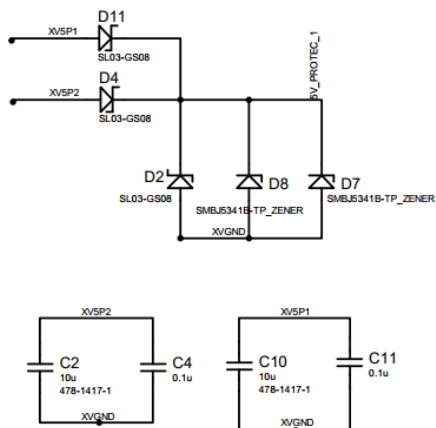
Un regulador de 3,3 V se utiliza para la alimentación de los bancos de I/O de la FPGA y la memoria SRAM, el restante se encarga de alimentar a los buffers bidireccionales. Por otro lado, la tensión de 1,2 V se utiliza para alimentar el núcleo de la FPGA. Por defecto, se toma la alimentación a partir de la línea XV5P2, pero si se retiran los diodos D4, D9, y D3 y se colocan los diodos D11, D15 y D14 se utiliza la línea XV5P1.

Para las tensiones de 3,3 V se utilizó el regulador línea NCP59300 (*NCP59300 LDO Regulator*, n.d.), capaz de manejar hasta 3 A de corriente. En cuanto a la tensión del núcleo de la FPGA, se optó por el regulador conmutado (*NCP1521B - Buck Converter - DC-DC 1.5 MHz, 600 mA*, n.d.) que entrega hasta 800mA, lo cual es suficiente para este proyecto. Un reemplazo posible para este integrado, ante la falta de stock, es el TPS62561 (*TPS62561 Step-Down Converter*, n.d.). Además, se adiciona una protección

ante picos de tensión para las entradas de cada una de las fuentes de alimentación, la cual se conforma por diodos TVS y diodos fusibles. El último elemento de las fuentes de alimentación son dos encapsulados metálicos, referenciados a tierra, que cubren a los convertidores DC-DC y a los diodos. Estos escudos actúan como una jaula de Faraday, lo que permite mejorar la compatibilidad electromagnética con el resto del diseño y componentes externos a la placa. El circuito esquemático se puede observar en la figura 3.

Fuente 3,3 V para dispositivos externos

\*NOTA: D11 no se suelda, PORQUE SE USA XV5P2



Fuente para el núcleo FPGA

\*NOTA: D15 no se suelda, PORQUE SE USA XV5P2

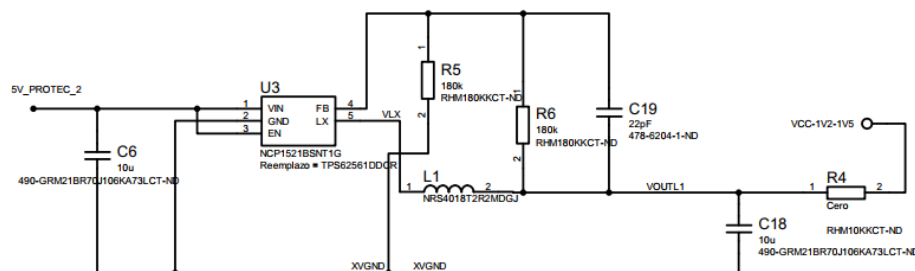
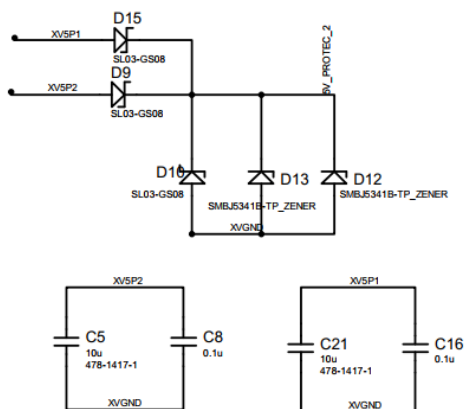


Figura 3 – Circuito esquemático de los convertidores DC-DC.

Una vez definidos los circuitos de regulación y conversión de tensión se efectúa el diseño del PCB. En esta versión se optó por integrar los diodos dentro de los escudos electromagnéticos para evitar que cualquier ruido generado por un elemento alineal sea disperso hacia el resto de los componentes. Esto llevó a agrupar los reguladores de tensión dentro de dos encapsulados metálicos, ya que no cabían los componentes de los tres reguladores más los diodos en uno solo. Uno de los escudos cubre los reguladores de la FPGA, tanto del núcleo como de los pines I/O y la memoria SRAM, y el otro los buffers bidireccionales.

En la Figura 4 se muestra la disposición final de los componentes pertenecientes a los reguladores de la FPGA junto a los diodos de protección y capacitores que toman la tensión de entrada desde XV5P2. En la figura 5 se presenta la distribución de los componentes pertenecientes al regulador de tensión para los buffers bidireccionales y los diodos de protección más los capacitores que toman tensión de XV5P2. Además, se encuentran todos los diodos y capacitores que se conectan a XV5P1. Estos últimos no deben ser soldados, al menos que se requiera obtener la alimentación desde esas líneas.

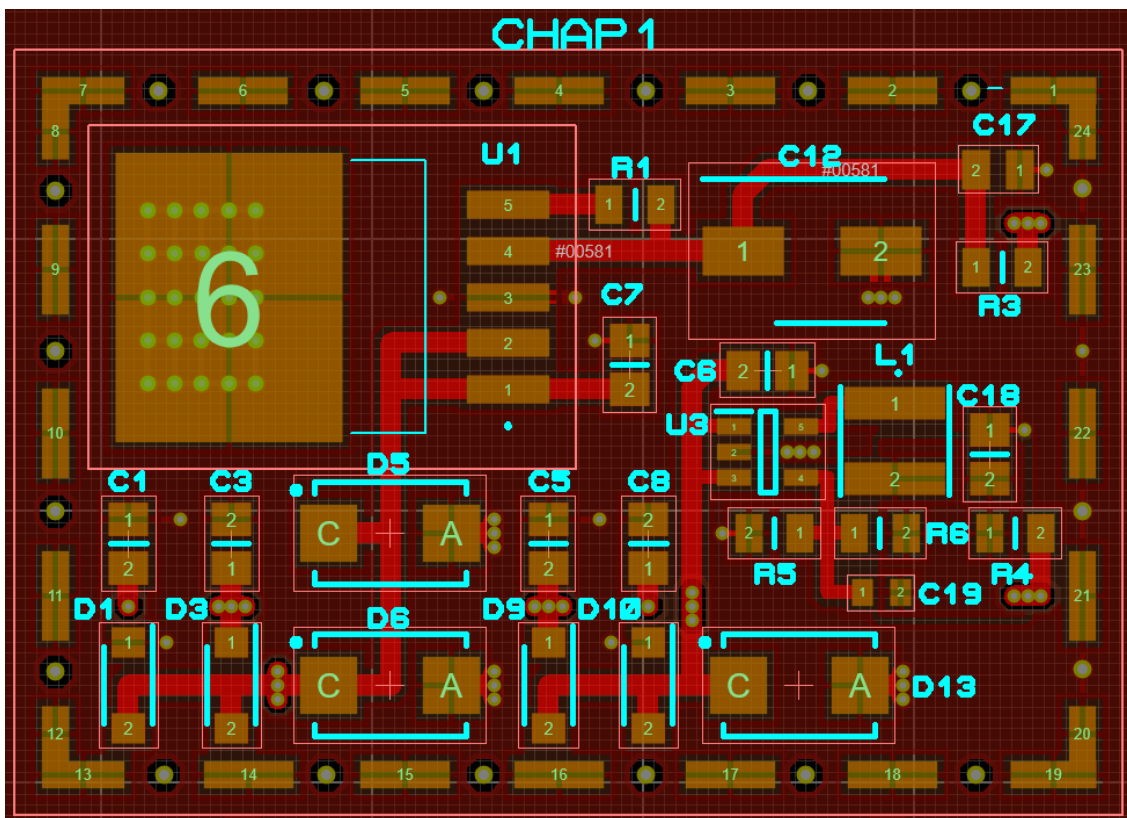


Figura 4 – Agrupación de los convertidores DC-DC para la FPGA.

Para determinar el número de capas a utilizar en el diseño del circuito, se tuvo en cuenta una consideración que brindan en la hoja de datos los fabricantes de los reguladores de voltaje. Estos aseguran un correcto funcionamiento de los dispositivos si existen dos capas particulares, una de tierra y la otra de VCC, respectivamente (en esta se incluyen las diferentes tensiones de salidas de las distintas fuentes). Así, se dio lugar a una configuración, o layer stackup, total de cuatro capas: la capa número 1 es el top copper (superior), donde se encuentra la gran mayoría de componentes y pistas; la capa número 2 (interna) corresponde al plano de masa; la capa número 3 (interna) corresponde

a las diferentes alimentaciones del circuito y, por último, la capa número 4 es la bottom copper (inferior), donde se colocan diversos capacitores y pistas que no son factibles de implementar en la capa superior.

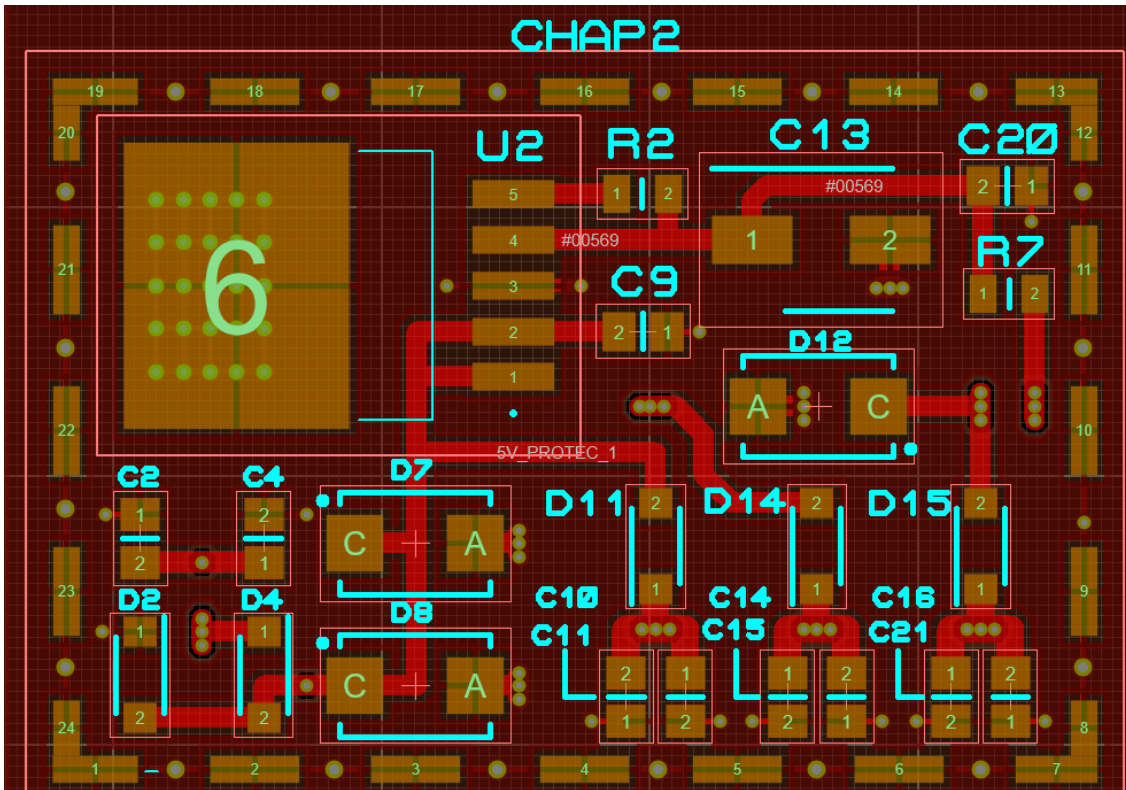


Figura 5 – Convertidor DC-DC para los buffers bidireccionales.

La ubicación de los componentes se realizó siguiendo las guías incluidas en las hojas de datos de los fabricantes de los reguladores. Sobre todo, las que mencionan cuáles elementos externos deben colocarse lo más próximo al regulador como sea posible.

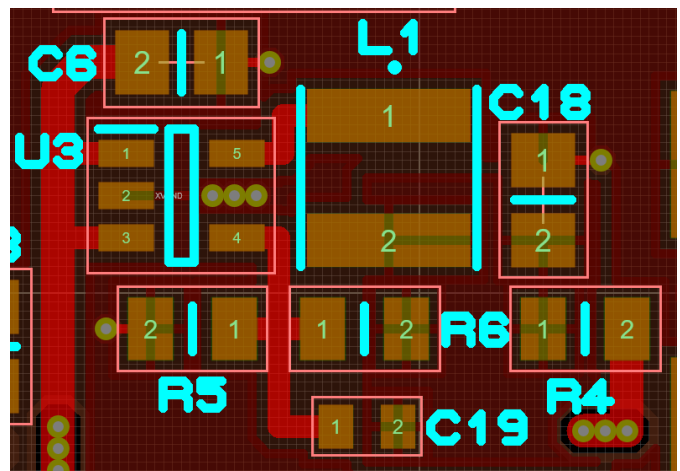
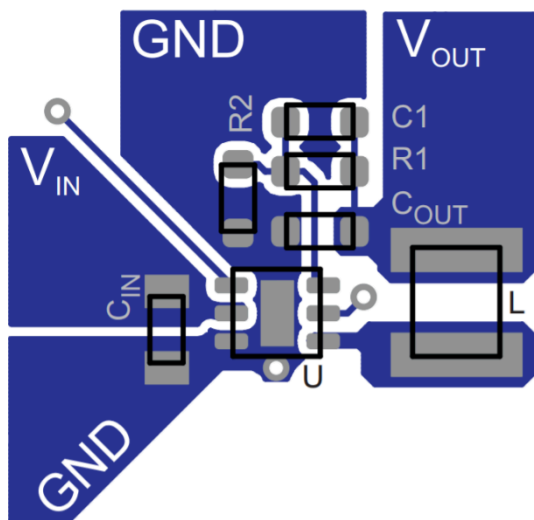


Figura 6 – Convertidor DC-DC para el núcleo de la FPGA.

El convertidor al cual se le prestó más atención respecto a esto fue al TPS62561, ya que se trata de una fuente conmutada y es más sensible al ruteo que las fuentes lineales. En la Figura 6 se muestra una comparación entre la disposición recomendada y la que se realizó en este diseño.

Por otro lado, en lo que respecta al ruteo de componentes, el fabricante sugiere aumentar el ancho y reducir la longitud de las pistas de gran transporte de corriente. Teniendo esto en cuenta, los anchos de las pistas se hicieron lo más grueso posible sin que se comprometa la disipación de calor de los pads al momento de la soldadura y se “levanten” los componentes por diferencia térmica entre sus extremos. Esto se explica en las recomendaciones de diseño IPC (2221A *Table of Contents*, n.d.). Para evitar este tipo de problemas, el ancho de la pista debe ser inferior al del pad más chico que se conecta.

Con la ayuda del software Saturn PCB (*Saturn PCB Toolkit*, n.d.) se chequeó que el diámetro de las vías colocadas y las pistas soporten la corriente máxima teórica que admiten los reguladores. En virtud de garantizar un retorno a tierra de baja inductancia y resistencia, se utilizaron varias vías pequeñas en paralelo. Si bien cuanto más gruesa es una vía, más baja es su inductancia a altas frecuencias, colocar varias en paralelo baja aún más el valor del conjunto. Esto se debe a que las vías actúan como inductores en paralelo y la suma total es menor a la más pequeña. Por el mismo motivo se implementó también esta técnica en las conexiones desde la capa superior hacia el plano de alimentación. Por último, para garantizar que las corrientes de retorno no generen bucles de corriente en la superficie, se colocaron vías de retorno a tierra cercanas a cada componente que se refiere al plano de tierra.

### 5.1.2 Capacitores de desacople

Como se mencionó previamente, la presencia de capacitores de desacople es requerida por todos los circuitos digitales, ya que las conmutaciones rápidas que deben producir necesitan de una disponibilidad de potencia en un tiempo relativamente corto. Adicionalmente, estos capacitores permiten derivar ruido de la alimentación a masa, mejorando la calidad de la energía entregada por la fuente.

Para que el capacitor de desacople cumpla su función de proporcionar la potencia instantánea necesaria, necesita ubicarse cercano al integrado al cual brinda energía, puesto que la inductancia de la línea aumenta la impedancia equivalente disminuyendo la rapidez de descarga de este. En este sentido, la calidad y el montaje del capacitor también juegan un rol importante en la disminución de la inductancia parásita total.

En este diseño, se deben colocar capacitores de desacople en los distintos componentes activos, además de los presentes en los reguladores de tensión, como: los buffers bidireccionales, la FPGA y el integrado de memoria SRAM. Además, se colocaron capacitores de filtrado cerca del conector del G-PPAL-BUS en cada pin de XV5P1 y XV5P2. En todos los casos mencionados anteriormente se utilizaron una pareja de capacitores de 10 $\mu$ F y 0.1 $\mu$ F en cada pin de alimentación. En la Figura 7 se muestra la disposición de los capacitores para el circuito integrado de la FPGA. Por otro lado, en la Figura 8 se pueden observar los capacitores de desacople correspondientes a los buffers bidireccionales y a las entradas de alimentación que ingresan desde el G-PPAL-BUS.

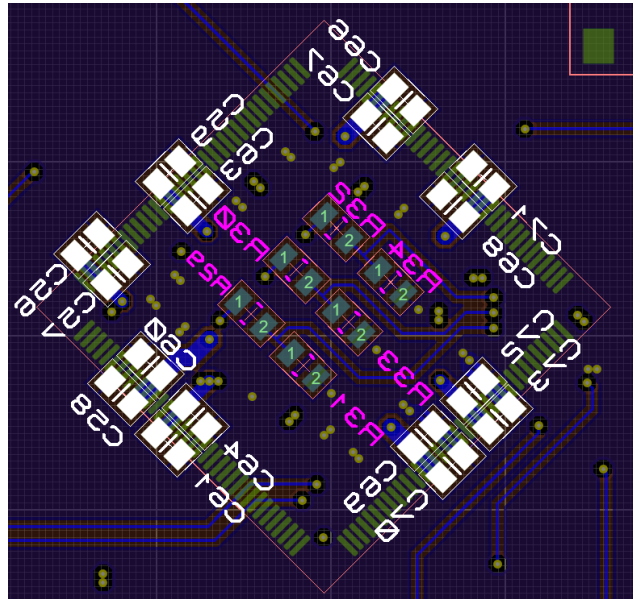


Figura 7 – Disposición de los capacitores de desacople de la FPGA.

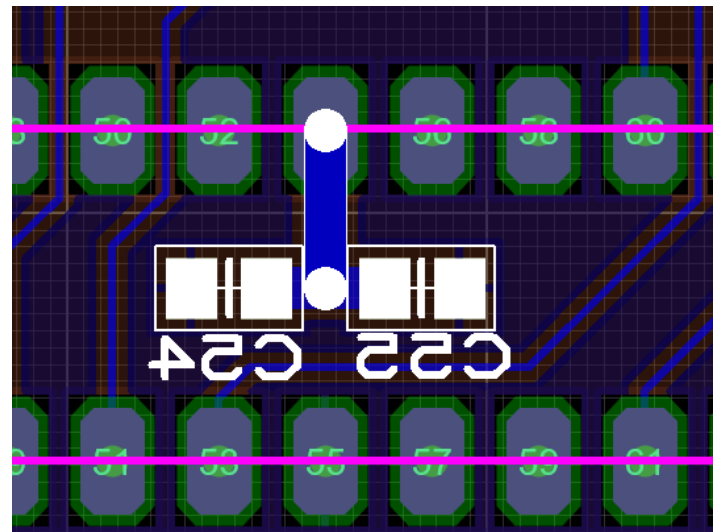
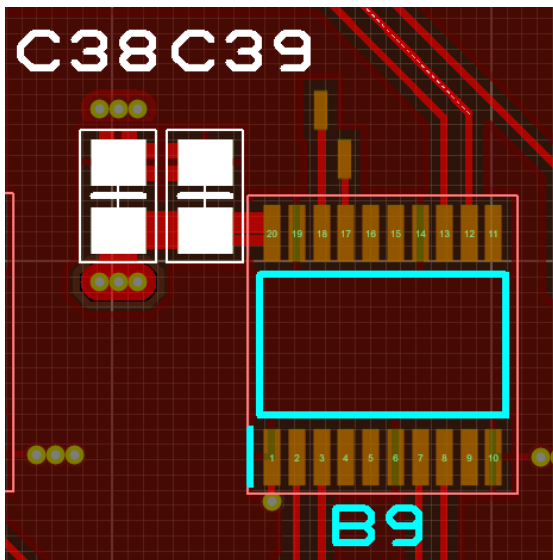


Figura 8 – Ubicación de los capacitores de desacople para los buffers bidireccionales (resaltados en blanco a la izq.) y las tensiones provenientes del BUS G (resaltados en blanco a la der.).

## 5.2 FPGA y módulos asociados

### 5.2.1 Utilización de pines de la FPGA

Una vez establecidos los requerimientos generales para el hardware, y luego de desarrollado y testado el diseño digital de la versión 4, fue posible determinar definitivamente la utilización de pines de la FPGA. En este sentido, la Tabla I muestra una clasificación funcional de las entradas y salidas empleadas sobre la FPGA. La cuenta arroja un total de 98 pines, lo cual representa una utilización del 90% de la FPGA de Xilinx, que posee un total de 108 puertos de entrada/salida. La ubicación de los pines puede realizarse libremente dentro de las entradas y salidas disponibles, lo cual se aprovecha para mejorar el ruteo del PCB. En la Figura 9 se observa la disposición final adoptada.

Tabla I: Utilización de pines de la FPGA.

Grupo	Número de puertos I/O requeridos	Tipo de puerto
Línea de reloj global de 50MHz (CLK_FPGA)	1	Entrada
Reset asincrónico (RESM_N)	1	Entrada
Pines Device Address (DA) del G-PPAL-BUS	4	Entrada
Señal para discriminar los módulos presentes en el G-PPAL-BUS (GA13A_N)	1	Entrada
Señal para discriminar los módulos presentes en el G-PPAL-BUS (GA14A_N)	1	Entrada
Direcciones del dato en el G-PPAL-BUS (GA_N)	16	Entrada
Reloj de 8MHz del G-PPAL-BUS (GEN)	1	Entrada
Identificador de acción en memoria (GW_N)	4	Entrada
Inhibidor de acción en memoria (MIH_N)	1	Entrada
Selección de modo de memoria en 16k o 64k (MODO_64k)	1	Entrada
Indicador del tipo de paridad (PR_N)	1	Entrada
Habilitación para escribir en el G-PPAL-BUS (READF1A4)	4	Entrada
Señal de inicio de ciclo (TMI_N)	1	Entrada
Dato proveniente del G-PPAL-BUS (GD_N)	24	Bidireccional
Dato por almacenar en la SRAM (DATO_SRAM)	8	Bidireccional
Dirección del dato en SRAM (ADDRESS_SRAM)	19	Salida
Señal output enable de la SRAM (OE_SRAM_N)	1	Salida
Señal chip enable de la SRAM (CE_SRAM_N)	1	Salida
Señal write enable de la SRAM (WE_SRAM_N)	1	Salida
Señal para controlar la dirección de los buffers bidireccionales (DIR_BUFF_BI)	1	Salida



con un resistor de pull up por defecto, sirven para establecer la configuración. Para este caso particular, se debe especificar que la FPGA se configure a partir de la memoria flash interna (modo Internal Master SPI). Luego de la configuración pueden emplearse como pines de entrada y salida, lo cual se aprovecha en el diseño.

Tabla II: Pines de configuración de la FPGA.

Nombre	Ubicación	Valor a lógico a adoptar
M0	IO L01P 2/M1 (Pin 38)	1
M1	IO L01P 2/M1 (Pin 37)	1
M2	IO L02P 2/M2 (Pin 39)	0
VS2	IO L04P 2/VS2 (Pin 43)	1
VS1	IO L03N 2/VS1 (Pin 44)	1
VS0	IO L04N 2/VS0 (Pin 45)	1

En la Figura 10 se muestra el esquemático de los divisores resistivos utilizados para configurar la FPGA y en la Figura 11 la ubicación en la placa de circuito impreso.

- M2: R32: 10K y R34: 100ohm para que tome el .bin de la flash interna**
- M1: R29 y R31: NO CONEC para que tome el .bin de la flash interna**
- M0: R30 y R33: NO CONEC para que tome el .bin de la flash interna**

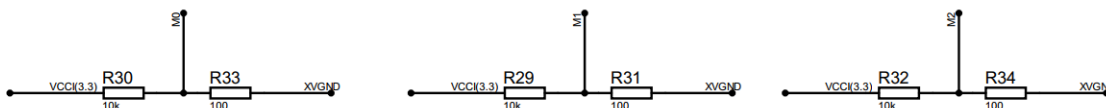


Figura 10 – Configuración por hardware para la FPGA.

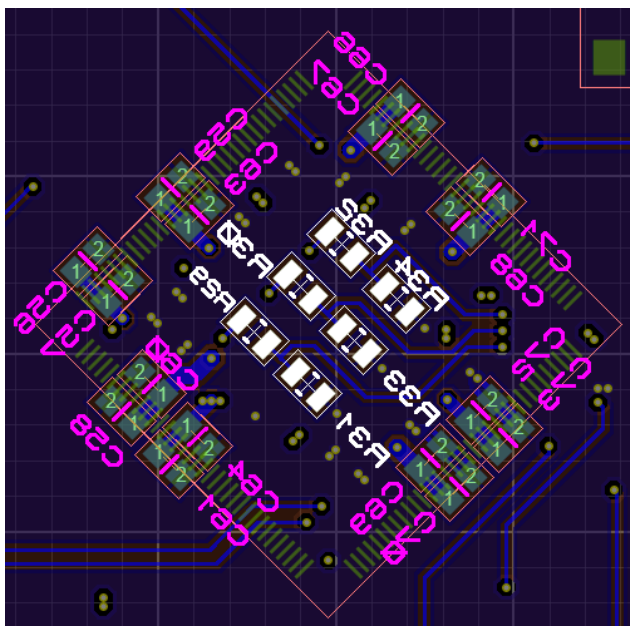


Figura 11 – Resistencias para configurar el modo Internal Master SPI (resaltadas).

## 5.2.2 Puerto de programación JTAG

La FPGA de Xilinx cuenta con una memoria flash interna, el cual guarda el código VHDL que permite la implementación del hardware deseado. A fin de que la FPGA ejecute las funciones previstas, es necesario entonces, grabar el código en la memoria flash, lo cual se realiza por medio de la interfaz de programación JTAG. El circuito utilizado es el mismo que el de los prototipos anteriores y se muestra en la Figura 12. Un detalle que se tuvo en cuenta es colocar el conector en la capa inferior (bottom), que es donde físicamente debe situarse para que no interfiera con el marco metálico. Finalmente, también se incluye en el PCB las inscripciones con los nombres de las funciones de cada pin, según se observa en la Figura 13, a fin de evitar errores en el conexionado.

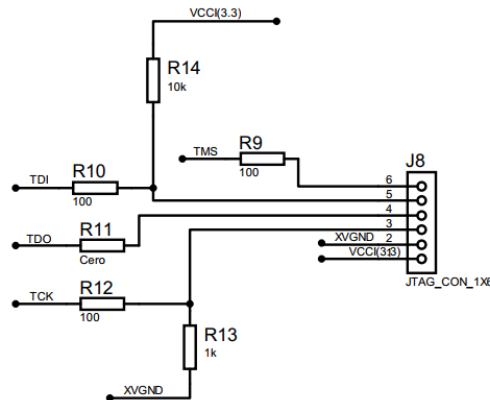


Figura 12 – Circuito esquemático del conector JTAG.

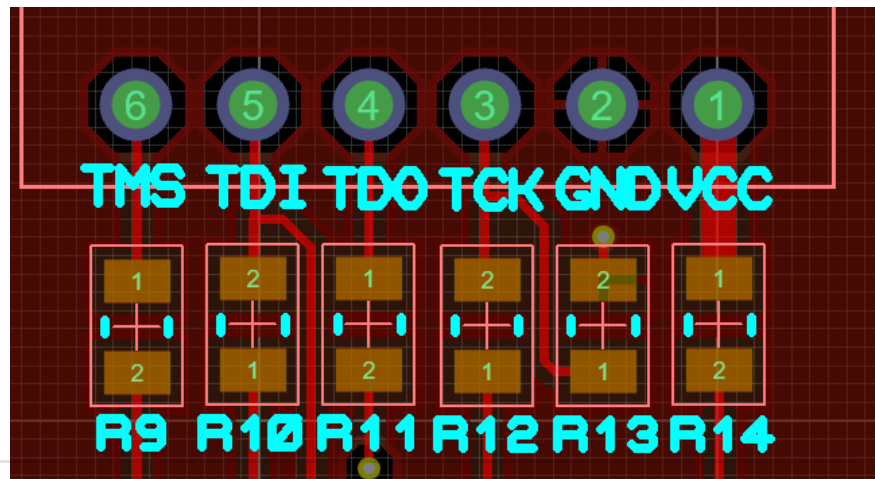


Figura 13 – Serigrafía del JTAG para ayudar en el conexionado.

### 5.2.3 Oscilador de referencia

La FPGA implementa un circuito secuencial síncrono, por lo cual requiere de una entrada de reloj, la cual es proporcionada por un oscilador de 50 MHz. Al igual que los reguladores de tensión, el cristal seleccionado es el mismo que el del proyecto ETHER-TMIO, con el objetivo de estandarizar los componentes entre proyectos.

El circuito esquemático se muestra en la Figura 14 y es el mismo que el sugerido por el fabricante del cristal en su hoja de datos. En la Figura 15 puede observarse el arreglo de los componentes en la placa de circuito impreso. Se colocó el cristal lo más cercano posible al pin de entrada correspondiente a la FPGA, ya que esta conexión es la más crítica de todo el diseño por ser la más rápida. En cuanto a las conexiones de alimentación, se colocaron tres vías en paralelo para el cristal y se aseguró, por medio de vías individuales, un retorno rápido hacia el plano de tierra para el resto de los componentes.

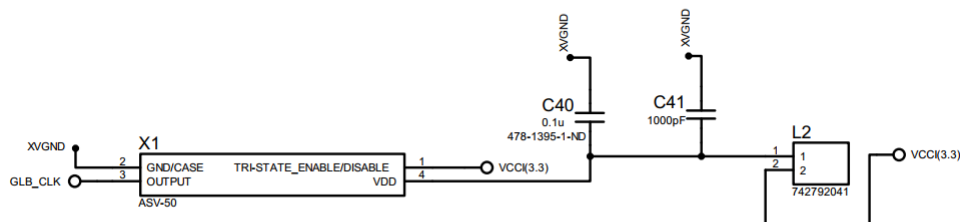


Figura 14 – Circuito esquemático del cristal de la FPGA.

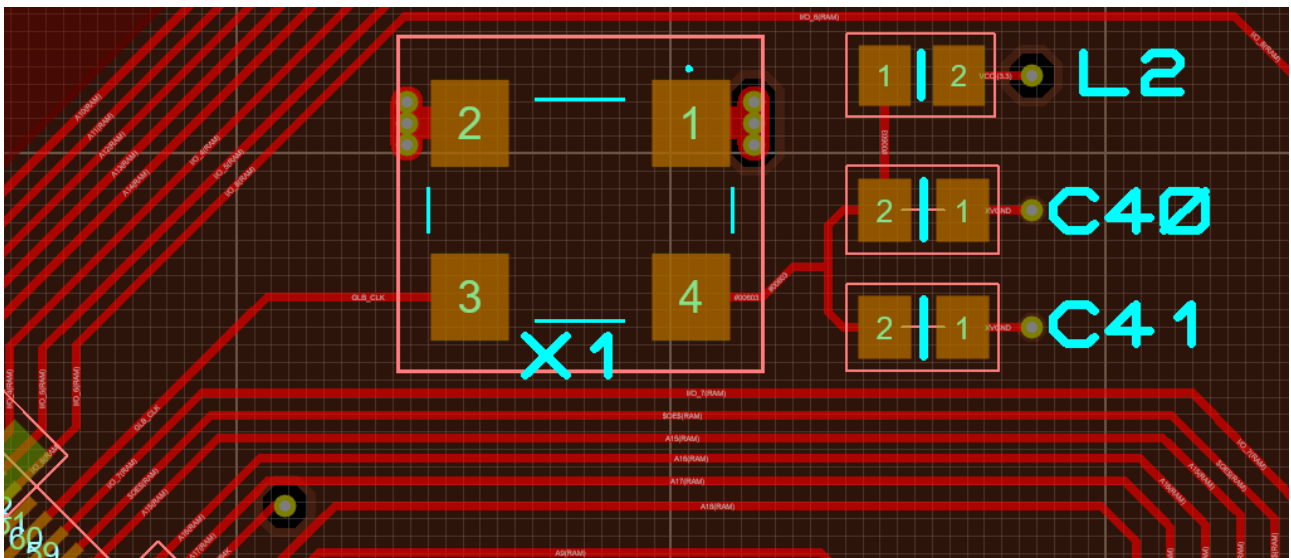


Figura 15 – Diseño del circuito del cristal en la placa de circuito impreso.

## 5.2.4 Conexión con SRAM

Uno de los periféricos centrales para el proyecto RAM-MU64 es el chip de memoria SRAM. Aquí es donde se almacenarán las 64k palabras de 28 bits necesarias para reemplazar cuatro bloques de memoria originales al mismo tiempo. Para lograr este aumento del almacenamiento de datos, se utiliza un integrado de memoria SRAM de la misma familia que el ya utilizado en los prototipos iniciales de este proyecto, pero con una capacidad de 4 Mbits. Cabe aclarar que sería suficiente utilizar una memoria que sea de 2 Mbits, pero, en base al análisis llevado a cabo sobre las opciones disponibles en el mercado, se las descartaron por distintas razones. En primer lugar, los tiempos de acceso de estos integrados se encuentran muy por encima de los observados en las memorias de 4 Mbits (50 ns frente a 10 ns, respectivamente) y no serían compatibles con la velocidad de reloj utilizada (40 o 50 MHz). Por otro lado, el costo de ambos integrados es similar, por lo que se optó por la versión de 4Mb para intentar ampliar la memoria a 80k palabras, función que hubiera sido utilizada en la unidad paginadora. Esto último se descartó, luego de realizar varias pruebas sobre la versión 4 y comprobar que, por el diseño del cableado en esa sección, no sería posible realizar un módulo de 80k. Por lo tanto, se utilizó el integrado CY7C1049GN30-10ZSXI, el mismo de la versión 4 y que ya fue comprobado que funciona de manera correcta con el diseño digital desarrollado. En la Figura 16 se muestra el circuito esquemático de la SRAM y en la Figura 17 la disposición en la placa de circuito impreso. Al igual que los otros elementos activos del diseño, este componente posee capacitores de desacople y se asegura un camino de baja inductancia hacia los planos de alimentación a través de vías paralelas.

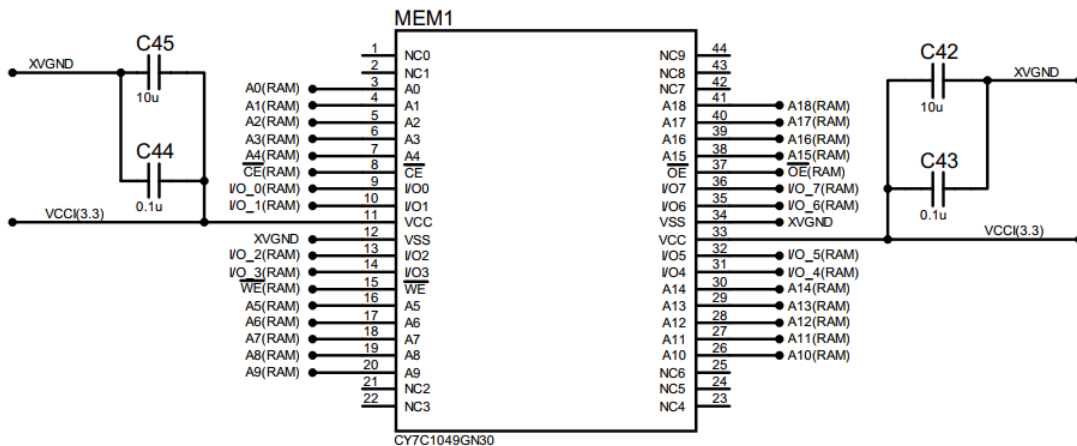


Figura 16 – Circuito esquemático y conexiones de la memoria SRAM.

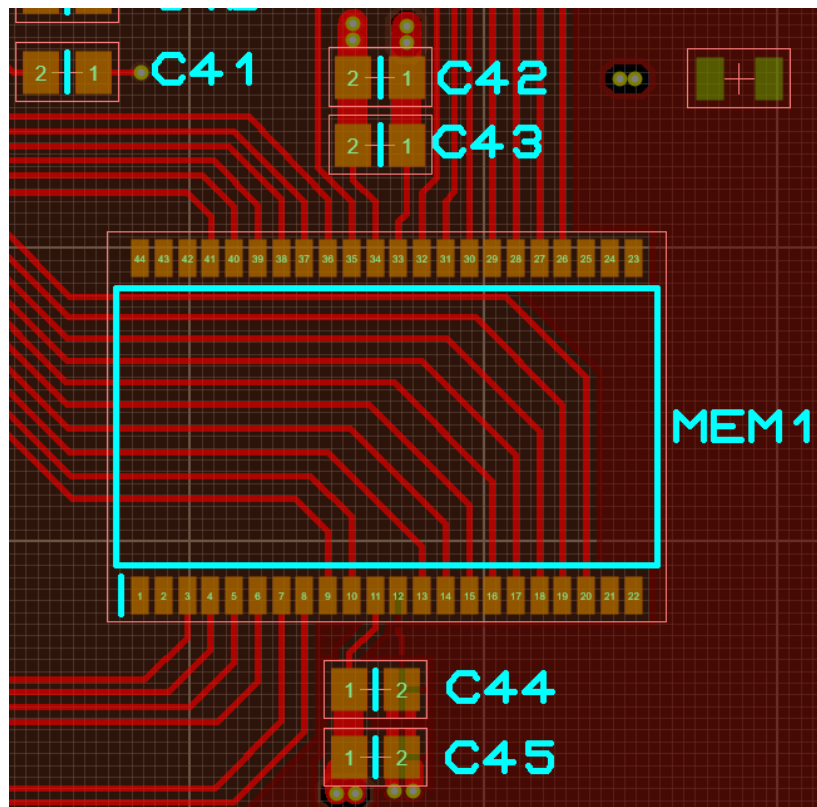


Figura 17 – Diseño de la memoria SRAM en la placa de circuito impreso.

### 5.2.5 LEDs indicadores de estado y puntos de prueba

En la Figura 18 se muestran las conexiones con el led que indica cuando la configuración de la FPGA está lista y el led indicador de falla de paridad.

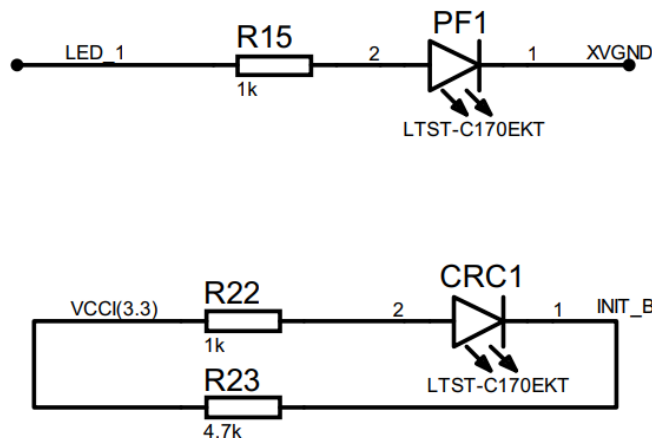


Figura 18 – LEDs indicadores de falla de paridad y configuración de la FPGA.

Por otro lado, con el objetivo de facilitar la tarea de mantenimiento sobre este diseño, se colocaron puntos de prueba de tensión a la entrada y salida de cada regulador como se muestra en la Figura 19.

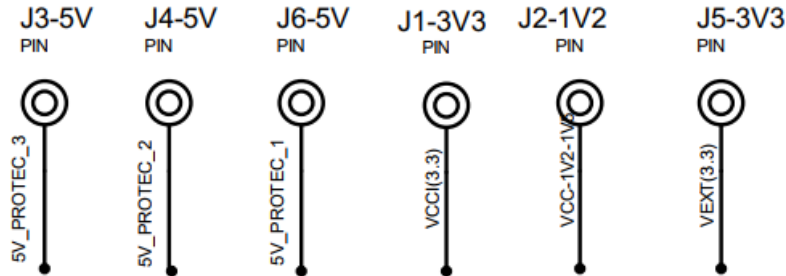


Figura 19 – Puntos de prueba para facilitar las tareas de mantenimiento.

### 5.3 Conexión con el G-PPAL-BUS

Debido a que el integrado de memoria SRAM y la FPGA funcionan con una tensión de alimentación entre 2,7 V ~ 3,6 V, es necesario realizar una adaptación para poder conectar el bloque con el bus del sistema, el cual utiliza un voltaje de 5 V. Se utilizaron 9 transceptores bidireccionales 74LVTH2245 para la mayoría de las conexiones: 4 para las entradas a la memoria, 2 para las salidas de la memoria y 3 para las señales bidireccionales. Además, un SN74LVC07A para las salidas que necesitan un adaptador del tipo *open collector*.

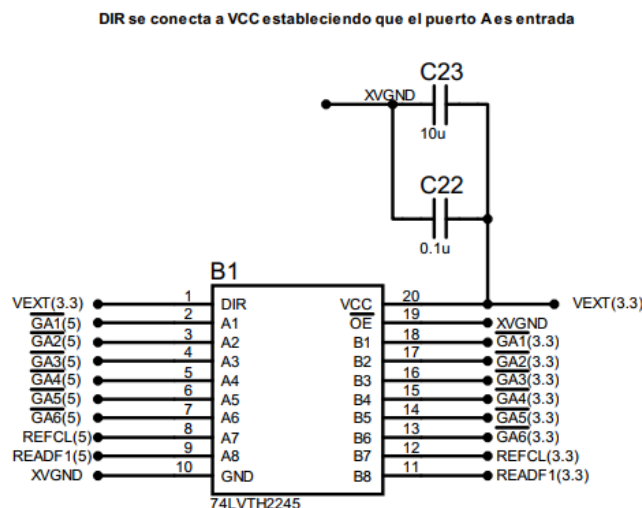


Figura 20 – Configuración del transceptor 74LVTH2245 como entrada a la memoria.

Los buffers fueron agrupados según el sentido de las señales que adaptan y en

estos el pin que controla la dirección de adaptación fue fijado a un '1' lógico o '0' lógico según el comportamiento deseado. Para el caso de las señales bidireccionales, este pin, al igual que el que habilita el funcionamiento, es controlado por lógica del diseño digital implementada sobre la FPGA. En la Figura 20 se muestra la configuración del buffer bidireccional 74LVTH2245 como entrada a la memoria, en la Figura 21 como salida de la memoria y en la Figura 22 en modo bidireccional. Por otro lado, en la Figura 23 se muestra la configuración del transceptor SN74LVC07A.

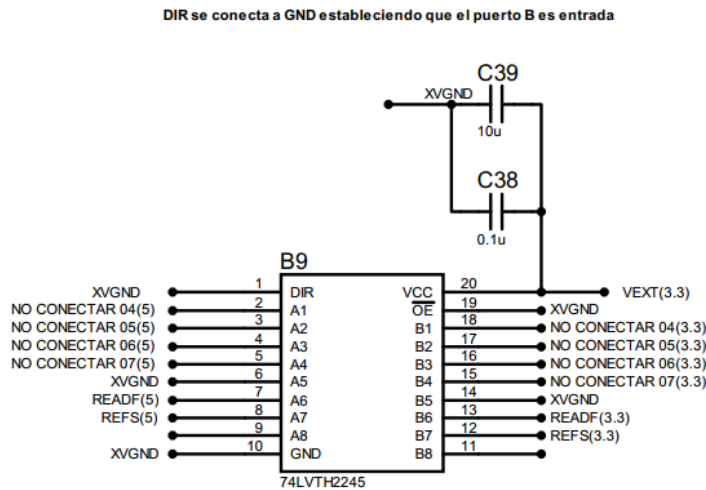


Figura 21 – Configuración del transceptor 74LVTH2245 como salida de la memoria.

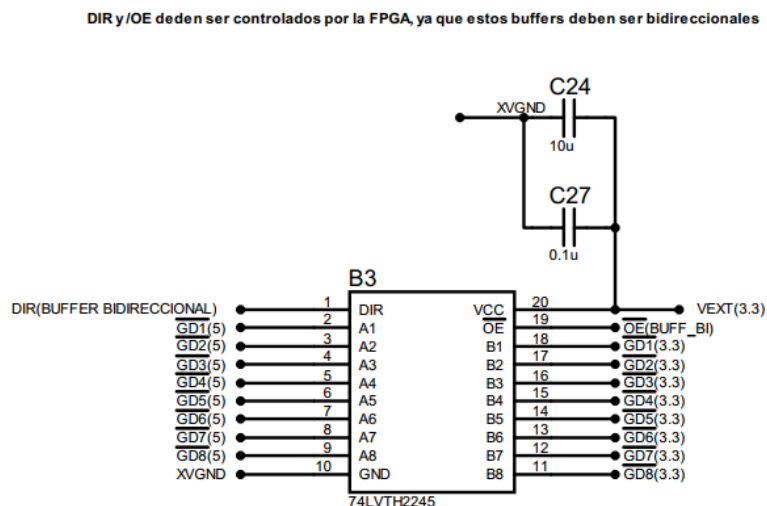


Figura 22 – Configuración bidireccional del transceptor 74LVTH224, controlado por la FPGA.



## 5.4 Características generales del PCB

El PCB del proyecto RAM-MU64 posee 4 capas de cobre, cuya distribución o layer stack up (Salmony, 2022) se observa en la Tabla III.

Tabla III – Características físicas de las capas del circuito impreso del proyecto RAM-MU64.

ID	Nombre	Tipo	Material	Grosor	Dieléctrico
TR	Top Resist	Superficie	Mascara antisoldante	0.01 mm	3,5
TOP	Top Cooper	Señal	Cobre	0.036 mm	
		Prepreg	FR4	0.32 mm	4,8
I1	Inner 1	Señal	Cobre	0.0036 mm	
		Core	FR4	0.772 mm	4,8
I2	Inner 4	Señal	Cobre	0.036 mm	
		Prepreg	FR4	0.32 mm	4,8
BOT	Bottom Copper	Señal	Cobre	0.0036 mm	
BR	Bottom Resist	Superficie	Mascara antisoldante	0.01 mm	3,5
<b>Grosor total</b>				1.575 mm	

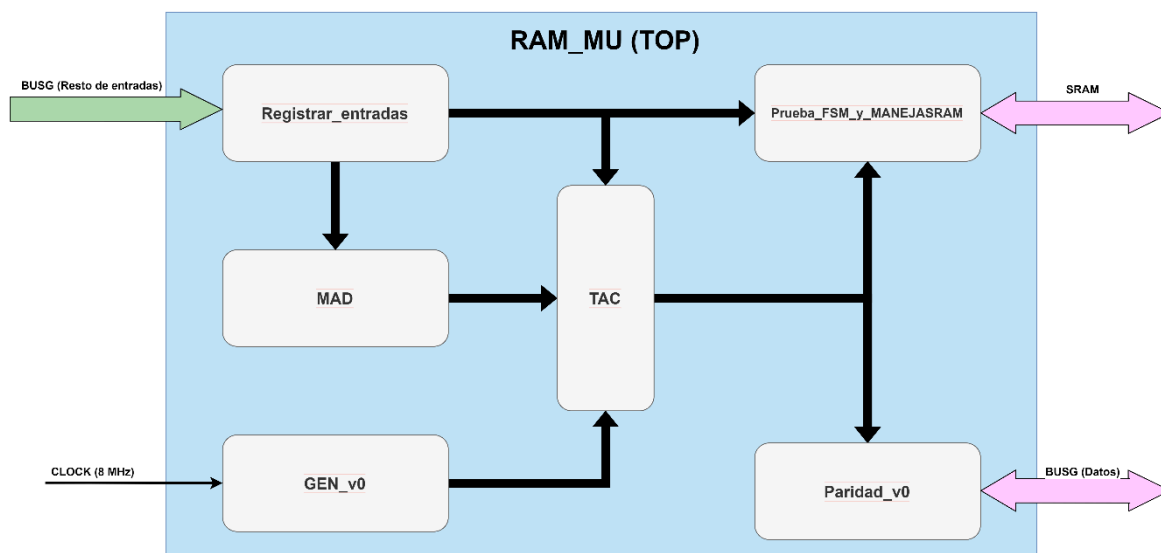
Al igual que en la versión 4, la capa superior posee la mayoría de los componentes y pistas de señales más las conexiones para las fuentes de alimentación. En cambio, el plano de tierra se lo dejó sólido para que cumpla de manera correcta con su trabajo. Respecto a la capa de alimentación, se divide en varios planos internos para distribuir los distintos voltajes entre los componentes y algunas pistas para complementar. Finalmente, en la capa inferior se encuentran algunos capacitores de acople y las resistencias de configuración de la FPGA más algunas pistas extra que no pudieron ser dibujadas en la capa superior.

Para todos los componentes se utilizaron footprints compatibles con las recomendaciones IPC (7351B Table of Contents, 2010). De esta manera, se asegura cierto nivel de estandarización para el proceso de fabricación y montaje de las placas, se mejora la calidad del diseño y se reduce el riesgo de inconvenientes con el fabricante. Como esta versión es una continuación de las anteriores, a la hora de realizar el cambio de los footprints se tuvo que redibujar cada pista. Además, se aseguró que todas las conexiones sean rectas y las esquinas de las señales de alta frecuencia sean a 45°. También se siguieron otras recomendaciones presentes en estas normas IPC como orientar a los componentes de la misma naturaleza en un mismo sentido, la forma de empalmar pistas que conectan varios pads a la vez y no colocar vías debajo de los componentes cuando fuera posible.

En cuanto a compatibilidad electromagnética, se realizaron las pistas teniendo en cuenta las corrientes de retorno e intentando que estos caminos sean lo más cortos posible y de baja inductancia a altas frecuencias y baja resistencia a bajas frecuencias. Además, tanto en el plano superior como en el inferior se rellenaron los espacios entre pistas con cobre para generar capacitores distribuidos y disminuir el efecto del ruido electromagnético externo sobre el diseño.

## 6. Diseño digital

El objetivo principal del diseño digital de este proyecto es ampliar la capacidad de memoria de los prototipos anteriores y de esta manera reemplazar, cuando sea posible, cuatro módulos de memoria originales de 16k24 palabras por un solo módulo que maneje 64k24 palabras. Dado que, en ocasiones especiales como en la unidad paginadora, es necesario contar con un módulo de 16k24, debe ser posible seleccionar manualmente el



*Figura 25 – Estructura del diseño digital v10 (RAM-MU16).*

modo de funcionamiento a través de un mismo diseño. Además de realizar las modificaciones de comportamiento necesarias, a partir del diseño digital anterior, se verificó el funcionamiento del código y se agregaron algunas mejoras pertinentes.

Junto con el análisis de los prototipos anteriores, más la documentación de la memoria original, fue posible estimar los cambios necesarios para cumplir con el objetivo de este proyecto. En la Figura 25 se muestra el diagrama en bloques del diseño digital del cual se partió, la versión 10 del diseño digital del proyecto RAM-MU.

El módulo principal o top module, agrupa e interconecta los distintos componentes que hacen al funcionamiento de la memoria, más el control de los buffers bidireccionales que realizan la traslación de niveles para las señales de datos. A continuación, se describe brevemente la función de cada uno:

- **Registrar\_entradas:** en este módulo se adaptan y sincronizan las señales de entrada para usarlas en el resto del diseño.
- **GEN:** este módulo se encarga de adaptar y sincronizar el flanco del reloj de 8 MHz proveniente del G-PPAL-BUS con el reloj interno de la FPGA. La señal sincronizada es la que se utiliza para los ciclos de memoria.
- **MAD:** el Memory Address Device identifica a qué módulo de memoria en particular son dirigidas las acciones ejecutadas por el procesador.

- TAC: el circuito de sincronización y control (TAC) procesa las señales necesarias para secuenciar los distintos ciclos de la memoria que se envían como señal de handshake a la unidad que está utilizando la memoria.
- Paridad\_v0: genera los bits de paridad correspondientes para la escritura de datos; verifica si existen fallos de paridad en lectura y genera la señal de error de paridad.
- Prueba\_FSM\_y\_MANEJASRAM: este componente se encarga de administrar los datos que se almacenan en la memoria y, por ende, del manejo del integrado de memoria SRAM. El módulo se compone de dos componentes adicionales: una máquina de estados y una implementación de registros de datos. Estos dos elementos trabajan en conjunto para realizar la descomposición de la palabra de 32 bits proveniente del bloque de paridad. La máquina de estados controla el proceso de descomposición, mientras que los registros de datos almacenan la información temporal necesaria.

## 6.1 Verificación del diseño digital RAM-MU16

La verificación del diseño digital RAM-MU16 fue realizada para obtener una verificación adecuada del código y, a partir de allí, testbenches unitarios (para cada módulo) que sirvan para contrastar el funcionamiento del nuevo diseño. De esta manera se asegura que el comportamiento se mantenga entre versiones. La bibliografía que se utilizó como guía fue *Writing Testbenches Functional Verification Of Hdl Models* (Bergeron, 2003, #).

Lo primero que se diagramó fue el plan de verificación. Para cada módulo se detallaron los aspectos clave de interés y la forma de llevar adelante la verificación. Luego se desarrollaron los testbenches y se realizaron los ajustes necesarios para que funcionen de manera correcta. Una vez que el comportamiento era el esperado, se observaron las estadísticas de cobertura de código (*Collecting Code Coverage in Active-HDL - Application Notes - Documentation - Resources - Support - Aldec*, n.d.) que permite cuantificar de manera porcentual el nivel de la verificación y se pulieron los testbenches para tener un nivel de cobertura aceptable.

### a) Módulo Registrar\_entradas.vhd

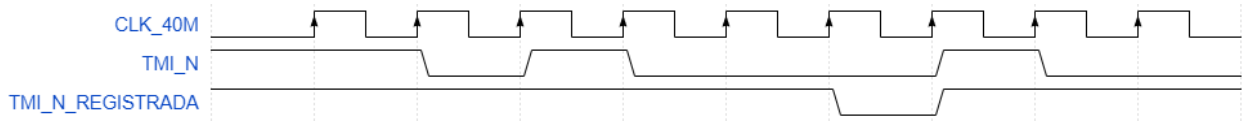
El módulo registrar entradas se encarga de capturar los estímulos externos a la FPGA y actúa como filtro para el resto del diseño. Su objetivo es eliminar posibles glitches que afecten el funcionamiento del modelo.

Para desacoplar la entrada del resto del diseño, se agregan diversos registros a la entrada y luego se compara la salida de cada uno de los registros con una compuerta OR en caso de lógica '0' como verdadero y con ANDs en las entradas de lógica '1' como verdadero. De esta manera, se asegura que la entrada no cause inestabilidad en el resto del diseño.

La forma de comprobar este módulo es con señales de estímulo ruidosas y comprobar que los valores de salida cambian de acuerdo con el funcionamiento esperado.

El testbench se puede encontrar en el anexo A.1. Al realizar estas pruebas se comprobó que en los escenarios ideales las salidas se comportan como era esperable, el problema radica cuando se inyectan señales ruidosas en los pines que filtran solo por OR o solo por AND. Es decir, que, si aparece un solo ruido de lógica negativa, afecta en el

siguiente flanco reloj a la señal “filtrada”. Esto puede llegar a ser muy problemático, sobre todo en pines críticos como TMI\_N, que es clave para el handshake entre la computadora y la memoria. En la figura 26, el marcador muestra el momento en que un pequeño ruido hace cambiar el estado de TMI\_N\_REGISTRADO. Es por eso que este aspecto se tendrá en consideración para el nuevo diseño.

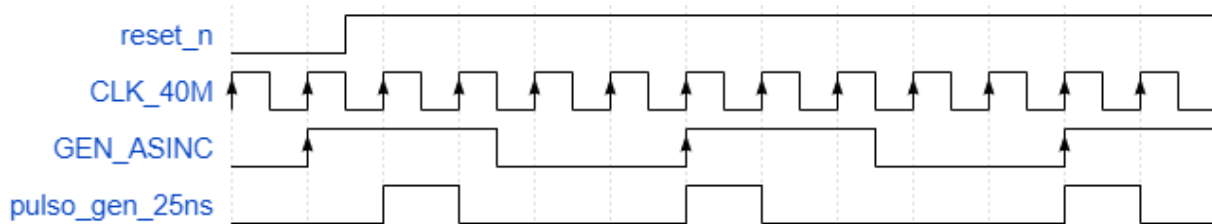


*Figura 26 – Muestra de la asimetría del filtro de entrada para TMI\_N.*

### b) Módulo Gen\_v0

El módulo *Gen\_v0.vhd* se encarga de sincronizar el reloj de 8MHz externo al modelo con el reloj de 40 MHz, la configuración utilizada en el proyecto RAM-MU16. Este tipo de módulos son críticos para el funcionamiento correcto del diseño, por lo que es importante realizar una verificación exhaustiva.

Las pruebas se pueden realizar desfasando los relojes utilizando una fuente de retardo. El módulo se comprueba utilizando una compuerta AND para asegurarse de que los dos relojes están sincronizados.



*Figura 27 – Sincronización del reloj de la SMR-MU con el de la FPGA.*

El testbench se puede encontrar en el anexo A.2. En este caso el comportamiento fue el esperado y no hay cambios para hacer en el nuevo diseño. Primero se simularon ambos relojes de manera simultánea y se modificó la señal RESET antes, después y durante un flanco del reloj principal (CLK\_40). La siguiente prueba que se realizó fue adelantando y atrasando el reloj de 8 MHz respecto al de 40 MHz. La última prueba fue hacer coincidir el flanco ascendente de un reloj con el descendente del otro, es decir, en la peor condición de desfase.

### c) Módulo MAD

El módulo MAD se encarga de iniciar el control y temporizado TAC. Este componente discrimina entre los módulos de memoria utilizando los pines DA4\_1 y GA16\_13. Además, la entrada MIH (memory inhibit) es utilizada por la computadora para deshabilitar la memoria en ciertas operaciones. Para verificar el módulo MAD, se utilizará una tabla de

verdad que incluya todas las combinaciones de entrada. Además, se comprobará el funcionamiento del módulo ante distintos desfases y valores de MIH. Por último, se probará el sistema de filtrado de MIH\_N, aunque al ser el mismo que el de los módulos *registrar\_entradas*, el comportamiento debe ser el mismo.

El testbench se puede encontrar en el Anexo A.3. La lógica descrita funciona correctamente, de acuerdo con lo especificado por la documentación de la memoria original. La siguiente prueba que se realizó fue manteniendo los valores de entrada fijos y conmutando MIH\_N, para verificar el filtro digital. Este último resultado fue igual al del módulo *registrar\_entradas*. En la Figura 28 se muestra una captura del testbench correspondiente.

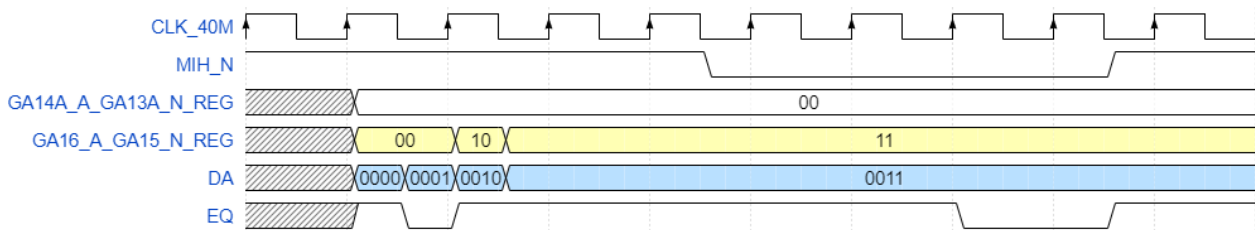


Figura 28 – Testbench del módulo MAD\_v0.

#### d) Módulo TAC

El módulo de temporización y control (TAC) produce las señales necesarias para secuenciar los distintos ciclos de la memoria, además de la señal RDY, que forma parte del "handshake" entre la unidad maestra que utiliza la memoria y la memoria. En este bloque se detecta cuando se inicia un ciclo de memoria, que puede ser de escritura o lectura. Además, genera señales que controlan los buffers tri-state de manera directa o que les indican a otros módulos cuando es seguro interactuar con los datos del BUS-PPAL.

La documentación de la memoria original (*Documentación Confidencial - ARA*, n.d.) incluye una explicación del procedimiento del "handshake". En ella, se muestran diagramas de secuencias con desfases entre las señales y los requerimientos temporales para cada situación. En la figura 29 se muestra la secuencia del ciclo de escritura a manera de ejemplo. Por lo tanto, esta información fue utilizada para realizar los testbenches y determinar si se cumple con los tiempos de respuesta mínimos.

El testbench se encuentra en el anexo A.4. Posee cuatro rutinas principales, dos ciclos de lectura y escritura con el tiempo mínimo y máximo de desfase especificado. Dado que esta es una prueba aislada, se deben introducir los retrasos en las señales pertinentes, como en TMI\_N, producto de los filtros digitales. Así, se puede obtener el tiempo de respuesta real de todo el diseño. En los cuatro casos los resultados fueron satisfactorios. La figura 30 muestra el caso en donde el desfase es de 85 ns para un ciclo de lectura, el tiempo de respuesta fue de 261 ns, mayor que el mínimo de 250 y menor que el máximo de 375 ns.

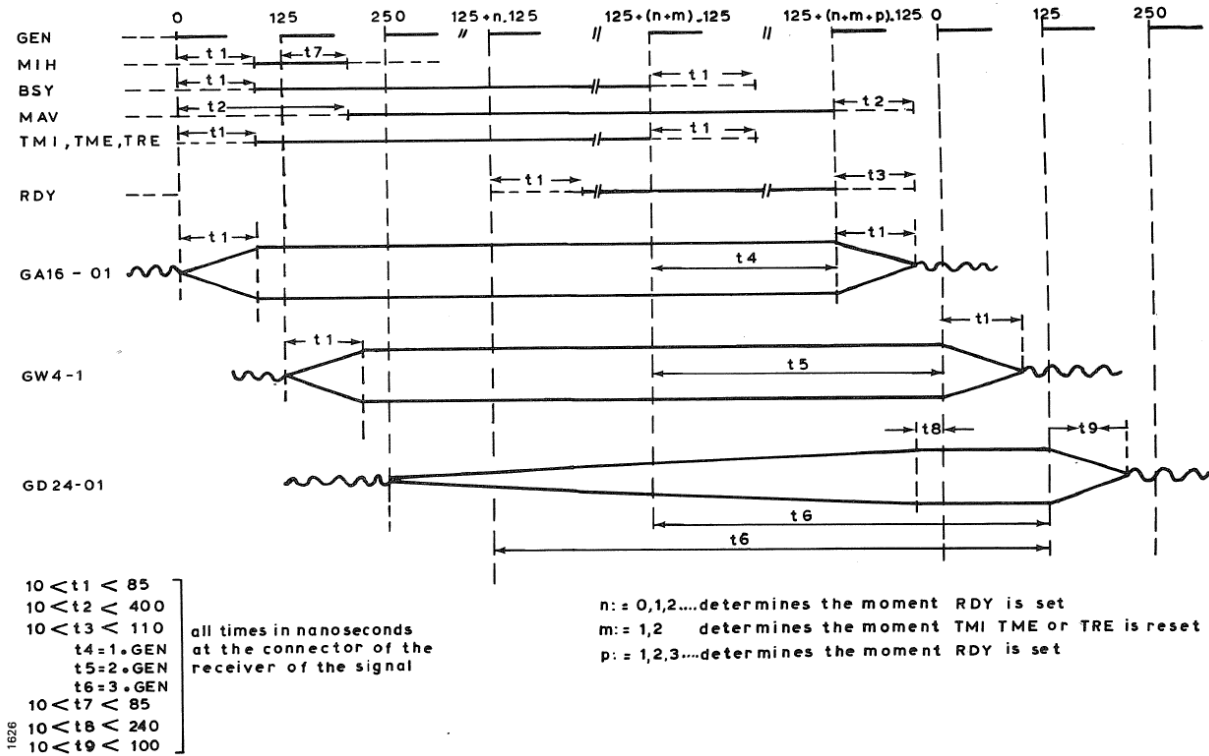


Figura 29 – Requerimientos temporales del handshake SMR-MU.

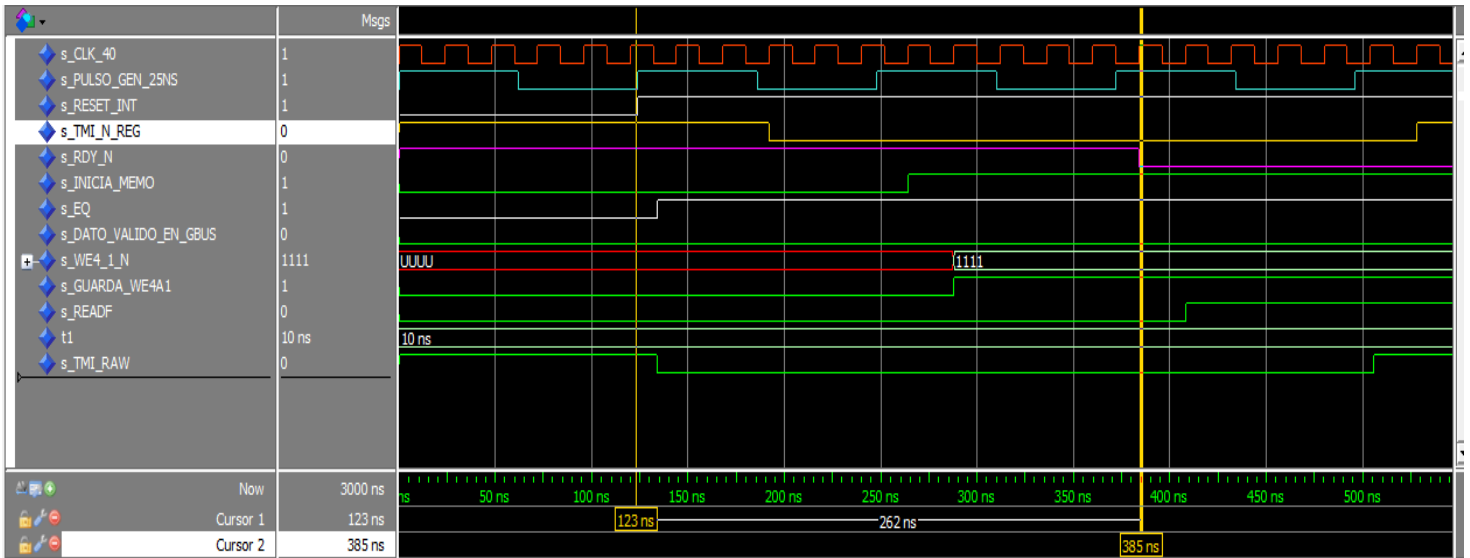


Figura 30 – Testbench del módulo TAC\_v0.

### e) Módulo Paridad\_v0

Este módulo genera los bits de paridad escritos en la memoria RAM y comprueba los bits de paridad leídos desde la memoria RAM, indicando un error de paridad si los datos no concuerdan. La unidad maestra que controla la memoria indica si se debe utilizar paridad par o impar. Posee tres componentes (*GEN\_PAR\_v0* (x2) y *PFF\_v0*) que se instancian y conectan entre sí. Los dos módulos de *GEN\_PAR\_v0* se utilizan para generar los bits de paridad tanto de los datos presentes en el BUS-PPAL como los guardados en la memoria RAM. El módulo *PFF\_v0* describe al flip-flop que genera la señal de fallo de paridad.

Para verificar este módulo se deben agregar datos con distintas paridades para asegurar que funcione correctamente ante cualquier combinación. En el modo de lectura se debe comprobar que los bits de paridad generados concuerden con los datos guardados. En el de escritura, solo se deben generar los bits de paridad a partir de los datos del BUS-PPAL.

El testbench utilizado para comprobar el módulo de *Paridad\_v0* se encuentra en el anexo A5. Aquí no se encontró ningún error y los cálculos de paridad son realizados de manera correcta. Se creó un script en Python para generar datos aleatorios de escritura y utilizar estos valores para validar el diseño VHDL que se encuentra en el anexo A.6. La figura 31 muestra el funcionamiento para un ciclo de lectura.

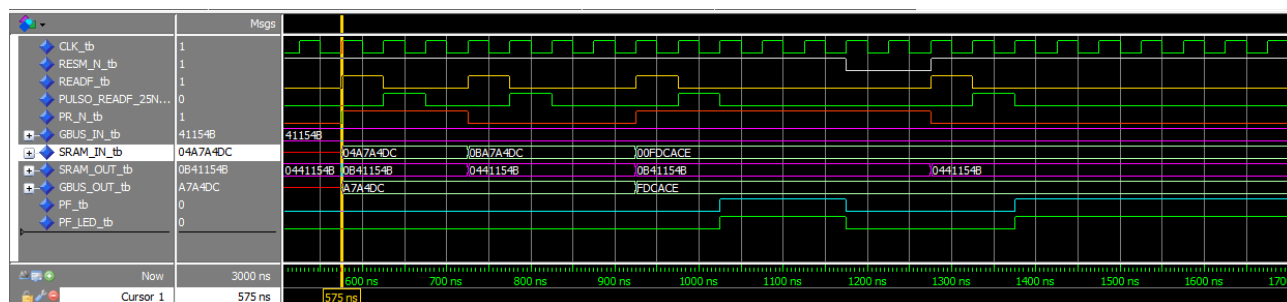


Figura 31 – Testbench durante un ciclo de lectura del módulo *Paridad\_v0.vhd*.

#### *Gen\_par\_v0.vhd*

El módulo *Gen\_par\_v0.vhd* sintetiza un circuito concurrente que genera paridad a partir de los 24 datos del BUS-PPAL originando 4 bits de paridad cada 6 bits y los agrega en formato de 32 bit para que sea posible entregarlo a la SRAM.

La verificación de este diseño es un caso particular del módulo *paridad\_v0*, salvo que en este caso solo se comprueba la generación de los bits de paridad de manera aislada.

El testbench se puede consultar en el anexo A.7. En este módulo tampoco se halló ningún problema.

#### *PFF\_v0.vhd*

Este diseño es un parity flip-flop y genera la señal fallo de paridad PF (parity fault) para indicar a la unidad PC\_NAVAL y PF\_LED como indicador de error en la placa hacia el operario. Para ello, detecta, ante un flanco descendente de *READF*, si existe un fallo de paridad y genera la señal correspondiente. La indicación del fallo de paridad sólo se dispara en un ciclo de lectura.

La verificación de este módulo, al igual que gen\_par\_v0, ya fue realizada en el testbench Paridad\_v0. No obstante, también debe realizarse un testbench para este módulo simulando ciclos de lectura y escritura con y sin fallos de paridad.

El anexo A.8 posee el testbench utilizado para comprobar el módulo PFF\_V0.

### f) Módulo Prueba\_FSM\_y\_MANEJASRAM

Este módulo se encarga de administrar los datos que se almacenan en la memoria y, por ende, del manejo del integrado de memoria SRAM. Está compuesto por dos componentes: una máquina de 18 estados llamado *FSM\_LEEYESCRIBE4B.vhd* y una implementación de registros de datos que contiene toda la lógica necesaria para realizar la descomposición de la palabra de 32 bits (24 bits de dato + 4 bits de paridad + 4 bits en '0') llamado *HW\_REGISTROS\_IO\_SRAM.vhd*. Este diseño no agrega lógica adicional, sólo interconecta las entradas y salidas pertinentes a los componentes internos y externos.

Debido a la interdependencia de los componentes, se decidió realizar un testbench para prueba\_fsm\_y\_maejaSRAM que compruebe todo el funcionamiento de manera integrada. Esto le agrega complejidad al testbench (más largo y con más aserciones) en comparación con los demás. Los aspectos que se deben verificar son la descripción de la máquina de estados, los valores de las señales que habilitan la lectura o escritura del chip SRAM, la dirección a la que se escribe o lee la memoria y la correcta descomposición de la palabra de 32 bits.

El Anexo A.9 muestra el testbench del módulo. Primero se comprobó un ciclo de escritura, luego uno de lectura y finalmente una escritura parcial. En un ciclo de escritura, primero se lee el dato almacenado en SRAM y luego se escribe la memoria SRAM con el nuevo dato. La FSM fue diseñada de esta manera en el proyecto RAM-MU16 para utilizar la misma lógica tanto en escrituras parciales como en totales. La Figura 32 muestra el comportamiento del módulo ante un ciclo de escritura parcial de la palabra inferior.

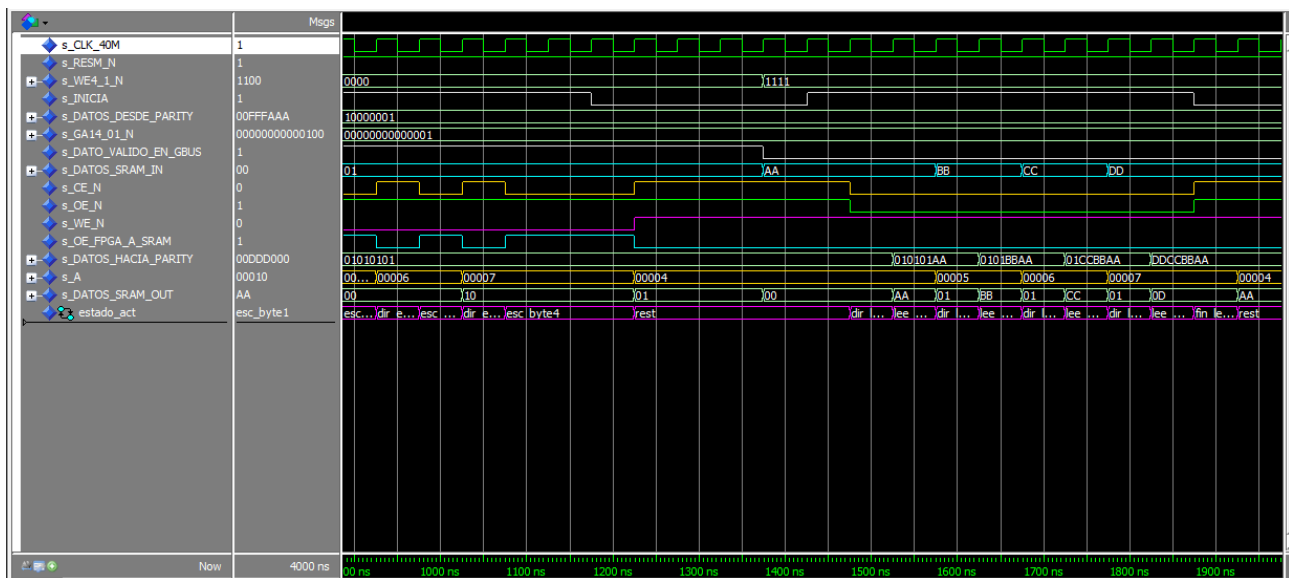


Figura 32 – Ciclo de escritura parcial en el módulo Prueba\_FSM\_y\_MANEJASRAM.

Dado que aquí se manejan el **buffer bidireccional y la memoria SRAM**, este componente debe modificarse para el proyecto RAM-MU64. Como la FSM se realizó de manera correcta con la transición de estados por un lado y la acción por otro, lo que se debe cambiar para la nueva versión 64k28 es la forma de direccionar la memoria; se debe agregar un bit más.

### g) Módulo RAM\_MU (archivo top)

Finalmente se realizaron las verificaciones sobre el archivo top del diseño digital, es decir aquel que contiene todos los componentes anteriormente mencionados y engloba el funcionamiento de todo el diseño. Las pruebas realizadas por este testbench son de integración, más que pruebas unitarias. Se llama integración porque involucran el funcionamiento en conjunto de los componentes digitales.

Para probar este módulo, se debe verificar que los ciclos de memoria sean ejecutados bajo los requerimientos de la memoria original. Para ello se deben realizar ciclos de escritura total, escritura parcial y lectura. La base de estas pruebas es la comprobación del módulo MAD, TAC y LEEYESCRIBU. Estos son los módulos más críticos ya que generan las señales relacionadas con el acceso a la memoria y el protocolo de intercambio de información.

Como punto de partida se utilizaron testbenches realizados en el proyecto RAM-MU16. A partir de allí se realizaron tres procedimientos los cuales engloban las acciones del G-PPAL-BUS y permiten intercalar ciclos de escritura y lectura de manera sencilla. En cada ciclo lo que se pretende observar es que los tiempos máximos y mínimos de acción se respeten y que los datos que se guardan en la SRAM sean los correctos. Los testbenches se encuentran en el anexo A.10. A continuación se hace un análisis un poco más detallado de cada ciclo de memoria.

#### Ciclo de escritura total

Las formas de onda que se muestran en la Figura 33 permiten comprobar visualmente el funcionamiento del diseño.

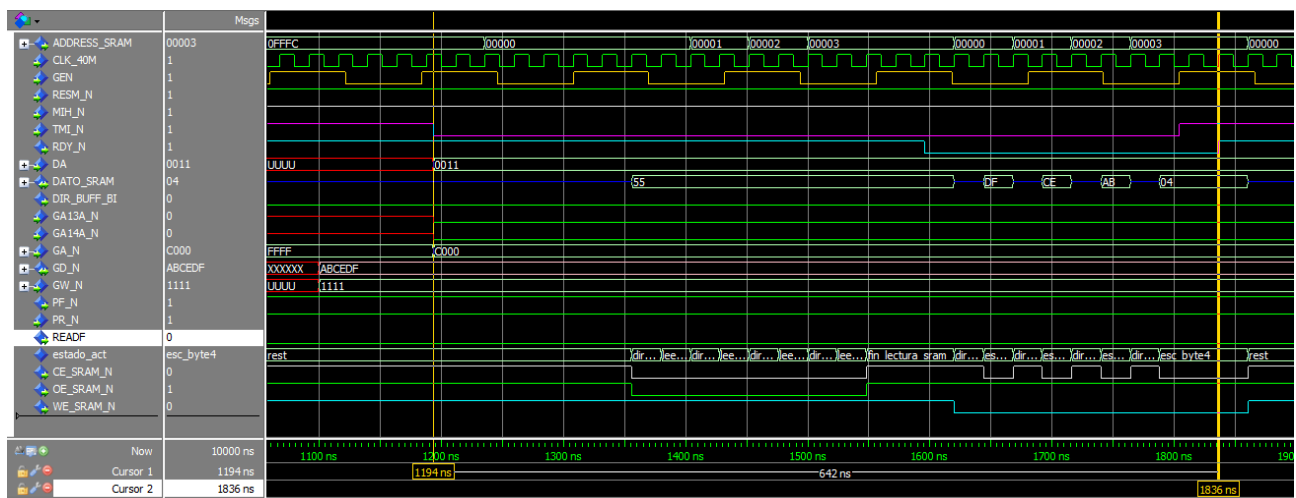


Figura 33 – Ciclo de escritura total del testbench RAM-MU (top).

El ciclo comienza a los 1194nS, momento en donde  $TMI\_N$  pasa de '1' a '0' y la palabra  $GW\_N = "1111"$ . En este instante se generan las señales para indicar a la FSM que un ciclo de memoria fue iniciado. El valor por escribir en memoria es  $0xABCDEF$ , el asignado a  $GD\_N$ . Una vez que la FSM sale del estado *rest*, comienza a leer el dato presente en la SRAM; que es direccionada por  $GA\_N$ . Una vez finalizado el proceso de lectura, comienza el proceso de escritura, en donde se guarda la palabra de 32 bits en 4 partes de 8bits. Este proceso comienza con los 8 bits menos significativos hasta los más significativos ( $0xDF - 0xCE - 0xAB$ ). También se puede apreciar como la dirección de la memoria SRAM,  $ADDRESS\_SRAM$ , va variando cada dos transiciones de estado para alocar de manera correcta cada dato. Finalmente se guardan los bits de paridad generados a partir del dato en el BUS-PPAL ( $0x04$ ).

### Ciclo lectura

El ciclo de lectura inicia cuando la señal  $TMI$  pasa de '1' a '0' y la palabra  $GW\_N = "1111"$ . Debido al filtro anti glitch, el sistema comienza a actuar tres flancos reloj después del cambio en la entrada ( $t=75$  nS). En este momento el bloque TAC es el que se encarga de determinar si la acción debe llevarse a cabo. Para ello, utiliza la señal EQ proveniente del bloque MAD y los relojes de 40MHz y 80 MHz, en aproximadamente dos ciclos reloj. Luego de esto la señal que inicia la FSM *inicia\_memo* pasa a '1' ( $t=125$ nS). Inmediatamente después del cambio de *inicia\_memo*, comienza el ciclo de lectura de la FSM que dura ocho ciclos ( $t=325$ nS). Un ciclo reloj después, la señal  $RDY\_N$  pasa a '0' ( $t=350$ nS). Teniendo en cuenta el diagrama provisto por el fabricante y la referencia con los tiempos límites se interpreta que, bajo la actual implementación, ( $t_1=42,5$ ) el modelo posee una constante  $n=2$ .

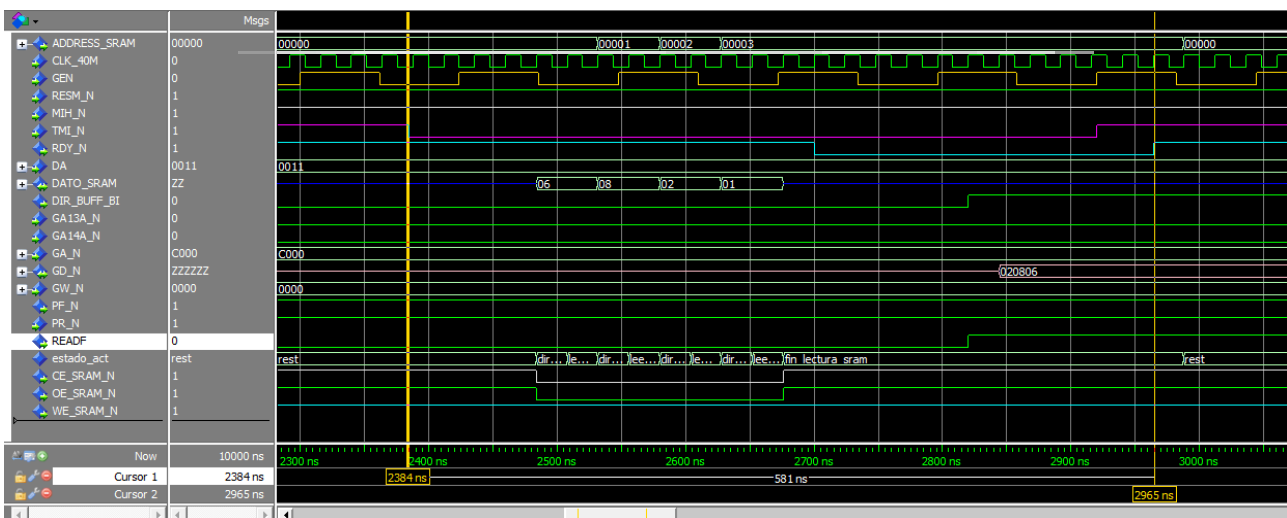


Figura 34 – Ciclo de lectura del testbench RAM-MU (top).

Luego se espera a que el maestro haga  $TMI\_N$  '1'. En ese instante, la señal  $READF$  se hace '1' y un ciclo reloj de 8MHz más tarde  $RDY\_N = '1'$  finalizando el ciclo de lectura.

Analizando todo esto se concluye que el ciclo de lectura en memoria se ejecuta de manera correcta en condiciones normales con un tiempo de respuesta menor a 750 nS.

Además, la falla de paridad funciona como lo esperado.

### Ciclo de escrituras parciales

Al realizar las escrituras parciales puede notarse el efecto del bloque HW\_IO\_SRAM, ya que los datos que son enviados desde el bloque PARIDAD\_V0 difieren a los que se presentan en la SRAM. El valor que se escribe en la SRAM se obtiene de concatenar las 4 salidas de 8 bits en DATOS\_SRAM\_OUT al momento en que la FSM se encuentra en los estados ESC\_BYTE\_1-4. Para asegurar que el valor leído por el bloque parity en la dirección a la que se intenta escribir sea el predeterminado en el testbench, también se muestran las señales S\_FFO\_SRAM\_1-4. A continuación se muestran distintas tablas con los valores utilizados:

### Media palabra superior GW = 0011

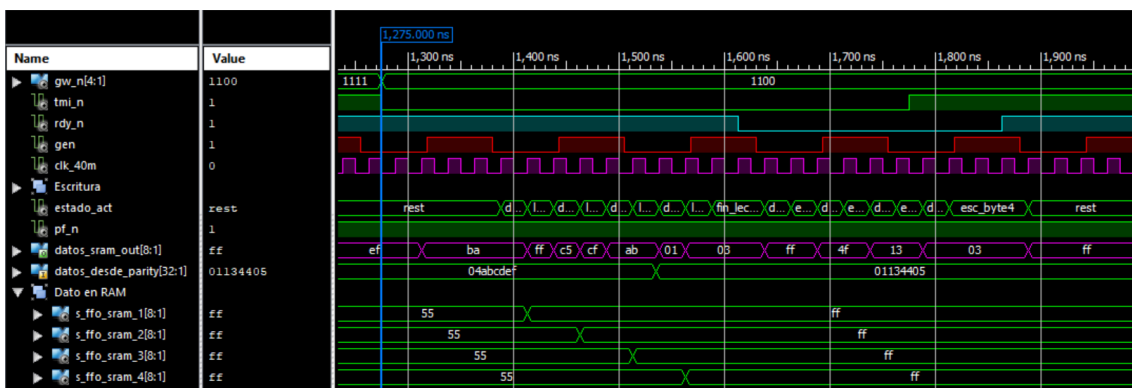
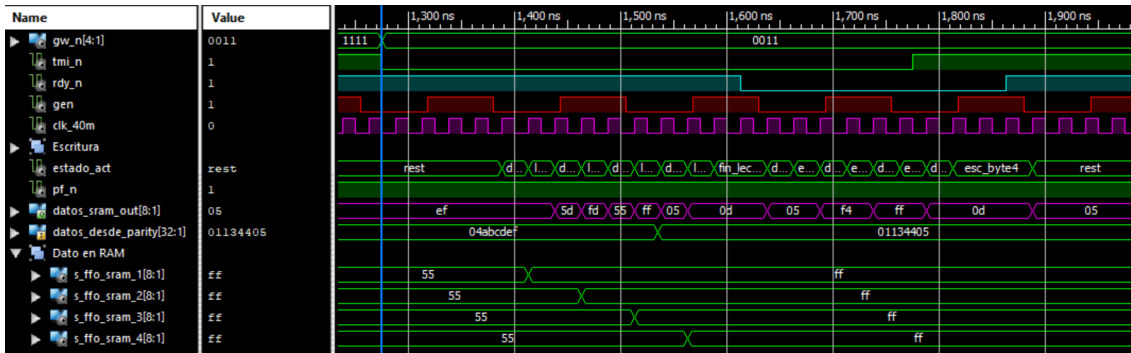


Figura 35 – Ciclo de escritura parcial de media palabra superior del testbench RAM-MU (top).

Tabla IV: Resultado del ciclo de escritura parcial de media palabra superior:

Valor actual (SRAM)	0xFF FF FF
Valor por escribir (BUS-PPAL)	0x13 44 05
Valor en memoria esperado (SRAM)	0x13 4F FF
Valor en memoria obtenido	0x 13 4F FF CORRECTO

*Media palabra inferior GW = 1100*

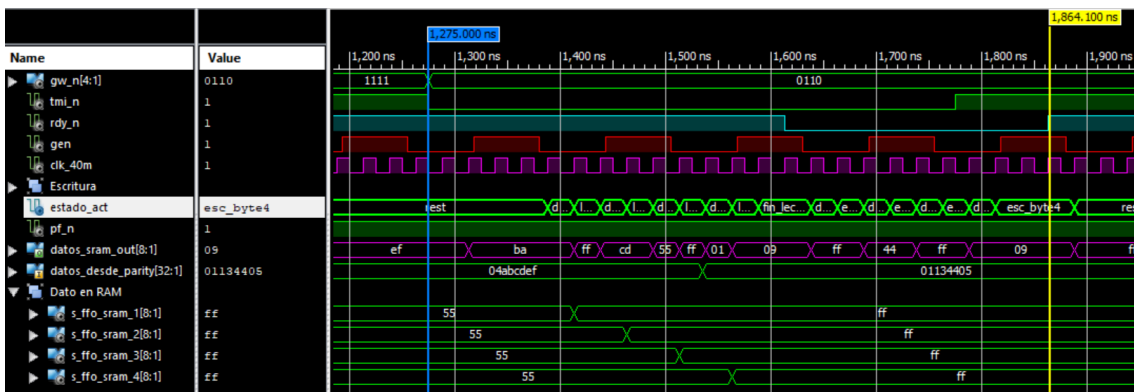


*Figura 36 – Ciclo de escritura parcial de media palabra inferior del testbench RAM-MU (top).*

*Tabla V: Resultado del ciclo de escritura parcial de media palabra inferior:*

Valor en memoria (actual)	0xFF FF FF
Valor por escribir	0x13 44 05
Valor en memoria esperado (final)	0xFF F4 05
Valor en memoria obtenido	0x FF F4 05 CORRECTO

*Media palabra del medio GW = 1001*

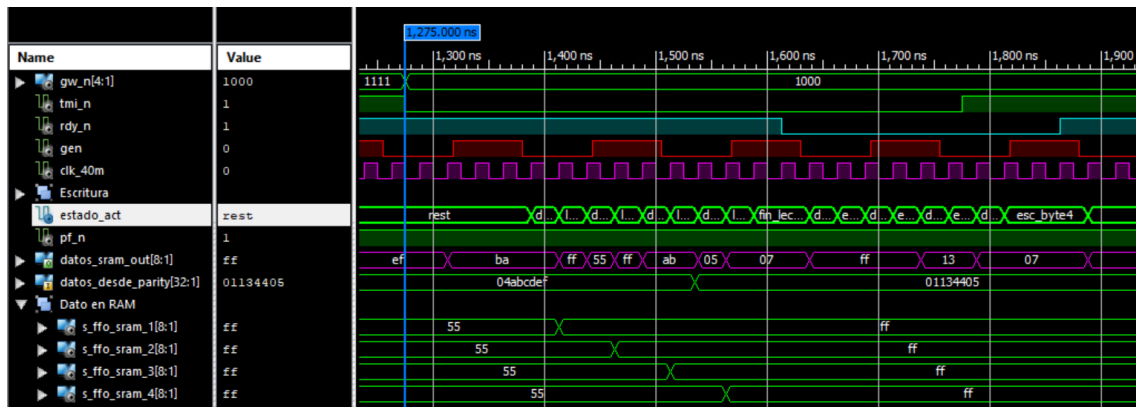


*Figura 37 – Ciclo de escritura parcial de la media palabra del testbench RAM-MU (top).*

*Tabla VI: Resultado del ciclo de escritura parcial de media palabra del medio:*

Valor en memoria (actual)	0xFF FF FF
Valor por escribir	0x13 44 05
Valor en memoria esperado (final)	0xFF 44 FF
Valor en memoria obtenido	0x FF 44 FF CORRECTO

*Carácter superior GW = 0111*

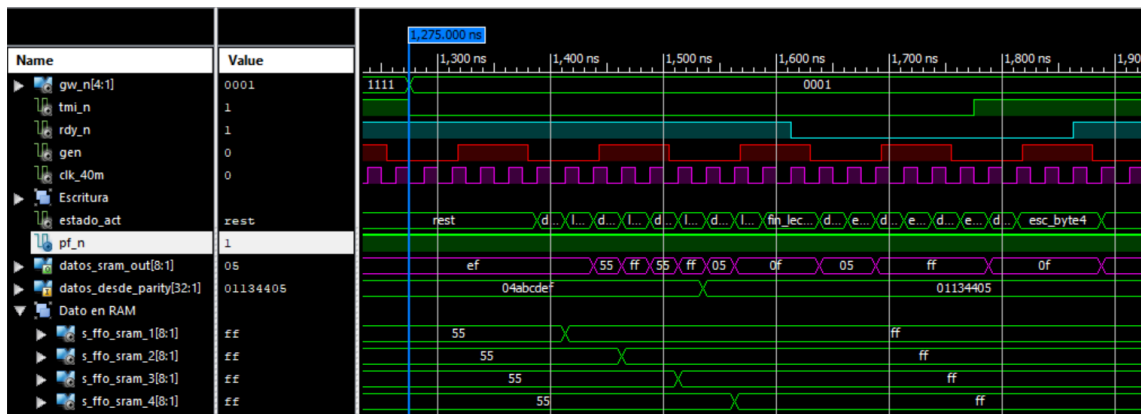


*Figura 38 – Ciclo de escritura parcial de carácter superior del testbench RAM-MU (top).*

*Tabla VII: Resultado del ciclo de escritura parcial de carácter superior:*

Valor en memoria (actual)	0xFF FF FF
Valor por escribir	0x13 44 05
Valor en memoria esperado (final)	0x13 FF FF
Valor en memoria obtenido	0x 13 FF FF CORRECTO

*Carácter inferior GW = 1110*



*Figura 39 – Ciclo de escritura parcial de carácter inferior del testbench RAM-MU (top).*

*Tabla VIII: Resultado del ciclo de escritura parcial de carácter inferior:*

Valor en memoria (actual)	0xFF FF FF
Valor por escribir	0x13 44 05
Valor en memoria esperado (final)	0x FF FF 05
Valor en memoria obtenido	0x FF FF 05 CORRECTO

## Análisis de cobertura de código

Finalmente, para obtener una visión cuantitativa de la efectividad del testbench implementado, se generó un detallado informe de cobertura de código (Code Coverage Analysis) utilizando Modelsim (*ModelSim HDL Simulator | Siemens Software, n.d.*). Este reporte no solo aborda la cobertura de sentencias (Stmts) y ramas (Branches), sino que también incluye métricas adicionales de cobertura de condiciones y expresiones (FEC) en los archivos del diseño digital.

La cobertura de código es una métrica fundamental que evalúa la proporción de sentencias, ramas, condiciones y expresiones ejecutadas durante las pruebas en relación con el total existente en el código fuente. Aunque el objetivo ideal es alcanzar un code coverage global del 100%, a veces esto implica realizar pruebas irrelevantes. Es crucial comprender que, especialmente en el análisis de expresiones, se requiere probar todas las condiciones posibles de un statement condicional, es decir, la completa tabla de verdad.

La cobertura de sentencias (% de Cobertura de Stmts) refleja el porcentaje de líneas de código que han sido ejecutadas al menos una vez durante las pruebas. Un 100% en este aspecto indica que cada línea de código ha sido atravesada, proporcionando una garantía adicional de que cada instrucción se ha ejecutado al menos una vez.

La cobertura de ramas (% de Cobertura de Branches) mide la proporción de bifurcaciones de control de flujo, como declaraciones if, que se han evaluado tanto en su resultado verdadero como falso. Un 100% en cobertura de ramas señala que todas las posibles rutas de control de flujo han sido probadas.

Además de la cobertura tradicional, se ha evaluado la cobertura de condiciones y expresiones (FEC), proporcionando una visión más profunda de la validez de las condiciones y expresiones lógicas en el código. La cobertura FEC Condiciones (%) representa el porcentaje de condiciones lógicas evaluadas durante las pruebas, mientras que la cobertura FEC Expresiones (%) indica la proporción de expresiones lógicas probadas.

La tabla IX muestra el reporte de cobertura de código logrado con el testbench tb\_RAM\_MU\_v4 que arroja un nivel de cobertura total del 81.3%.

Tabla IX: Reporte de cobertura de código del testbench tb RAM MU v4.

Instancia	Total	Statement	Branch	Expresión	Condition
RAM MU	95,8	100	91,7	52,2	21,4
Registrar_entradas	92,9	100	85,7	50	-
GEN	100	100	100	-	100
MAD	100	100	100	80	-
TAC	100	100	100	47,6	75
PARIDAD	100	100	100	-	-
GEN PAR BUS-PPAL	100	100	-	50	-
CHEQUEO_PAR_SRAM	100	100	-	79,2	-
PFF	100	100	100	-	100
Maneja SRAM	100	100	100	-	-
REGISTROS_IO_SRAM	100	100	100	-	-

Los niveles más bajos se encuentran en la columna *expression* y *condition*, aunque al inspeccionarlas de manera individual se observa que las condiciones que no fueron probadas no son particularmente relevantes. Es posible analizar cada una de las condiciones que no son testeadas gracias a una vista especial como se muestra en la figura 40. Por ejemplo, aquí se ve que no fue comprobada una condición en la línea 573 del archivo RAM\_MU en donde se genera la señal CE\_SRAM\_N. Aquí lo que se quiere lograr es que CE\_SRAM\_N sea asignada por la lógica del diseño y en caso de que la unidad maestra decida deshabilitar la memoria, CE\_SRAM\_N sea '1' lógico. Dado que se trata de una lógica sencilla, se puede ignorar sabiendo que el resultado va a ser el esperado.

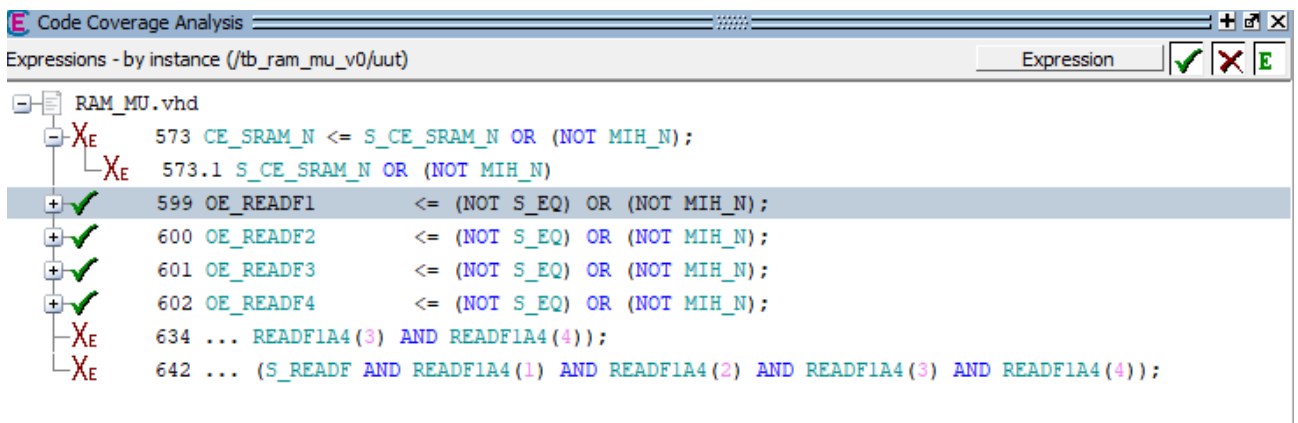


Figura 40 – Vista de Análisis de Cobertura de Código de Modelsim.

Teniendo en cuenta estos detalles, más la inspección visual de los ciclos de memoria, se puede concluir que el testbench es lo suficientemente completo como para servir de referencia y utilizarlo para comprobar el nuevo diseño digital.

## 6.2 Cambios para el diseño digital RAM-MU64

Teniendo en cuenta que el handshake entre la memoria y el procesador fue descrito de manera correcta y su funcionamiento está comprobado, en un principio sólo se tuvo que ahondar en el problema del direccionamiento. Es decir, lograr responder desde un lugar o slot de memoria a los requerimientos que están pensados para ser realizados en las otras, aunque físicamente la placa se encuentre en un solo lugar. Por este motivo, una de las principales modificaciones debió realizarse en el módulo MAD, ya que es el encargado de monitorizar cada petición de memoria en el BUS-PPAL y determinar en cuál módulo desea el procesador realizar la acción de lectura o escritura.

El bloque MAD de la versión 10 realiza el mismo funcionamiento que el de la memoria original, el cual genera una señal llamada EQ que se hace '1' lógico cada vez que coinciden los valores de la dirección del dispositivo (DA4\_N, DA3\_N, DA2\_N y DA1\_N) con las direcciones de memoria (GA14AN, GA13AN, GA16N y GA15N). Además, debe estar habilitada la memoria, es decir, MIH\_N debe ser '1'. La siguiente ecuación representa esta operación:

$$EQ = \text{not}(DA4N \text{ xor } GA14AN) \text{ and } \text{not}(DA3N \text{ xor } GA13AN) \text{ and } \text{not}(DA2N \text{ xor } GA16N) \\ \text{and } \text{not}(DA1N \text{ xor } GA15N) \text{ and } MIHN$$

Las señales GA16N y GA15N son las que se utilizan de manera activa para identificar entre los cuatro módulos de memoria, por lo tanto, son las que no se tendrán en cuenta en el nuevo diseño cuando este funcione en modo 64k28. Esto es posible, ya que el procesador accede a cada módulo de manera secuencial y no puede realizarlo en paralelo debido a la naturaleza del diseño. Por lo tanto, este módulo responderá a cada acción que se presente en el BUS-PPAL. La nueva fórmula que describe el comportamiento de la señal EQ en modo 64k se muestra a continuación:

$$EQ = \text{not}(DA4N \text{ xor } GA14AN) \text{ and } \text{not}(DA3N \text{ xor } GA13AN) \text{ and } MIHN$$

Una señal externa que indica el modo de trabajo de la memoria hace que el modelo del MAD opte por comportarse de acuerdo con la ecuación 1 o la 2, logrando así el objetivo de la flexibilidad del diseño.

Una vez solucionado el problema de la identificación de órdenes, se debe analizar cómo se estructurará la información en el chip de memoria SRAM. Cada dato proveniente del BUS-PPAL está compuesto por 24 bits y la memoria debe generar 4 bits de paridad a partir de estos. Por lo tanto, por cada dirección la memoria debe almacenar 28 bits de información. Debido a que el tamaño de cada palabra del chip de memoria RAM es de 8 bits, cada dato original (con la estructura del BUS-PPAL) se guarda fraccionado la información en 4 partes de 8 bits. Teniendo en cuenta esto, cada parte de 8 bits se la asocia a una dirección compuesta por los bits originales más 2 bits extra que identifican cada sub-palabra. La estructura se demuestra en la Tabla X.

*Tabla X: Comparación de la distribución de los datos en la memoria original y la disposición correspondiente en la memoria SRAM.*

Bits	31 - 24	23 - 16	15 - 8	7 - 0
Datos en la memoria original	Paridad (4)	GD_N(24)		
Datos en SRAM	Paridad (4)	GD_N(8)	GD_N(8)	GD_N(8)

El diseño anterior fue previsto para un chip de memoria SRAM de 1Mbit, es decir 17 bits de direcciones, y el actual para poder abarcar más cantidad de palabras es de 4Mbit o 19 bits de direcciones. Es por esto que se agregaron los bits GA15\_N y GA16\_N a la dirección de cada sub-palabra para almacenar de manera ordenada los datos como se muestra en la Tabla XI y XII.

*Tabla XI: Composición de las direcciones de memoria en la SRAM de 64k28 palabras.*

B18	B17	B16	B15-2	B1	B0
X	GA16	GA15	GA14_01	A2	A1

Tabla XII: Distribución de los datos de los cuatro bloques de 16k28 palabras en la memoria de 64k28 palabras.

	Cuarto bloque (Slot 4)	Tercer bloque (Slot 3)	Segundo bloque (Slot 2)	Primer bloque (Slot 1)
Direcciones de memoria	0x30000 – 0x3FFFF	0x20000 – 0x2FFFF	0x10000 – 0x1FFFF	0x00000 – 0x0FFFF

Cabe recordar que los bits A1 y A2 son direcciones auxiliares que se utilizan para guardar cada dato de 28 bits en bloques de 32 bits, separados en 4 subpartes de 8 bits ya que la memoria RAM sólo posee 8 bits de datos por dirección.

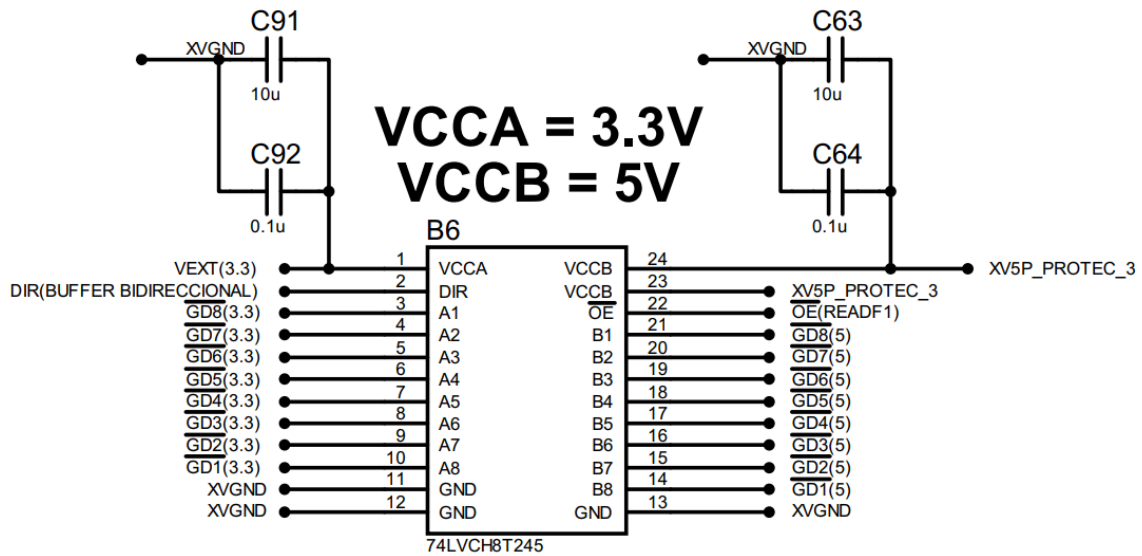


Figura 41 – Conexiones de los buffers bidireccionales del diseño anterior.

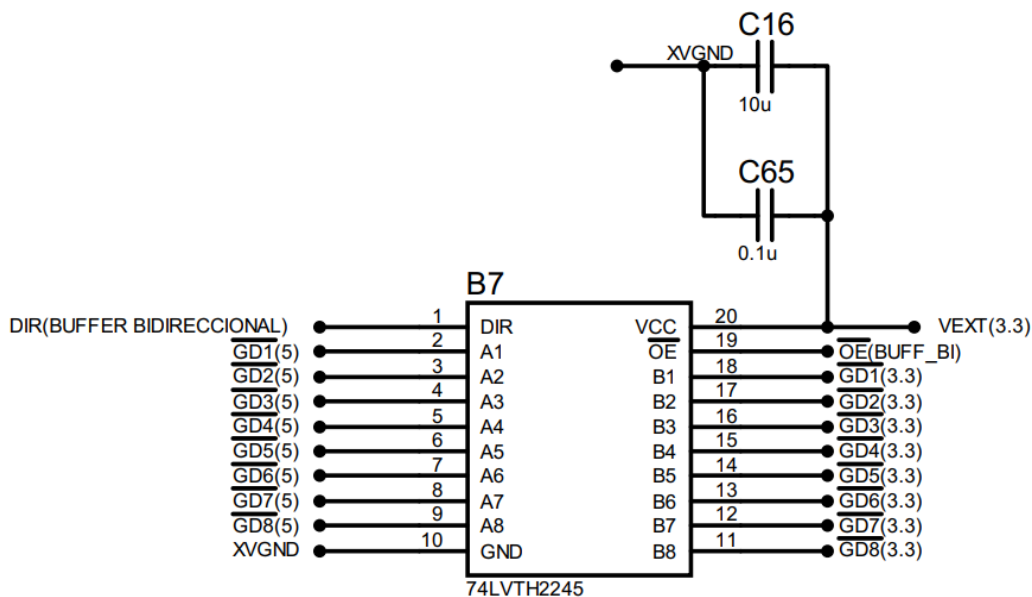


Figura 42 – Conexiones de los buffers bidireccionales del nuevo diseño.

Otra modificación necesaria respecto al nuevo diseño de hardware fue modificar el comportamiento de la habilitación de los buffers bidireccionales. La figura 41 muestra la disposición de las entradas y salidas de los buffers bidireccionales del diseño anterior y la 42 del nuevo diseño. La diferencia entre estas conexiones es que las tensiones de 5v ahora se encuentran en los puertos B de los buffers y las de 3,3v en los puertos A.

Teniendo en cuenta la tabla de verdad que describe el funcionamiento del buffer, que se muestra en la tabla XIII, se puede concluir que el comportamiento del pin que gobierna la dirección de traslación (DIR) debe ser el opuesto al del diseño anterior. Es decir, basta con negar la función lógica que se utilizó anteriormente.

Tabla XIII: Tabla de verdad del buffer bidireccional 74LVCH8T245.

Entradas		Salidas
OE	DIR	
L	L	Datos del bus B hacia el bus A
L	H	Datos del bus A hacia el bus B
H	X	Alta impedancia (Z)

Además de las modificaciones introducidas producto del cambio de hardware, se realizaron modificaciones en el diseño teniendo en cuenta algunas observaciones en la verificación y análisis de la versión anterior. Uno de estos puntos es la manera de acondicionar las señales que llegan hasta los pines de la FPGA. En el diseño anterior se utilizó en el módulo registrar entradas filtros de glitches asimétricos, como se muestra en la figura 43.

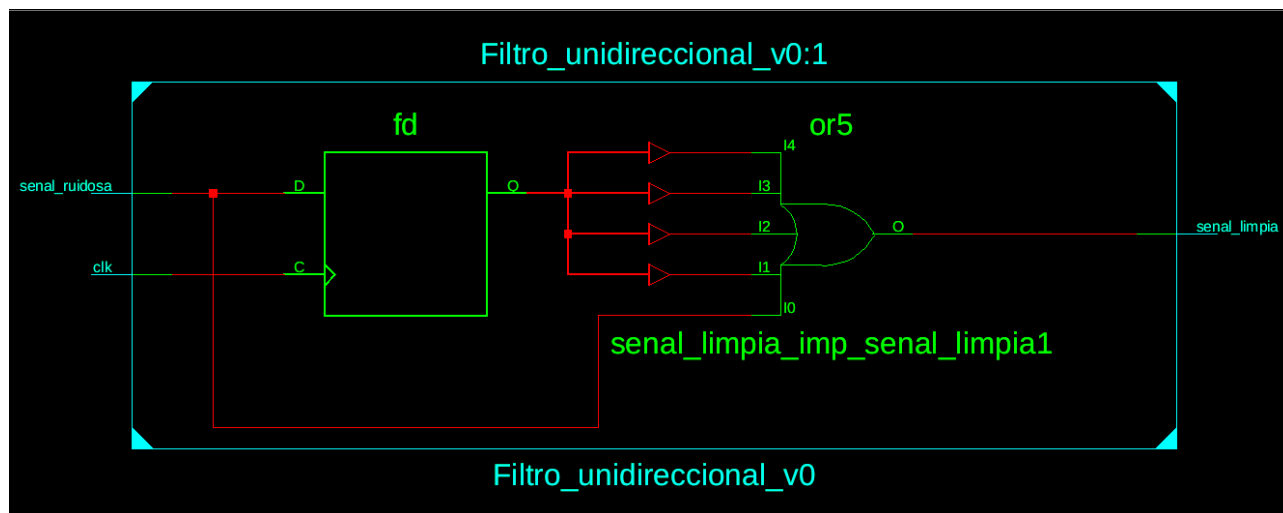


Figura 43 – Estructura de los filtros de entrada asimétricos.

Este, está compuesto por un registro de las  $N$  entradas anteriores que luego se comparan con una compuerta OR debido al tipo de lógica negada que se utiliza en el BUS-PPAL. La salida de este tipo de filtros es '0' solo cuando se mantiene durante  $N$  ciclos ese valor a la entrada, pero el valor '1' se coloca con solo un ciclo. Por este motivo se lo llama filtro asimétrico, porque se filtran sólo los '0' o '1'. Esto último depende si la compuerta lógica colocada a la salida de los registros es AND o OR. En la Figura 44 se muestra una simulación con un filtro asimétrico de '0' de cuatro (4) ciclos donde se puede

ver el comportamiento descrito anteriormente.

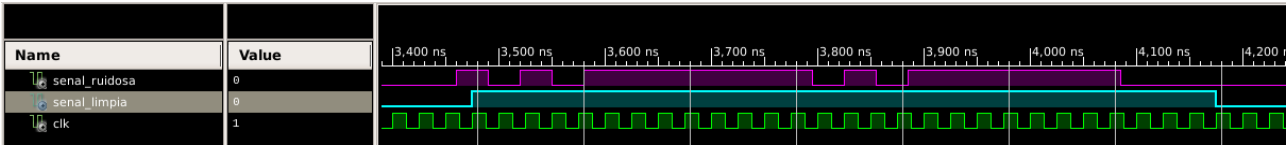


Figura 44 – Funcionamiento de un filtro asimétrico de 4 ciclos.

Este filtro es aceptable para ciertas señales de entrada, como puede ser el RESET general del sistema. Uno se quiere asegurar que no haya reinicios aleatorios por ruido electromagnético y la vuelta al estado normal del pin debe ser lo más rápida posible. En cambio, para casi todo el resto de las señales, es más conveniente utilizar un filtro simétrico que cambie su salida sólo si mantiene un '1' o un '0' durante N ciclos de reloj y si la salida actual es diferente de la entrada. En la figura 45 se muestra gráficamente el comportamiento. En este ejemplo el filtro está configurado para cambiar la salida luego de que un valor se sostenga por tres (3) flancos ascendentes, es por eso que en el primer

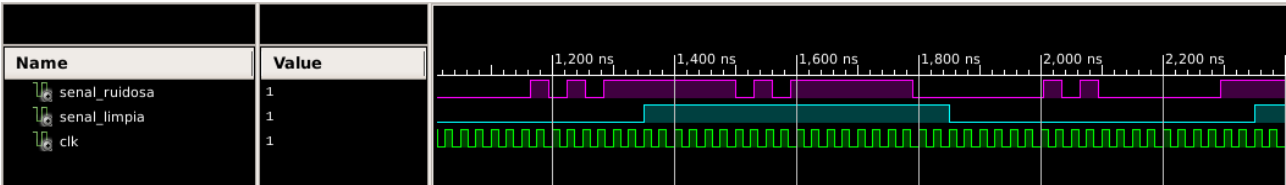


Figura 45 – Ciclo de funcionamiento de un filtro simétrico de 3 ciclos.

pulso de dos ciclos la salida no cambia.

La desventaja que tiene este tipo de filtros es la lógica adicional que se necesita para guardar el estado de la salida actual y las comparaciones, lo que se traduce en la utilización de más LUTs. En la figura 46 se muestra la lógica de un filtro simétrico de 3 ciclos.

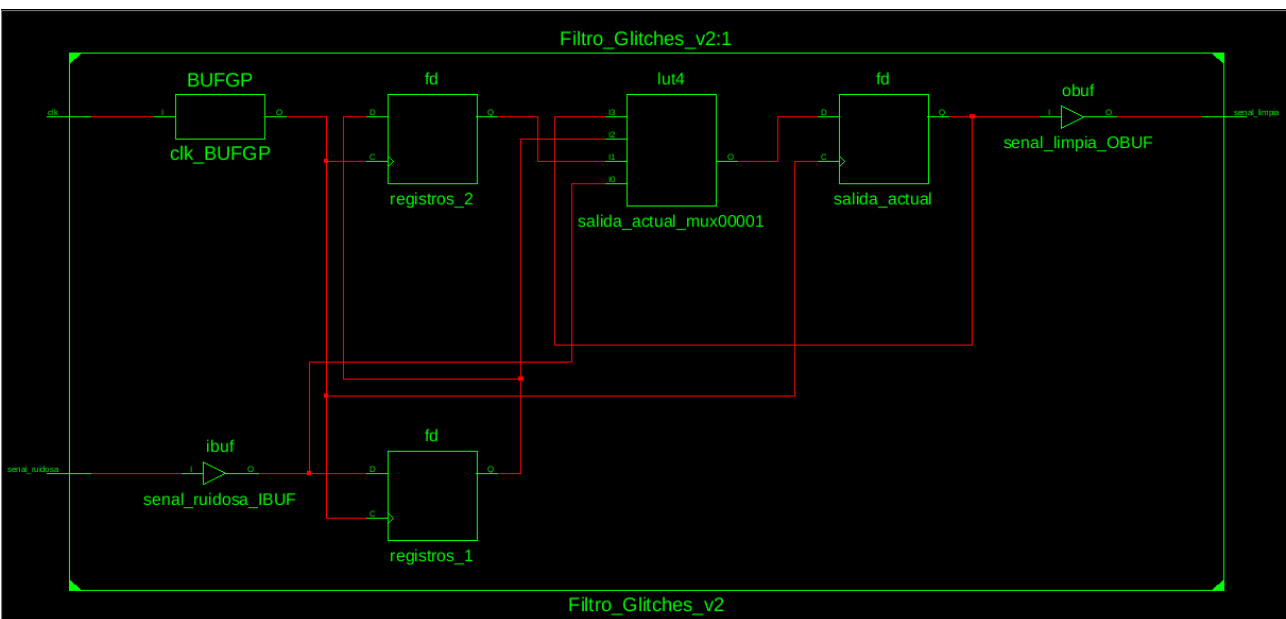
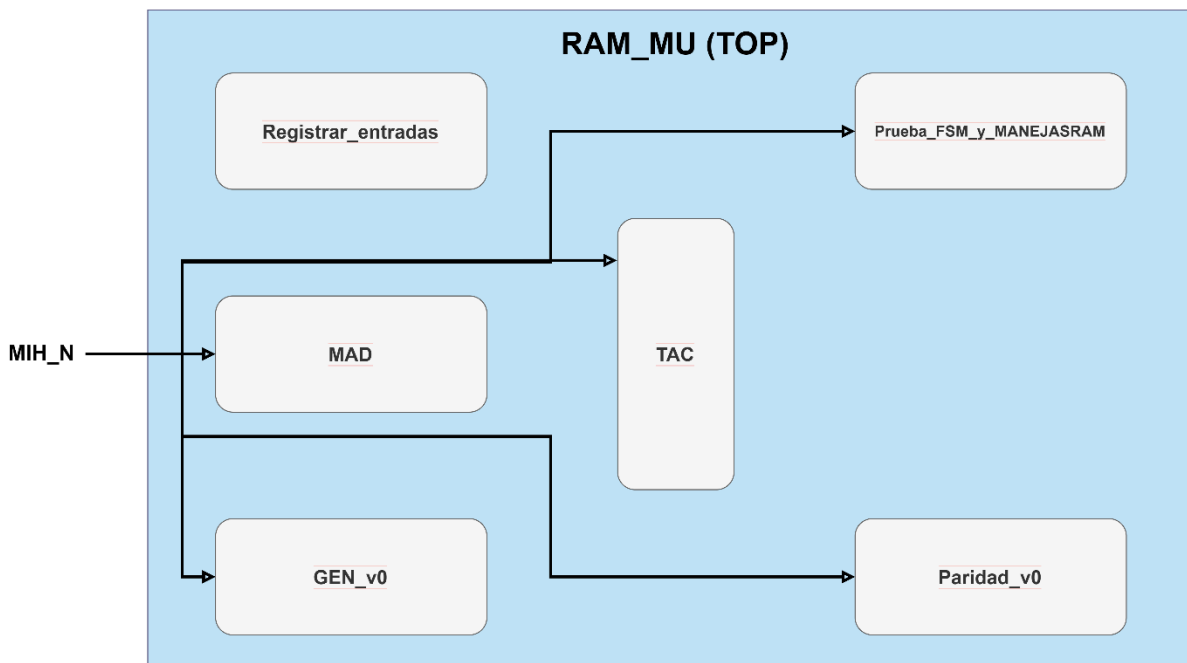


Figura 46 – Lógica de un filtro simétrico de 3 ciclos.

El filtro simétrico fue utilizado en primer lugar para la entrada MODO64\_K y MIH\_N. En el caso de MODO64\_K es vital que su valor no cambie durante el funcionamiento de la memoria, por lo que se le agregó un retraso de 10 ciclos. De esta manera se asegura que el valor colocado en el hardware sea el deseado por el operador durante todo un ciclo de encendido.

En cuanto a MIH\_N, se agregó un filtro simétrico debido a que se utiliza como RESET para varios módulos dentro del módulo TOP. En las versiones anteriores, se filtraban los ruidos con un filtro asimétrico sólo dentro del módulo MAD, como se observa en la figura 47.

Teniendo en cuenta que ante una inhibición de los módulos de memoria por parte de la CPU se deben reiniciar todos los procesos internos y que es importante que cuando ocurra no sea aleatorio, se removió el filtro del bloque MAD y se lo colocó dentro de Registrar\_Entradas. De aquí sale la señal MIH\_N\_REGISTRADA la cual luego es utilizada como reset interno como se muestra en la figura 48.



*Figura 47 – Utilización de la señal MIH\_N en versiones anteriores.*

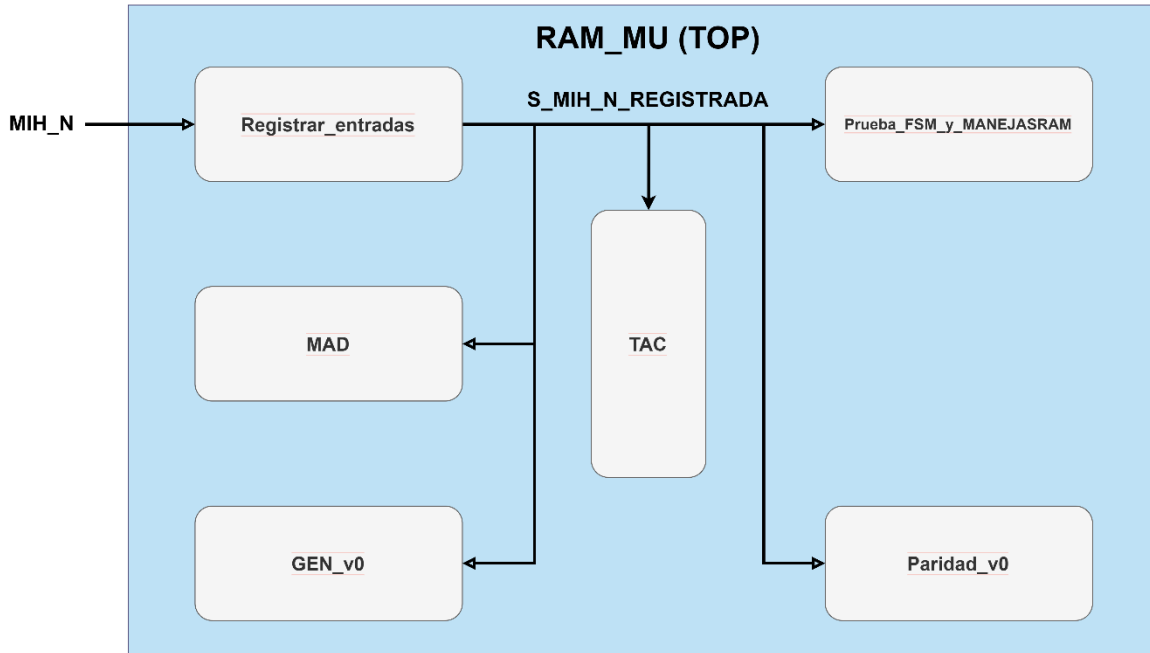


Figura 48 – Utilización de la señal MIH\_N en el nuevo diseño.

### 6.3 Componentes del diseño digital

Debido a que la descripción de cada componente perteneciente al diseño digital se realizó en un formato especial adjuntado en el Anexo B, en este apartado sólo se mencionan los distintos componentes que conforman el top design. En la figura 49 se muestra el diagrama RTL del diseño.

#### a) Registrar\_entradas\_v1.vhd

Este bloque permite eliminar problemas de meta-estabilidad que se dan cuando se manejan dos señales de reloj con flancos distintos y variables, favoreciendo a conseguir un mayor grado de confiabilidad sobre el funcionamiento de la memoria. Además, sirve para filtrar el ruido eléctrico presente en las señales que llegan hasta la placa de circuito impreso.

La figura 50 muestra el diagrama de entidad. A este componente se conectan las entradas crudas provenientes desde los pines de la memoria y luego se distribuyen al resto del diseño las señales internas libres de glitches. Las entradas MODO\_64K y MIH\_N se filtran a través del componente Filtro\_Glitches\_v2. En el resto de las señales se utilizan soluciones a medida, ya que se deben agregar otras condiciones como la retención de datos ante un ciclo de memoria.

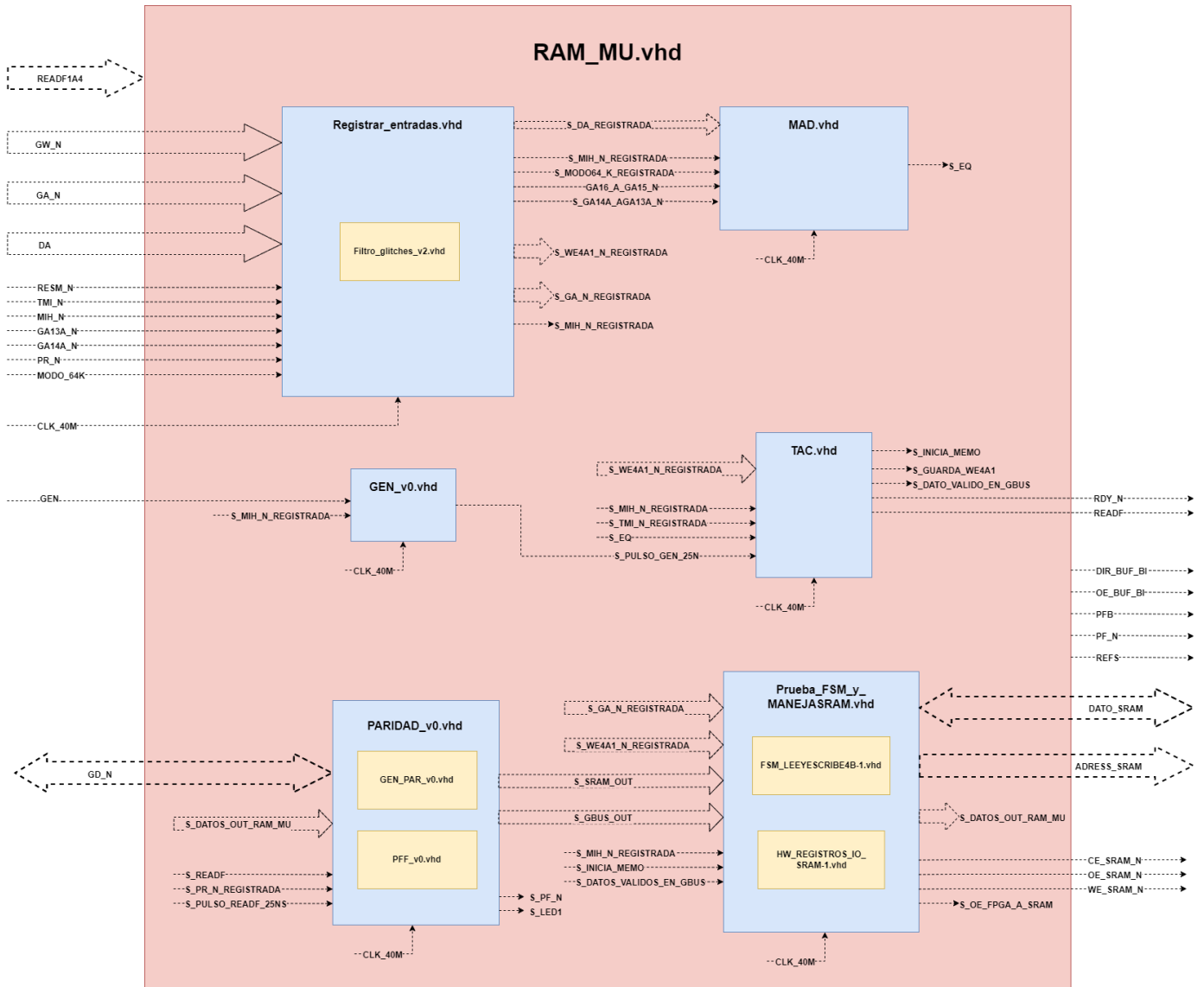


Figura 49 – Diseño digital del proyecto RAM-MU64 entidad principal (top).

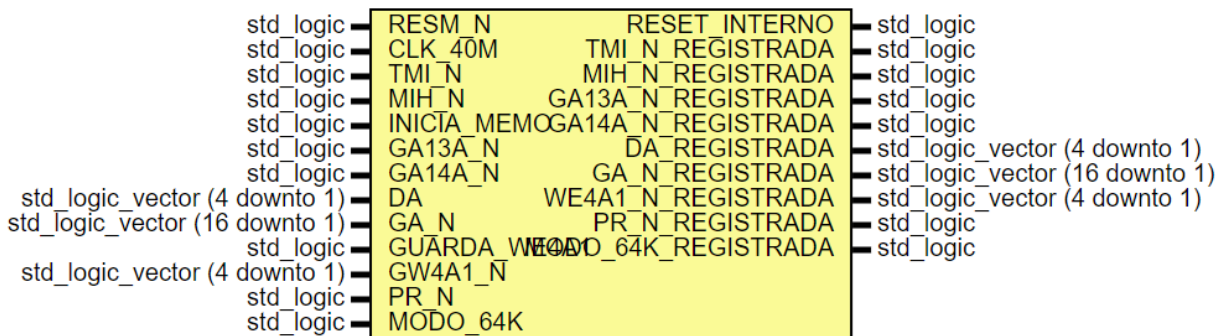


Figura 50 – Diagrama de la entidad registrar\_entradas\_v1.

### b) Filtro\_glitches\_v2.vhd

Filtro de glitches, basado en la implementación de Cypress (*Glitch Filter 2.0*, n.d.). La señal limpia se crea de tal manera que un cambio es hecho sólo si se mantuvo el valor de entrada durante  $n$  ciclos. Por ejemplo, si  $n=3$  la señal ruidosa de entrada tiene que mantener ese valor durante 3 ciclos para que la señal filtrada tome el mismo valor. La figura 51 muestra el diagrama de entidad. En esta segunda versión se realiza un cambio en la salida cuando se detectan  $n$  '0' o '1' consecutivos, de lo contrario se mantiene el

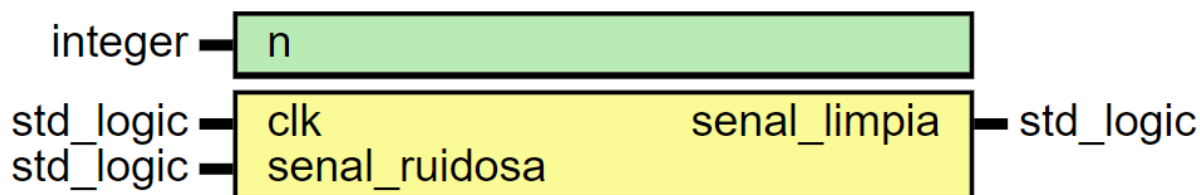


Figura 51 – Diagrama de la entidad Filtro\_glitches\_v2 .

valor de salida.

### c) gen\_v0.vhd

Este módulo se encarga de sincronizar el reloj de 8MHz del BUS-PPAL con el reloj interno de la FPGA de 50MHz. Para ello se utilizan dos registros en cascada a la entrada de reloj asincrónica y se toman las salidas de estos registros para detectar un flanco ascendente. La salida del reloj sincronizado toma el valor '0' por defecto y '1' ante cada flanco ascendente. Se utilizan las salidas pertenecientes a los registros para evitar metaestabilidad. La figura 52 muestra el diagrama de entidad.

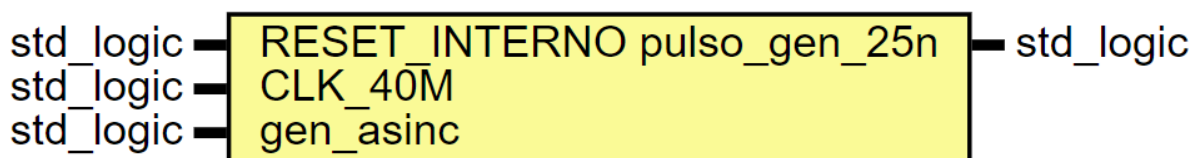


Figura 52 – Diagrama de la entidad gen\_v0 .

### d) MAD\_v0.vhd

El bloque MAD es el encargado, al igual que en el diseño original, de identificar a qué módulo en particular son dirigidas las acciones ejecutadas por el CPU. Para ello, se utilizan distintas entradas del BUS-PPAL: GA13A, GA14A, GA15\_N, GA16\_N y DA. La señal EQ se utiliza en el resto del diseño para activar o no los procedimientos de lectura o escritura. La figura 53 muestra el diagrama de entidad.

La característica nueva que agrega esta versión es la posibilidad de responder ante todas las peticiones de un conjunto de cuatro módulos, habilitando así que se pueda elevar la capacidad de un sólo módulo de 16k a 64k. Dado que este diseño debe utilizarse en ciertas ocasiones en 16k de manera obligatoria, se agrega una entrada extra llamada MODO\_64K la cual indica cómo debe comportarse el MAD.

El conjunto de cuatro módulos de memoria se identifica comparando DA(3) con GA13A y DA(4) con GA14A. Si estos valores son idénticos, las órdenes sobre el BUS-PPAL son para ese conjunto en particular. Para identificar cada uno de los cuatro módulos de manera individual se utilizan las señales restantes: DA(1) se compara con GA15\_N y DA(2) con GA16\_N. Con esto, es posible determinar si el módulo debe responder o no ante cada acción de escritura o lectura.

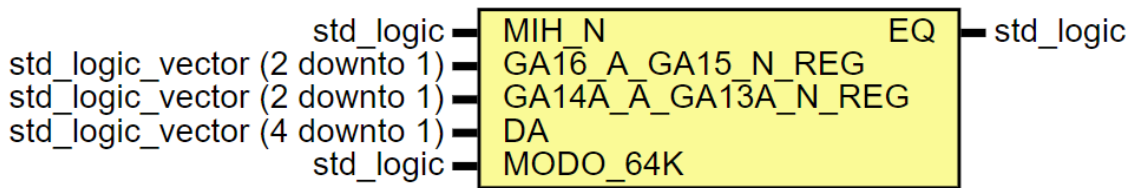


Figura 53 – Diagrama de la entidad MAD\_v0 .

#### e) TAC\_v1.vhd

El circuito de sincronización y control (TAC) procesa las señales necesarias para secuenciar los distintos ciclos de la memoria que se envían como señal de handshake a la unidad que está utilizando la memoria. La figura 54 muestra el diagrama de entidad.

En este bloque se detecta cuando se inicia un ciclo de memoria (INICIA\_MEMO = '1'). Además, se implementa una lógica que permite inferir el momento que está el valor correcto de GW y, así, indicar la acción a realizar con la memoria. También, se divide, para un ciclo de escritura, el momento donde los datos están establecidos en el bus (DATO\_VALIDO\_EN\_GBUS), para poder enviar al bloque correspondiente la señal de habilitación de los flip-flops que toman y retienen dichos datos. Por último, se implementa una lógica para evitar que DATO\_VALIDO\_EN\_GBUS = '1' si nos encontramos en un ciclo de lectura. Finalmente, se generan las señales de aviso RDY y READF con el temporizado específico del handshake original, según corresponda.

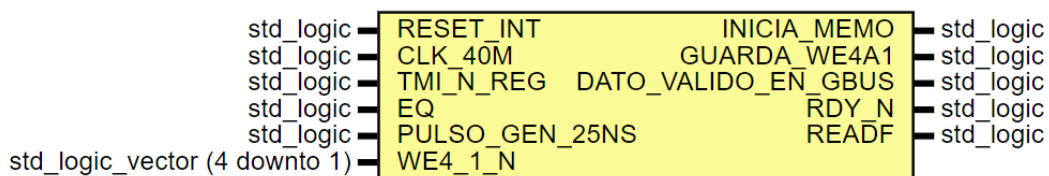


Figura 54 – Diagrama de la entidad TAC\_v1.

### f) Paridad\_v0.vhd

El componente Paridad\_v0 se encarga de generar los bits de paridad correspondientes para la escritura de datos, de comprobar si existen fallos de paridad en la lectura y de generar la señal PF (Parity Fault) del BUS-PPAL, al igual que controlar el LED indicador de fallo de paridad PF\_LED. Para esto hace uso de dos bloques más PFF\_v0 y GEN\_PAR\_v0. La figura 55 muestra el diagrama de entidad.

Cuando un dato proviene del BUS-PPAL (ciclo de escritura), se generan los 4 bits de paridad a partir de los 24 bits de datos (GBUS\_IN). En la operación inversa (ciclo de lectura), se comprueba que los bits de paridad de los datos almacenados (SRAM\_IN) sean iguales a los que fueron generados al momento de la escritura. Para determinar si se encuentra en un ciclo de lectura se utiliza la entrada READF. READF toma el valor '1' lógico en un ciclo de lectura y a partir de esto, se habilita la comprobación de paridad. La entrada PULSO\_READF\_25NS es un registro de READF y se utiliza para detectar un flanco descendente en READF.

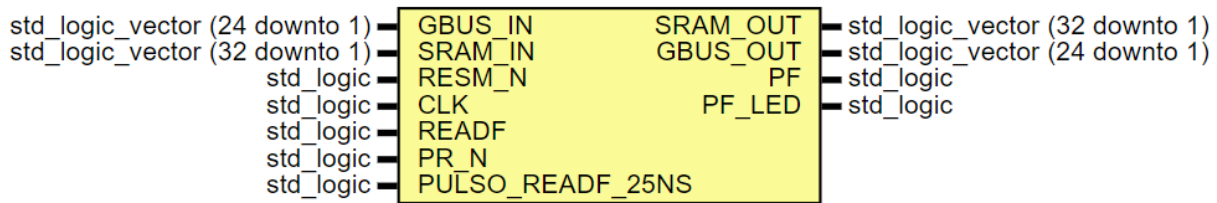


Figura 55 – Diagrama de la entidad paridad\_v0.

### g) GEN\_PAR\_V0

Este módulo se encarga de generar los 4 bits de paridad a partir de una palabra de 24 bits, de acuerdo con el funcionamiento de la memoria original. La figura 56 muestra el diagrama de entidad.

La entrada PR\_N indica si se debe generar paridad par o impar a partir de los 24 bits de DATOS\_IN. La salida DATOS\_OUT posee los valores de DATOS\_IN más los cuatro bits de paridad generados. Como la memoria SRAM guarda datos de a 8 bits, se debe generar una señal de 32 bits los cuales los últimos 4 no se usan y se completan con '0'.

La paridad se genera de la siguiente manera, los primeros ocho bits (1 a 8 de DATOS\_IN) se asignan al primer bit de paridad. Los siguiente cuatro (9 a 12 de DATOS\_IN) se utilizan para el segundo bit y el tercero se realiza a partir de próximos cuatro (13 a 16 de DATOS\_IN). El último bit de paridad se realiza con los últimos ocho bits de entrada (17 a 24 de DATOS\_IN).

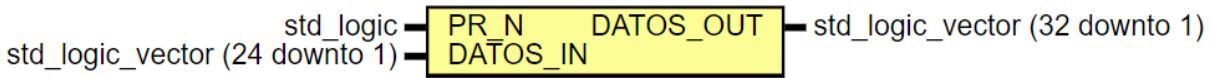


Figura 56 – Diagrama de la entidad GEN\_PAR\_v0.

#### h) PFF\_V0.vhd

Esté módulo se encarga de generar las señales de Parity Fault (PF) y del led indicador de Parity Fault (PF\_LED). La figura 57 muestra el diagrama de entidad.

La entrada PFA indica si existe un fallo en la comprobación de paridad ('1' falla de paridad y '0' sin falla) y las señales READF y PULSO\_READF\_25NS se utilizan para sincronizar la salida PF con el handshake del BUS-PPAL.

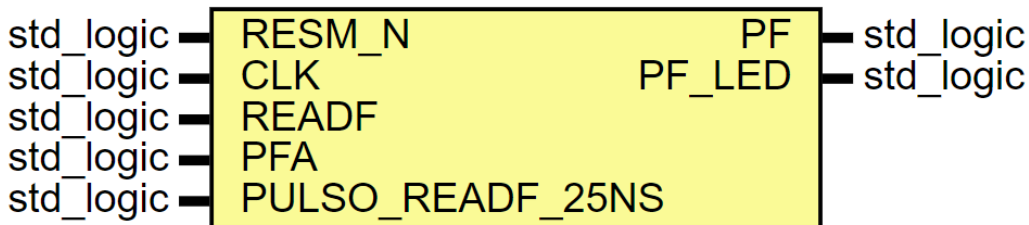


Figura 57 – Diagrama de la entidad PFF\_v0.

#### i) Prueba\_y\_ManejaSRAM.vhd

Este módulo se encarga de administrar los datos que se almacenan en la memoria y, por ende, del manejo del integrado de memoria SRAM. Está compuesto por dos componentes: una máquina de 18 estados y una implementación de registros de datos que contiene toda la lógica necesaria para realizar la descomposición de la palabra de 32 bits (24 bits de dato + 4 bits paridad + 4 bits en '0'). Este diseño no agrega lógica adicional, sólo interconecta las entradas y salidas pertinentes a los componentes internos y externos. La figura 58 muestra el diagrama de entidad.

La forma de escribir o leer se realiza de a 8 bits, ya que de esta manera funciona el integrado de memoria SRAM (el manejo de datos lo hace mediante 8 pines de entrada-salida). El bloque recibe los 4 bits (WE1 a WE4) que le indican el formato de la palabra que se irán a escribir o leer de la memoria. Del bloque TAC recibe la señal INICIA\_MEMO que le indica cuándo iniciar un ciclo de memoria, junto con la señal DATO\_VALIDO\_EN\_GBUS. Del bloque PARIDAD obtiene los datos provenientes del BUS-PPAL junto con la paridad generada sobre estos. Además, si el BUS-PPAL quiere leer datos desde la memoria, este bloque interactúa con otra instancia del bloque GEN\_PAR\_v0 para obtener los bits de paridad a partir del dato almacenado en la SRAM. Estos bits se compararán con los previamente guardados en esa dirección de memoria. Finalmente, copiarán esos datos en el BUS-PPAL para que puedan ser leídos.

Por otro lado, se encargará de escribir el dato como se lo indiquen las señales WE (ya sea, escribir la palabra completa, la mitad superior de la palabra, el carácter inferior, entre

otros casos)

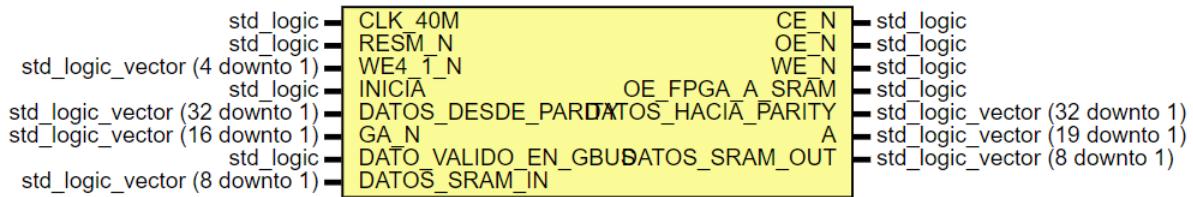


Figura 58 – Diagrama de la entidad prueba\_fsm\_y maneja\_SRAM.

### j) FSM\_LEEYESCRIBE4B.vhd

Este bloque describe la máquina de estados (FSM) que se complementa con HW\_REGISTROS para implementar la lógica de los ciclos de memoria y el control del hardware. Cada vez que se inicia un ciclo de memoria por medio de la entrada INICIA\_MEMO, se extraen los datos de memoria modificando los pines de control de la SRAM y la dirección A1y2 que compone cada dato. Si se trata de un ciclo de lectura, aquí terminaría la secuencia de estados. En caso de que se trate de un ciclo de escritura o escritura parcial, la FSM continúa con el proceso de colocar los nuevos datos en la SRAM. La figura 59 muestra el diagrama de entidad.

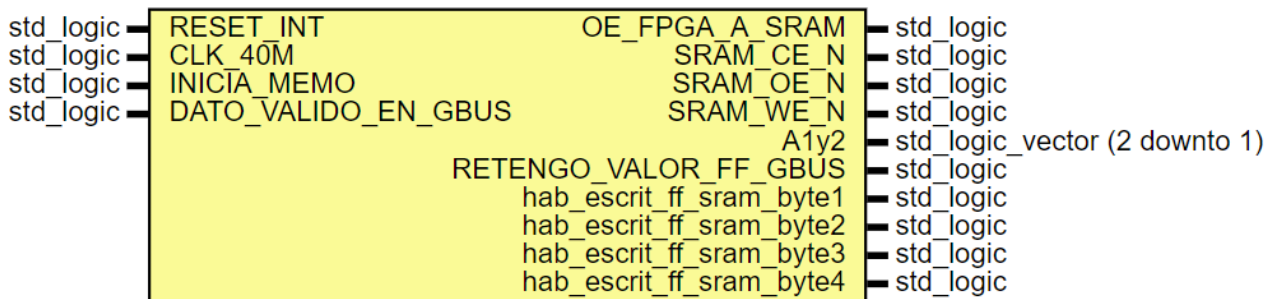


Figura 59 – Diagrama de la entidad FSM\_LEEYESCRIBE4B.vhd.

### k) HW\_REGISTROS\_IO\_SRAM

Este módulo es el encargado de interactuar a bajo nivel con el chip de SRAM. Coloca y extrae valores, tanto en el módulo de paridad como en la memoria SRAM, en función de las entradas que recibe del bloque FSM\_LEEYESCRIBE4B. Teniendo en cuenta que el chip de memoria SRAM puede ser accedido de 1 byte a la vez, se deben escribir los 28 bits del dato en bloques de 32 bits. De esta manera un dato queda dividido en SRAM en 4 partes iguales de 8 bits. Para poder direccionar estos, se deben crear una nueva dirección que traduzca la dirección original de 16 bits a una de 18. Los bytes extraídos de cada dirección se envían al bloque encargado de comprobar la paridad. La figura 60 muestra el diagrama de entidad.

En primer lugar, se define para cada grupo de 8 bits que componen el dato de 32 bits

a almacenar en SRAM, máscaras de bits que serán utilizadas para discriminar los bits utilizados en ese ciclo de memoria. Esta operación es particularmente importante para el caso de las escrituras parciales, ya que se desea mantener los datos previamente escritos. Luego, se realiza la operación AND entre los datos que ingresan desde el bloque de paridad y las máscaras. El resultado de estas operaciones es registrado mediante flip-flops, los cuales poseen una entrada *enable* gobernada por la máquina de estados. De esta manera se retiene el dato proveniente del bloque de paridad cuando es necesario.

Para el caso de los datos provenientes de la SRAM se realiza el mismo procedimiento que se describió anteriormente. En este caso, las máscaras se deben conformar con el valor correspondiente para poner en '0' la parte de la palabra que debe ser sobrescrita por los datos que vienen del BUS-PPAL, es decir desde el módulo de paridad.

A la hora de colocar los datos finales al bloque de paridad se copian directamente desde los flip-flops que almacenan los valores leídos desde la SRAM. Para la copia de valores desde la entrada del parity hacia la SRAM se realiza una operación XOR a partir de los datos almacenados en los registros del BUS-PPAL y la memoria SRAM. De esta manera, se asegura que sólo sean modificados los bits necesarios en cada dirección de la memoria.

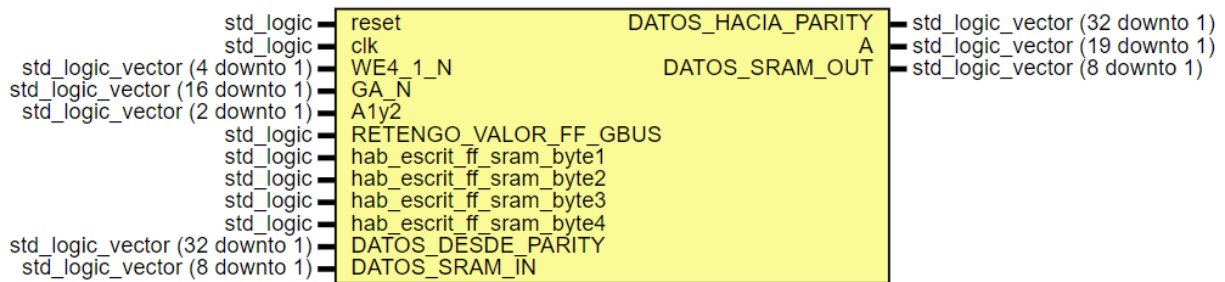


Figura 60 – Diagrama de la entidad HW\_REGISTROS\_IO\_SRAM.vhd.

## 6.4 Etapa de pruebas y corrección de errores del diseño digital en prototipo anterior

El hardware utilizado para efectuar las pruebas del nuevo diseño digital fue la versión 4 de la placa de circuito impreso, lo que propició una optimización del tiempo de ejecución de este proyecto. Se debe considerar que la versión 5 del hardware, a pesar de reemplazar algunos componentes respecto a la versión anterior, no implica alteraciones en la lógica del sistema.

En la fase inicial, se comprobó la funcionalidad en el modo de 16k; operando la última versión del diseño digital en la versión 4 del prototipo, aunque sin la capacidad de modificar el modo de operación a través del jumper designado. Seguidamente, se efectuaron pruebas en el modo de 64k mediante la misma configuración estática. Al concluir, se sometió a prueba la versión final del diseño digital, que sí facilita el cambio de modos mediante el uso del jumper.

### 6.4.1 Configuración inicial del prototipo

Inicialmente, se procedió a adecuar el pinout del proyecto Xilinx para que se

alineara con la arquitectura de la versión 4 del hardware. Esto implicó un mapeo detallado de los pines físicos de la FPGA para su correspondencia con los buffers bidireccionales, la memoria SRAM, el oscilador, entre otros componentes. La configuración se efectuó utilizando un archivo .CSV en concordancia con el esquemático de la versión 4.

Con el objetivo de validar la precisión de estas configuraciones y el ensamble correcto del hardware, se ejecutaron múltiples pruebas utilizando diseños digitales específicos. Las pruebas realizadas incluyeron:

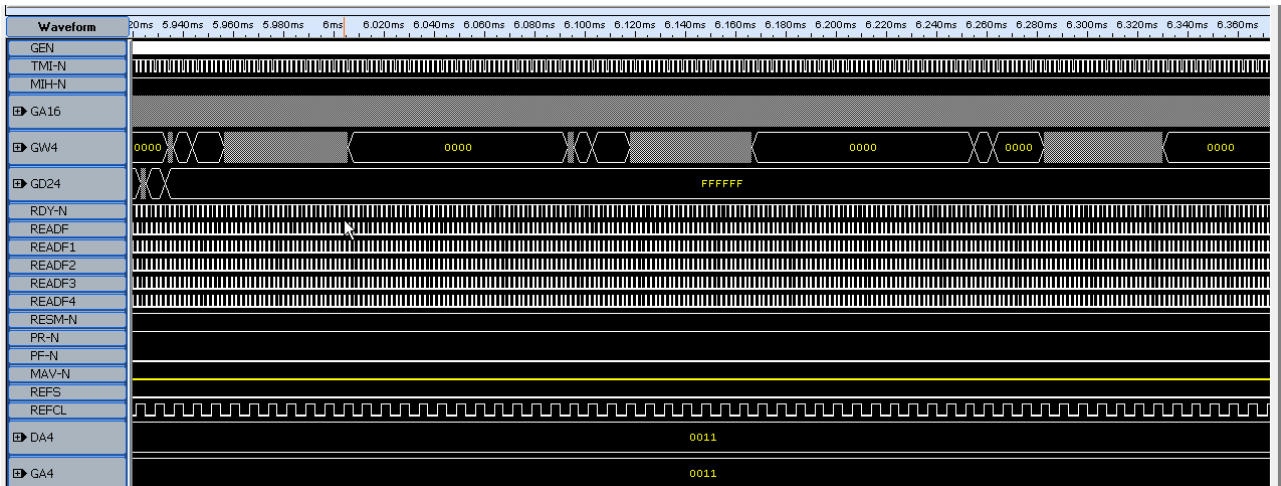
- Monitorización de la transmisión de señales entre el BUS-PPAL y la FPGA.
- Envío de señales desde la FPGA hacia el BUS-PPAL.
- Identificación del estado del jumper, el cual determina el modo operativo entre 16k y 64k.
- Verificación de la conectividad entre los pines de la FPGA y la memoria SRAM.
- Ejecución de secuencias de escritura, lectura y reescritura en diversas direcciones de la SRAM para evaluar su correcto funcionamiento.

Tras confirmar el éxito en todas las pruebas, se pudo garantizar la integridad del sistema. Esto permitió la instalación del nuevo prototipo RAM-MU64 en los equipos de prueba para iniciar los ensayos reales de funcionamiento sobre una computadora naval.

#### 6.4.2 Funcionamiento del modo 16k

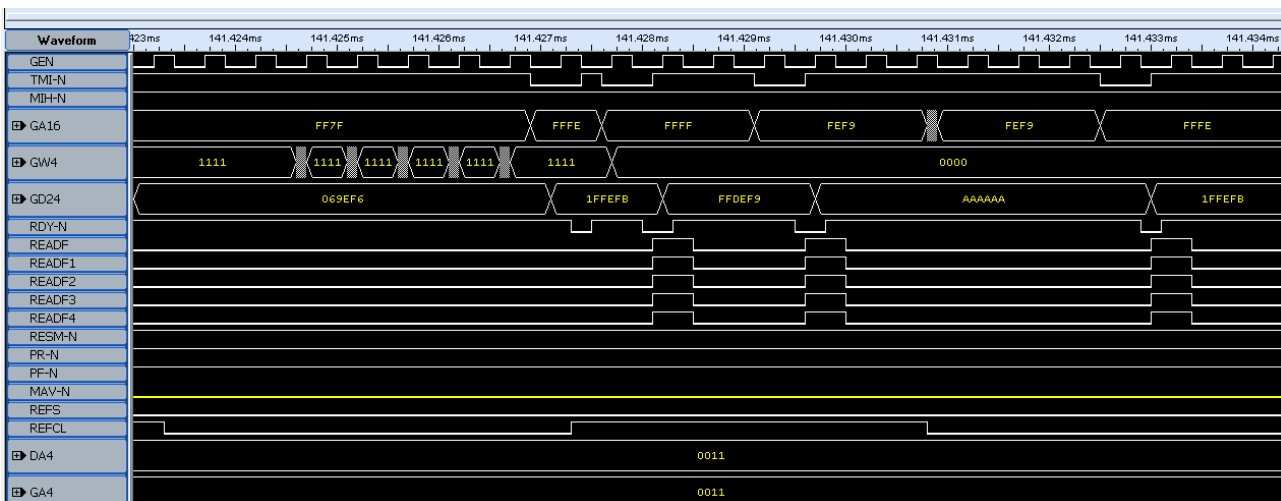
Para la primera prueba del diseño digital adaptado se modificó mínimamente el último código RTL estable de la versión 3 de hardware llamado *RAM\_MU\_VHDLv10*. Uno de los cambios obligados fue incrementar la longitud de la señal utilizada para conectar la entidad TOP del diseño con la memoria SRAM, ya que cuenta con dos bits más de direcciones; ahora son 19 y antes eran 17. Los bits extras no son utilizados internamente en el diseño y son dejados en cero lógico para la escritura. Se subió el código al dispositivo y se corrió el UTEST. A esta nueva versión se la llamó *RAM\_MU\_VHDLv10\_1*.

El resultado no fue satisfactorio y el equipo falló en cargar programa en la memoria. Este resultado fue sorprendente ya que los cambios en el diseño digital fueron mínimos y el hardware fue comprobado exhaustivamente. En la Figura 61 se puede ver una captura de la memoria funcionando de manera errónea. Lo que se aprecia es que existe un handshake entre la computadora y la memoria, pero los bits de datos siempre son '1' lógico (GD = 0xFFFFF). Estas capturas fueron tomadas en varias oportunidades con el analizador lógico con el prototipo colocado en el probador de memorias para analizar los datos.



*Figura 61 – Captura de la memoria en modo 16k sin funcionar.*

Ante este panorama se volvió a comprobar todas las conexiones en la placa y se corrieron nuevamente los firmwares especiales de prueba. Dado que los resultados fueron otra vez insatisfactorios se comenzó a comparar los esquemáticos de la versión 3 con la del cuatro para encontrar diferencias. Aquí es donde se encontró el potencial problema, las conexiones de los buffers bidireccionales estaban intercambiados. Este tema ya se mencionó en la sección de hardware, pero vale la pena destacar que hasta este momento esto no fue percatado. La siguiente pregunta que se formuló fue, porque las pruebas que se hicieron para comprobar la transmisión de datos entre el BUS-PPAL y la FPGA no se tuvo este problema. La respuesta es que esas pruebas se realizaron mirando el esquemático de la versión 4 solamente y el código no fue copiado de la versión anterior, sino que se hizo desde cero. Todo este análisis y revisión consumió aproximadamente



*Figura 62 – Captura de la memoria en modo 16k funcionando.*

tres semanas y varias versiones del proyecto VHDL.

Una vez aclaradas estas dudas, se negó la señal lógica pertinente y se realizó una nueva prueba cuya captura se muestra en la Figura 62.

Esta prueba si fue exitosa y por lo tanto ya se obtuvo una versión del diseño digital RAM-MU llamada *RAM\_MU\_VHDLv10\_5*, adaptado a la versión 4 del prototipo de hardware.

Luego de hacer una prueba funcionando a la memoria de manera independiente, se la colocó en conjunto con otras memorias originales y se tomaron varias capturas. Dado que el funcionamiento es el esperado y es al menos igual de bueno que el prototipo anterior, se comenzó a trabajar en las pruebas del modo 64k.

### 6.4.3 Funcionamiento del modo 64k

Para comprobar el funcionamiento del modo 64k24 palabras, se creó una nueva versión del diseño digital llamado *RAM\_MU\_VHDLv10\_6*. Luego de hacer pruebas en un testbench y confirmar que en teoría el módulo respondía correctamente ante peticiones de escritura en diferentes slots de memoria de manera secuencial, se realizó la prueba UTEST. La primera prueba fue un éxito, se pudo comprobar cómo la memoria respondía de la manera esperada como se muestra en la figura 63.

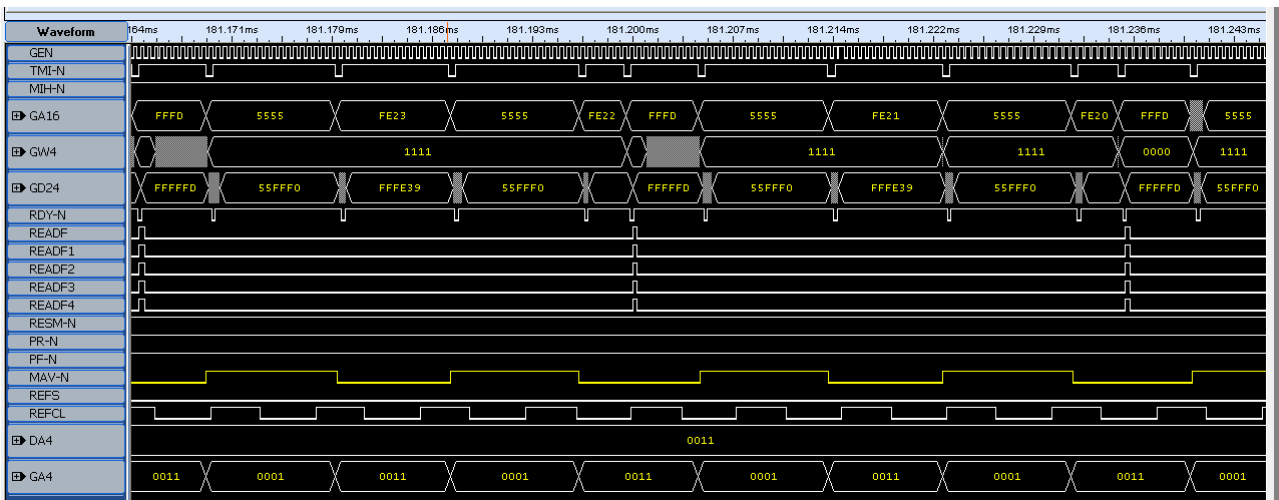


Figura 63 – Captura de la memoria en modo 64k reaccionando ante peticiones de distintos slots.

Luego de este ensayo, se realizó una nueva versión del diseño digital llamada *RAM\_MU\_VHDLv10\_6*; en la cual se incluyó el soporte para el jumper de cambio de modo entre 16k24 palabras y 64k24 palabras. En ambos modos los resultados fueron el mismo que en las descripciones con modo “forzado”, por lo que se obtuvo una primera versión del nuevo diseño digital para el proyecto RAM-MU64.

A pesar de pasar satisfactoriamente el UTEST, otras pruebas como el parity test y el funcionamiento en algunas posiciones en particular de memoria el diseño digital falló. En base a esto se decidió tomar esta versión como referencia, ya que la lógica principal funcionaba correctamente, y atacar los problemas que surgieron uno a uno hasta eliminarlos completamente. En conjunto con la solución de estos errores, se agregaron las mejoras propuestas de manera teórica.

### 6.4.4 Modificación y depuración del diseño digital

La primera modificación que se introdujo fue el filtrado de ruidos simétrico en la entrada digital que se utiliza para cambiar el modo de operación de la memoria. Esto se debe a que es fundamental evitar un cambio de modo durante la ejecución de un ciclo de

memoria, ya que podría ocasionar una pérdida de datos en memoria. Esta versión fue llamada *RAM\_MU\_VHDLv11\_1* para separarla del resto y dado que se agregan cambios importantes en algunos componentes RTL. Las pruebas con el filtro simétrico fueron iguales a las anteriores, por lo que se infiere que el funcionamiento no fue afectado; lo cual era esperable. Sin embargo, los tests de paridad y las pruebas en la computadora naval seguían arrojando errores de paridad y no funcionaban en los slots de la unidad paginadora.

El primer problema que se analizó fue el de los errores de paridad, ya que en el pasado ocurrió algo muy similar durante el desarrollo del código *RAM\_MU\_VHDLv10*. El diseño digital se fue probando en distintos equipos que se encuentran en las instalaciones del laboratorio del SIAG y en algunos slots en particular ocurrían problemas de paridad. Todas las capturas se hacían en conjunto con una memoria original, para tener una referencia e identificar cuál señal no se comportaba como el sistema esperaba. Analizando estos puntos y el código, se descubrió el problema. Resulta que en las pruebas iniciales de la versión *RAM\_MU\_VHDLv10\_1* se modificaron unas líneas de descripción del módulo PFF (parity flip-flop) que no fueron revertidas, lo cual es la causa de los errores mencionados. Se trata del principio de funcionamiento del parity flip-flop en el cual una vez la señal de salida se hace '0' lógico no puede volver a '1' lógico al menos que se reinicie el componente. Se realizó una nueva versión del diseño digital llamado *RAM\_MU\_VHDLv11\_2* que contiene estos cambios. La figura 64 muestra una captura de esta versión del diseño digital funcionando correctamente.

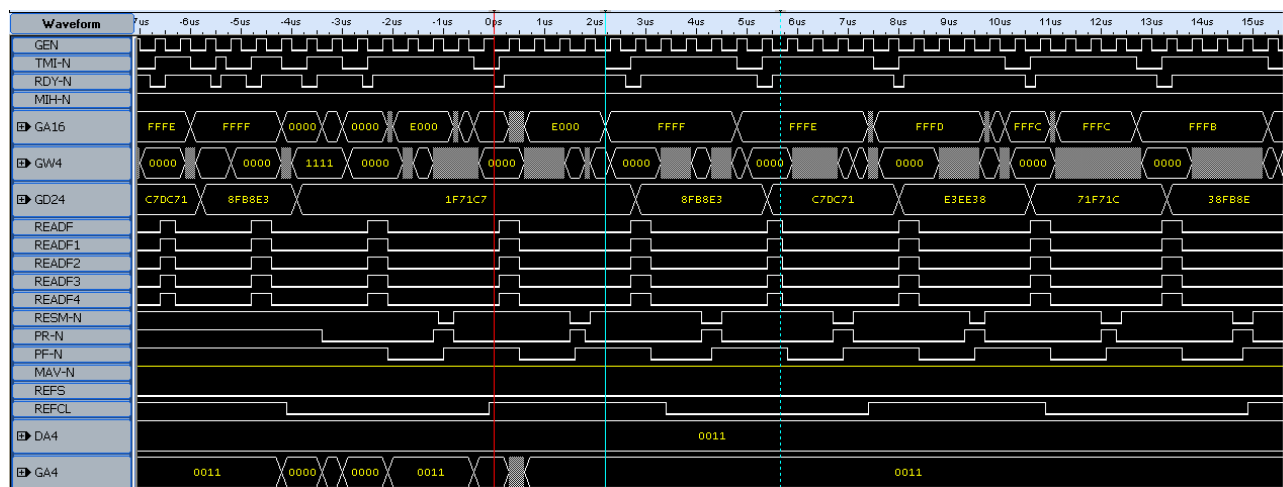


Figura 64 – Captura de la memoria RAM-MU64 funcionando en PC-NAVAL-01.

Esta versión superó exitosamente las pruebas en el equipo PC-NAVAL-01, pero no funcionó correctamente en los slots de la unidad PC-NAVAL-02. Estos slots presentan una particularidad: el pin MIH\_N se utiliza de forma activa para inhabilitar de manera independiente los distintos módulos de memoria y direccionar las páginas de memoria.

Ante esta situación, se realizó un análisis exhaustivo mediante varias capturas adquiridas con el analizador lógico. Entre ellas, se registraron capturas durante el ciclo de encendido de la computadora para obtener los primeros diálogos entre la computadora y la memoria. A partir de la comparación de estos datos con secuencias de la memoria original, se formularon distintas hipótesis. Una de ellas planteaba que la unidad paginadora direccionaba los módulos de memoria RAM de una manera diferente a la del

resto de los slots. Sin embargo, esta opción fue descartada tras consultar el manual de cableado externo entre la placa y la computadora (*Documentación Confidencial - ARA, n.d.*).

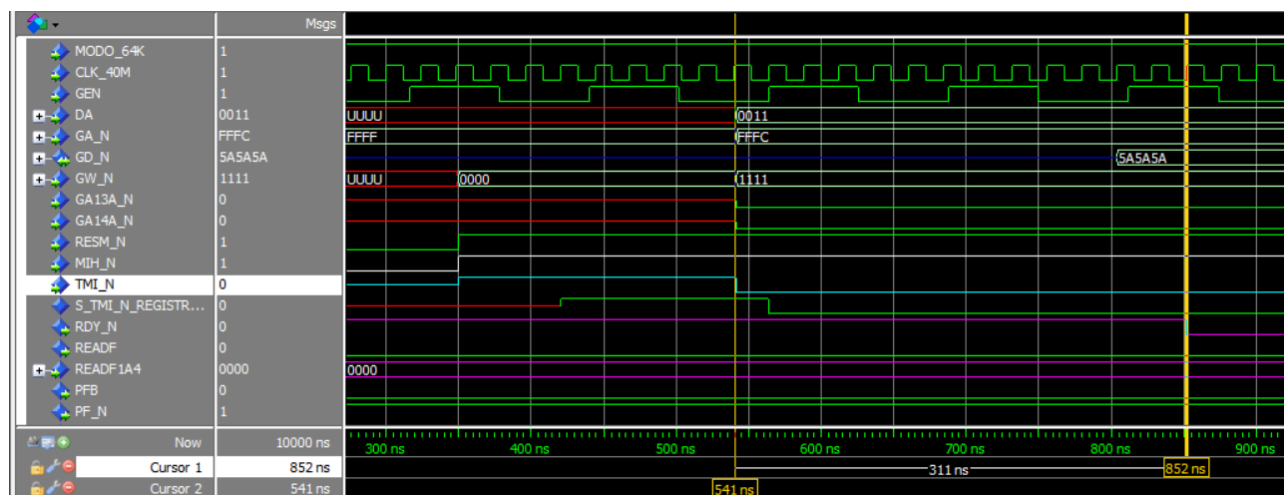
Posteriormente, se descubrió que un cambio realizado en el código para mejorar el tiempo de respuesta del pin RDY\_N ante un desfase de los relojes había provocado que la memoria respondiera más rápido de lo que la computadora puede registrar. La modificación que provocó este cambio fue filtrar la entrada TMI\_N con compuertas AND en vez de compuertas OR. Para demostrar que esa modificación fue la responsable de achicar la duración del pulso negativo del pin RDY\_N, se realizó un nuevo testbench. Este testbench replica el comportamiento de la unidad PC-NAVAL-02 acorde a las capturas realizadas. Este banco de pruebas se puede encontrar en el anexo A.11.

Las simulaciones se realizaron utilizando las dos versiones del filtro. En cada simulación se tomaron los tiempos de RDY\_N ante distintos desfases entre el clock de la lógica digital y el de la computadora. La tabla XIV muestra la comparación de estos resultados de manera condensada.

*Tabla XIV: Respuesta con OR.*

Tiempo de respuesta de RDY_N	Memoria original	RAM-MU64 (filtro AND)	RAM-MU64 (filtro OR)
Ciclo lectura	375 ns	278 ns	402 ns
Ciclo escritura	375 ns	311 ns	431 ns

Se puede apreciar como en el ciclo de escritura, el diseño con el filtro AND responde unos 60ns más rápido que la memoria original. En el de lectura es aún más rápido, casi 100 ns. Esto termina de confirmar que el cambio se debe revertir, puesto que el tiempo mínimo en el cual RDY\_N puede hacerse verdadero es tres ciclos de reloj o 375ns. Este requerimiento se terminó de confirmar con estas pruebas, ya que en la descripción del handshake no está especificado de manera clara. La figura 65 muestra el retraso en un ciclo de escritura para el diseño con el filtro AND.



*Figura 65 – Retraso del pin RDY\_N en ser verdadero con el diseño que falló.*

Analizando el problema de manera detallada también se hizo una nueva observación.

La señal MIH\_N solo estaba siendo filtrada dentro del módulo MAD. Esto estuvo así desde la versión 10 y representa un potencial problema, ya que esa señal es utilizada para la gran mayoría de módulos como reset interno. Por lo tanto, se decidió filtrar la señal en el módulo REGISTRAR ENTRADAS, conectar la señal libre de glitches con el resto del sistema y eliminar el filtrado interno en el bloque MAD para evitar un doble filtrado que ocasionaría un retardo mayor al admitido. Este cambio en el diseño también fue referenciado en la sección 5.2.

Todos los anteriores cambios se condensaron en una nueva versión llamada *RAM\_MU\_VHDLv11\_3*, la cual fue probada una vez más en la unidad PC-NAVAL-03. Todas las pruebas fueron satisfactorias y por lo tanto el diseño final ya fue logrado.

Por último, se realizaron pruebas en dos unidades en donde el prototipo de 16k había fallado. Una de ellas era una computadora que se utiliza como banco de pruebas y otra en la computadora naval de un buque pequeño. Este cambio puede atribuirse a los nuevos buffers bidireccionales que son más poderosos que en la versión 3 del hardware y al filtro colocado en el pin MIH\_N.

## 7. Conclusiones

Durante el desarrollo y la implementación del proyecto RAM-MU64, se lograron todos los objetivos planteados. El dispositivo es capaz de almacenar 16k24 o 64k24 palabras de manera correcta, asegurando un comportamiento similar al de las memorias originales. Por lo tanto, el proyecto RAM-MU64 ha demostrado ser una alternativa viable para la sustitución de las memorias existentes en las computadoras navales de los buques del Comando de la Flota de Mar. Esto representa un ahorro significativo en mantenimiento al reducir hasta en un 75% la sustitución de cuatro memorias.

El diseño digital desarrollado para RAM-MU64 ha demostrado ser robusto y confiable a lo largo del tiempo. Esto fue constatado durante tres años de uso continuo, en donde no se han reportado fallos.

Se completó el armado y prueba del prototipo Versión 4. La última versión que incorpora algunos cambios (Versión 5), no pudo ser probada debido a que no se pasó el prototipo a etapa de producción.

El desarrollo del proyecto fue acorde a lo planificado. Cada etapa se completó dentro del marco de tiempo establecido a priori, con un par de semanas de margen de error. Cabe destacar que, durante todo el periodo de trabajo, el alumno no pudo acceder a las instalaciones físicas del SIAG. Esto requirió un método de trabajo remoto, en el cual el codirector y sus ayudantes realizaron gran parte de los ensayos con los equipos físicos. Esto hizo que algunas pruebas demoren más de lo planeado, pero en ningún momento se generó un retraso significativo.

Gracias a la verificación y análisis exhaustivo de las versiones anteriores a este proyecto, las modificaciones para las nuevas características fueron relativamente sencillas de implementar. También pudieron observarse potenciales problemas que fueron solucionados en el nuevo diseño. Si bien hubo una etapa de depuración de errores, estos fueron principalmente errores de ejecución y no de diseño.

A nivel personal, esta experiencia ha sido muy enriquecedora. Representó la primera oportunidad de trabajar con FPGAs en el *mundo real* y permitió comprender los desafíos y requisitos de implementación que esto conlleva. En cuanto al diseño de hardware, se adquirieron conocimientos clave sobre todo en materia de desarrollo del circuito impreso de cuatro capas. No solamente lo relacionado a lo estrictamente técnico, sino también a los requisitos que una orden de fabricación debe cumplir para que pueda ser realizada por cualquier proveedor del mundo, sin inconvenientes.

En resumen, el proyecto RAM-MU64 representa un logro técnico altamente satisfactorio que culmina un proceso de varias iteraciones. A lo largo del desarrollo, se aplicaron extensamente los conocimientos adquiridos durante la carrera, consolidando y ampliando las habilidades previas.

Se espera que esta solución continúe brindando beneficios a la Flota de Mar y a las Fuerzas Armadas en general.

## 8. Bibliografías y referencias bibliográficas

Bergeron, J. (2003). *Writing testbenches*. Springer US.

*Collecting Code Coverage in Active-HDL - Application Notes - Documentation - Resources*

- *Support - Aldec*. (n.d.). Aldec, Inc. Retrieved March 4, 2024, from

<https://www.aldec.com/en/support/resources/documentation/articles/1012>

*CY7C1049GN30-10ZSXI*. (n.d.). Infineon.

<https://www.infineon.com/cms/en/product/memories/sram-static-ram/asynchronous-sram/cy7c1049gn30-10zsxi/>

*Documentación Confidencial - ARA*. (n.d.).

*Glitch Filter 2.0*. (n.d.). Cypress. Retrieved March 4, 2024, from

[https://www.infineon.com/dgdl/Infineon-Component\\_Glitch\\_Filter\\_V2.0-Software+Module+Datasheets-v02\\_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0e886222183c&utm\\_source=cypress&utm\\_medium=referral&utm\\_campaign=202110\\_globe\\_en\\_all\\_integration-files&redirId=File\\_1\\_3\\_](https://www.infineon.com/dgdl/Infineon-Component_Glitch_Filter_V2.0-Software+Module+Datasheets-v02_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0e886222183c&utm_source=cypress&utm_medium=referral&utm_campaign=202110_globe_en_all_integration-files&redirId=File_1_3_)

*ModelSim HDL simulator | Siemens Software*. (n.d.). Siemens EDA. Retrieved March 4, 2024, from <https://eda.sw.siemens.com/en-US/ic/modelsim/>

*NCP1521B - Buck Converter - DC-DC 1.5 MHz, 600 mA*. (n.d.). Onsemi. Retrieved March 4, 2024, from <https://www.onsemi.com/pdf/datasheet/ncp1521b-d.pdf>

*NCP59300 LDO Regulator*. (n.d.). Onsemi. Retrieved March 4, 2024, from

<https://www.onsemi.com/products/power-management/linear-regulators-ldo/NCP59300>

Salmony, P. (2022, November 28). *PCB Stackup Basics | Phil's Lab | Industry Expert |*

*Altium Designer*. Altium Resources. Retrieved March 4, 2024, from

<https://resources.altium.com/p/pcb-stackup-basics>

*Saturn PCB Toolkit*. (n.d.). Saturn PCB Design, Inc. Retrieved March 4, 2024, from

<https://saturnpcb.com/saturn-pcb-toolkit/>

*7351B table of contents*. (2010, June 3). IPC.org. Retrieved March 4, 2024, from

<https://www.ipc.org/TOC/IPC-7351B.pdf>

*SN74LVC07A - Hex Buffer and Driver*. (n.d.). Texas Instruments. Retrieved March 4, 2024,

from <https://www.ti.com/product/es-mx/SN74LVC07A>

*SN74LVTH2245 - Octal bus transceivers*. (1997, September). Texas Instruments.

Retrieved March 4, 2024, from

[https://www.ti.com/lit/ds/symlink/sn74lvth2245.pdf?ts=1707924349743&ref\\_url=https%253A%252F%252Fwww.google.com%252F](https://www.ti.com/lit/ds/symlink/sn74lvth2245.pdf?ts=1707924349743&ref_url=https%253A%252F%252Fwww.google.com%252F)

*TPS62561 Step-Down Converter*. (n.d.). Texas Instruments. Retrieved March 4, 2024,

from <https://www.ti.com/product/es-mx/TPS62561>

*2221A Table of Contents*. (n.d.). IPC.org. Retrieved March 4, 2024, from

<https://www.ipc.org/TOC/IPC-2221A.pdf>

*Xilinx FPGA*. (n.d.). Xilinx.



Universidad Tecnológica Nacional  
Facultad Regional Bahía Blanca

**Ingeniería Electrónica**

**Proyecto Final**

**Anexo A - Desarrollo de prototipo para ampliación de  
la memoria RAM de computadoras navales, de 16k  
palabras de 24 bits a 64k palabras**

**Autor**

**Diego Hernan Alvarez**

**Director/es**

**Mg. Guillermo Friedrich**

**Ing. Adrian Laiuppa**

**Codirector**

**Ing. Christian L. Galasso**

**Bahía Blanca | 10 de abril de 2024**

## Índice de contenidos

Anexo A: Bancos de prueba (testbenchs) .....	3
A.1 test_registrar_entradas.vhd .....	3
A.2 gen0.vhd.....	7
A.3 MAD.vhd.....	9
A.4 TAC.vhd.....	11
A.5 PARIDAD_V0.vhd.....	17
A.6 generar_paridad_SMR-MU.py.....	20
A.7 test_PFF_v0.vhd.....	22
A.8 test_Gen_par_v0.vhd.....	23
A.9 Prueba y maneja SRAM.vhd .....	25
A.10 RAM_MU.vhd.....	30
A.11 tb_VHDL_ISE_pagger.vhd.....	40

# Anexo A: Bancos de prueba (testbenchs)

## A.1 test\_registrar\_entradas.vhd

```
-- FILEPATH: /d:/OneDrive - ss.z/Proyecto Final/2-Verificacion del HDL/verif_modelsim_RAM-MU-16/test-
RegistrarEntradas-RAM-MU16/registrar_entradas_tb.vhd
library ieee;
use ieee.std_logic_1164.all;

--!
--! El testbench verifica el funcionamiento del módulo registrar_entradas.
--! El modulo registrar_entradas se encarga de registrar las entradas del sistema
--! para que puedan ser utilizadas por otros módulos.
--!
entity registrar_entradas_tb is
end registrar_entradas_tb;
architecture tb_arch of registrar_entradas_tb is
    -- Component declaration
    component registrar_entradas
        port (
            RESM_N           : in std_logic;
            CLK_40M          : in std_logic;
            TMI_N            : in std_logic;
            INICIA_MEMO      : in std_logic;
            GA13A_N          : in std_logic;
            GA14A_N          : in std_logic;
            DA               : in std_logic_vector (4 downto 1);
            GA_N             : in std_logic_vector (16 downto 1);
            GUARDA_WE4A1     : in std_logic;
            GW4A1_N          : in std_logic_vector (4 downto 1);
            PR_N             : in std_logic;
            RESET_INTERNO    : out std_logic;
            TMI_N_REGISTRADA : out std_logic;
            GA13A_N_REGISTRADA : out std_logic;
            GA14A_N_REGISTRADA : out std_logic;
            DA_REGISTRADA    : out std_logic_vector (4 downto 1);
            GA_N_REGISTRADA  : out std_logic_vector (16 downto 1);
            WE4A1_N_REGISTRADA : out std_logic_vector (4 downto 1);
            PR_N_REGISTRADA  : out std_logic
        );
    end component;

    -- Signal declarations
    signal clk_40m           : std_logic := '0';
    signal resm_n           : std_logic := '0';
    signal tmi_n            : std_logic := '0';
```

```

signal inicia_memo      : std_logic          := '0';
signal ga13a_n         : std_logic          := '0';
signal ga14a_n         : std_logic          := '0';
signal da              : std_logic_vector(4 downto 1) := (others => '0');
signal ga_n            : std_logic_vector(16 downto 1) := (others => '0');
signal guarda_we4a1  : std_logic          := '0';
signal gw4a1_n        : std_logic_vector(4 downto 1) := (others => '0');
signal pr_n           : std_logic          := '0';
signal reset_interno  : std_logic;
signal tmi_n_registrada : std_logic;
signal ga13a_n_registrada : std_logic;
signal ga14a_n_registrada : std_logic;
signal da_registrada  : std_logic_vector(4 downto 1);
signal ga_n_registrada : std_logic_vector(16 downto 1);
signal we4a1_n_registrada : std_logic_vector(4 downto 1);
signal pr_n_registrada : std_logic;

-- Clock period definitions
constant clk_40m_period : time := 25 ns; --40MHz

begin
  -- Instantiate the DUT
  dut : registrar_entradas
  port map(
    RESM_N          => resm_n,
    CLK_40M         => clk_40m,
    TMI_N           => tmi_n,
    INICIA_MEMO     => inicia_memo,
    GA13A_N         => ga13a_n,
    GA14A_N         => ga14a_n,
    DA              => da,
    GA_N            => ga_n,
    GUARDA_WE4A1    => guarda_we4a1,
    GW4A1_N         => gw4a1_n,
    PR_N            => pr_n,
    RESET_INTERNO   => reset_interno,
    TMI_N_REGISTRADA => tmi_n_registrada,
    GA13A_N_REGISTRADA => ga13a_n_registrada,
    GA14A_N_REGISTRADA => ga14a_n_registrada,
    DA_REGISTRADA   => da_registrada,
    GA_N_REGISTRADA => ga_n_registrada,
    WE4A1_N_REGISTRADA => we4a1_n_registrada,
    PR_N_REGISTRADA => pr_n_registrada
  );

  -- Clock process
  clk_process : process
begin

```

```

    clk_40m <= '0';
    wait for clk_40m_period/2;
    clk_40m <= '1';
    wait for clk_40m_period/2;
end process;

-- Stimulus process
stimulus_process : process
begin
    -- Comprobar un ciclo normal de funcionamiento.
    resm_n      <= '1';
    tmi_n       <= '1';
    inicia_memo <= '0'; -- No hay un ciclo de memoria, GA y DA deben ser copiadas a la salida.
    ga13a_n     <= '1';
    ga14a_n     <= '1';
    da          <= "0000";
    ga_n        <= "1100101100001101";
    guarda_we4a1 <= '0';
    gw4a1_n     <= "1111";
    pr_n        <= '1';

    wait for clk_40m_period * 5;

    assert reset_interno = '1' report "reset_interno no es '1' cuando RESM_N es '0'" severity error;
    assert tmi_n_registrada = '1' report "tmi_n_registrada no es '1' cuando TMI_N es '1'" severity
error;
    assert ga13a_n_registrada = '1' report "ga13a_n_registrada no es '1' cuando GA13A_N es '1'" severity
error;
    assert ga14a_n_registrada = '1' report "ga14a_n_registrada no es '1' cuando GA14A_N es '1'" severity
error;
    assert da_registrada = "0000" report "da_registrada no es '0000' cuando DA es '0000'" severity
error;
    assert we4a1_n_registrada = "0000" report "we4a1_n_registrada no es '0000' cuando GW4A1_N es '1111'"
severity error;

    -- Test with noise in GW4A1_N for less than two clock cycles
    wait for clk_40m_period;
    gw4a1_n <= "1110";
    wait for clk_40m_period;
    assert we4a1_n_registrada = "0000" report "we4a1_n_registrada no es '0000' cuando GW4A1_N tiene
ruido por menos de dos ciclos de reloj" severity error;

    gw4a1_n <= "1111";
    wait for clk_40m_period * 3;

    assert we4a1_n_registrada = "0000" report "we4a1_n_registrada no es '0000' cuando GW4A1_N tiene
ruido por menos de dos ciclos de reloj" severity error;

```

```

-- Test with noise in GW4A1_N for more than two clock cycles
wait for clk_40m_period;
gw4a1_n <= "1110";
wait for clk_40m_period;
gw4a1_n <= "1110";
wait for clk_40m_period;
gw4a1_n <= "1111";
wait for clk_40m_period * 3;

assert we4a1_n_registrada = "0000" report "we4a1_n_registrada no es '0000' cuando GW4A1_N tiene
ruido por más de dos ciclos de reloj" severity error;

-- Test in normal mode
wait for clk_40m_period;
gw4a1_n <= "1111";
wait for clk_40m_period * 3;

assert we4a1_n_registrada = "0000" report "we4a1_n_registrada no es '0000' en modo normal" severity
error;

-- Test GUARDA_WE4A1 behavior
wait for clk_40m_period;
gw4a1_n <= "1001";
wait for clk_40m_period * 2;
guarda_we4a1 <= '1';
wait for clk_40m_period;
gw4a1_n <= "0000";
wait for clk_40m_period * 4;

assert we4a1_n_registrada = "0110" report "we4a1_n_registrada no es '1111' cuando GUARDA_WE4A1 es
'1'" severity error;

-- Test TMI_N behavior '1' constante con ruido en un periodo.
tmi_n <= '1';
wait for clk_40m_period;
tmi_n <= '0';
wait for clk_40m_period;
tmi_n <= '1';
wait for clk_40m_period;
assert tmi_n_registrada = '1' report "tmi_n_registrada no es '1' cuando TMI_N es '1' constante con
ruido en un periodo" severity error;

-- Test TMI_N behavior '0' constante con ruido en un periodo.
tmi_n <= '0';
wait for clk_40m_period * 3;
assert tmi_n_registrada = '0' report "tmi_n_registrada no es '0' cuando TMI_N es '0' constante con
ruido en un periodo" severity error;
tmi_n <= '1';

```

```

wait for clk_40m_period;
tmi_n <= '0';
wait for clk_40m_period;
assert tmi_n_registrada = '0' report "tmi_n_registrada no es '0' cuando TMI_N es '0' constante con
ruido en un periodo" severity error;

tmi_n <= '1';
ga13a_n <= '1';
ga14a_n <= '1';
da <= "0101";
ga_n <= "1100101100001101";

-- Check that ga13a_n, ga14a_n, da and ga_n are not changed while inicia_memo = '1'.
inicia_memo <= '1';
wait for clk_40m_period * 3;
ga13a_n <= '0';
ga14a_n <= '0';
wait for clk_40m_period;
da <= "1111";
wait for clk_40m_period;
ga_n <= "1010101010101010";
wait for clk_40m_period;
wait;
end process;

end tb_arch;

```

## A.2 gen0.vhd

```

-- Este es un banco de pruebas para el componente GEN_v0.
-- Proporciona señales de estímulo al componente y verifica su comportamiento.
-- FILEPATH: /d:/OneDrive - ss.z/Proyecto Final/2-Verificacion del HDL/verif_modelsim_RAM-MU-16/test-GEN-V0-
RAM-MU16/test_GEN_v0.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity GEN_v0_tb is
end GEN_v0_tb;

architecture Behavioral of GEN_v0_tb is

    -- Component declaration
    component GEN_v0 is
        port (
            RESET_INTERNO : in std_logic;
            CLK_40M       : in std_logic;

```

```

        gen_asinc      : in std_logic;
        pulso_gen_25n : out std_logic);
end component;

-- Signals
signal s_reset_interno : std_logic;
signal s_clk_40M       : std_logic;
signal s_gen_asinc     : std_logic;
signal s_pulso_gen_25n : std_logic;

-- Clock period definitions
constant clk_period      : time := 25 ns;  --40MHz
constant gen_asinc_period : time := 125 ns; --8MHz

begin

-- Instantiate the component
 uut : GEN_v0
port map(s_reset_interno, s_clk_40M, s_gen_asinc, s_pulso_gen_25n);

-- Clock process definitions
clk_process : process
begin
    s_clk_40M <= '0';
    wait for clk_period/2;
    s_clk_40M <= '1';
    wait for clk_period/2;
end process;

-- Clock process definitions
gen_asinc_process : process
begin
    s_gen_asinc <= '0';
    wait for gen_asinc_period/2;
    s_gen_asinc <= '1';
    wait for gen_asinc_period/2;
end process;

-- Stimulus process
stim_proc : process
begin
    s_reset_interno <= '0';
    -- hold reset state for 100 ns.
    wait for 250 ns;
    s_reset_interno <= '1';

    wait;
end process;

```

```
end Behavioral;
```

### A.3 MAD.vhd

```
-- Testbench para el módulo MAD_v0

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_MAD_v0 is
end tb_MAD_v0;

architecture Behavioral of tb_MAD_v0 is

    -- Declaración del componente
    component MAD_v0 is
        Port (
            CLK_40M : in  STD_LOGIC;
            MIH_N : in  STD_LOGIC;
            GA16_A_GA15_N_REG : in  STD_LOGIC_VECTOR (2 downto 1);
            GA14A_A_GA13A_N_REG : in  STD_LOGIC_VECTOR (2 downto 1);
            DA : in  STD_LOGIC_VECTOR (4 downto 1);
            EQ : out  STD_LOGIC
        );
    end component;

    -- Señales
    signal s_clk_40M : STD_LOGIC := 'U';
    signal s_mih_n : STD_LOGIC := 'U';
    signal s_ga16_a_ga15_n_reg : STD_LOGIC_VECTOR (2 downto 1) := "UU";
    signal s_ga14a_a_ga13a_n_reg : STD_LOGIC_VECTOR (2 downto 1) := "UU";
    signal s_da : STD_LOGIC_VECTOR (4 downto 1) := "UUUU";
    signal s_eq : STD_LOGIC;

    -- Definiciones del periodo del reloj
    constant c_CLK_40M_period : time := 25 ns;

begin

    -- Instanciar el módulo MAD_v0
    uut: MAD_v0
        port map (
            CLK_40M => s_clk_40M,
            MIH_N => s_mih_n,
            GA16_A_GA15_N_REG => s_ga16_a_ga15_n_reg,
            GA14A_A_GA13A_N_REG => s_ga14a_a_ga13a_n_reg,
```

```

    DA => s_da,
    EQ => s_eq
);

-- Proceso del reloj
clk_process: process
begin
    s_clk_40M <= '0';
    wait for c_CLK_40M_period*4;
    while now < 1000 ns loop
        s_clk_40M <= not s_clk_40M;
        wait for c_CLK_40M_period;
    end loop;
    wait;
end process;

-- Proceso de estímulo
stimulus_process: process
begin
    -- MIH_N = 1
    s_mih_n <= '1';
    wait for c_CLK_40M_period*6;

    -- Probar diferentes combinaciones de entradas
    s_ga16_a_ga15_n_reg <= "00";
    s_ga14a_a_ga13a_n_reg <= "00";
    s_da <= "0000";
    wait for c_CLK_40M_period/2;
    assert s_eq = '1' report "Prueba 1 fallida" severity error;
    wait for c_CLK_40M_period/2;

    -- Probar diferentes combinaciones de entradas
    s_ga16_a_ga15_n_reg <= "00";
    s_ga14a_a_ga13a_n_reg <= "00";
    s_da <= "0001";
    wait for c_CLK_40M_period/2;
    assert s_eq = '0' report "Prueba 2 fallida" severity error;
    wait for c_CLK_40M_period/2;

    s_ga16_a_ga15_n_reg <= "10";
    s_ga14a_a_ga13a_n_reg <= "00";
    s_da <= "0010";
    wait for c_CLK_40M_period/2;
    assert s_eq = '1' report "Prueba 1 fallida" severity error;
    wait for c_CLK_40M_period/2;

    s_ga16_a_ga15_n_reg <= "11";
    s_ga14a_a_ga13a_n_reg <= "00";

```

```

s_da <= "0011";
wait for c_CLK_40M_period/2;
assert s_eq = '1' report "Prueba 1 fallida" severity error;
wait for c_CLK_40M_period/2;

-- Agregar más casos de prueba aquí
s_mih_n <= '0';
wait for c_CLK_40M_period*2;
assert s_eq = '1' report "MIH_N aún no ha sido capturado." severity error;
wait for c_CLK_40M_period*4;
assert s_eq = '0' report "MIH_N debería haber sido capturado a estas alturas." severity error;

s_mih_n <= '1';
wait for c_CLK_40M_period;
assert s_eq = '1' report "MIH_N debería haber sido capturado a estas alturas." severity error;
wait for c_CLK_40M_period*4;

s_mih_n <= '0';
wait for c_CLK_40M_period;
assert s_eq = '1' report "Debería esperar 3 ciclos para ser '0'." severity error;
s_mih_n <= '1';
wait for c_CLK_40M_period*2;

s_mih_n <= '0';
wait for c_CLK_40M_period*2;
assert s_eq = '1' report "Debería esperar 3 ciclos para ser '0'." severity error;
s_mih_n <= '1';

wait;
end process;

end Behavioral;

```

## A.4 TAC.vhd

```

-----
-----
library ieee;
use ieee.std_logic_1164.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

entity tb_TAC is
end tb_TAC;

```

architecture behavior of tb\_TAC is

```
-- Component Declaration for the Unit Under Test (UUT)
```

```
component TAC
```

```
port (
```

```
    RESET_INT : in std_logic; --reset
    CLK_40M    : in std_logic; --clock 40 MHz
    TMI_N_REG  : in std_logic; --uso la se al tmi registrada del
    --del bloque "registrar entradas" cuando es '0' INICIA CICLO MEMO
    --inicio ciclo normal de memoria (transport memory internal)
    EQ         : in std_logic; --cuando es '1' el modulo de MEMO
    --FUE DIRECCIONADO y debe iniciar el TAC siempre y cuando TMI = '0'
    PULSO_GEN_25NS : in std_logic;
    WE4_1_N    : in std_logic_vector (4 downto 1);

    INICIA_MEMO : out std_logic; --ES UNA SE AL QUE MANTIENE UN '1'
    --CUANDO DEBE INICIAR UN CICLO DE MEMO. NO ES UN FLANCO!!!
    GUARDA_WE4A1 : out std_logic; --SE GUARDA POR '1' EL VALOR DE
    --WE4A1 PARA MANTENER FIJA LA ACCION A REALIZAR POR LA MEMO
    DATO_VALIDO_EN_GBUS : out std_logic; --SE GUARDA POR '1' LOS DATOS DEL
    --GBUS PORQUE EL DATO ES VALIDO (ESTA CORRECTAMENTE ESTABLECIDO)
    RDY_N       : out std_logic; --(ready signal) RDY = 0 inici 
    --ciclo de memoria, RDY = 1 termin  el ciclo de memoria.
    READF       : out std_logic; --con esta se al le digo al CPIO
    --que tengo los datos en los buffers de salida listos a presentarse
    --en el BUS-G, luego el CPIO me responde con las se ales READF1a4
    --que habilitan para escritura en el BUS-G mis buffers de salida
);
```

```
end component;
```

```
--Inputs
```

```
signal s_RESET_INT    : std_logic := '0';
signal s_CLK_40       : std_logic := '0';
signal s_TMI_N_REG    : std_logic;
signal s_EQ           : std_logic := '0';
signal s_PULSO_GEN_25NS : std_logic := '0';
signal s_WE4_1_N     : std_logic_vector(4 downto 1);
```

```
--TMI_N
```

```
signal s_TMI_RAW      : std_logic;
signal s_TMI_N_ant1  : std_logic;
signal s_TMI_N_ant2  : std_logic;
signal s_TMI_N_ant3  : std_logic;
```

```
--GW4A1_N
```

```
signal s_CICLO_MEMORIA : std_logic_vector(4 downto 1);
signal s_WE4_1_N_RAW   : std_logic_vector(4 downto 1);
signal s_GW4A1_N_ANT1 : std_logic_vector(4 downto 1);
```

```

signal S_GW4A1_N_ANT2 : std_logic_vector(4 downto 1);

--Outputs
signal s_INICIA_MEMO      : std_logic;
signal s_GUARDA_WE4A1    : std_logic;
signal s_DATO_VALIDO_EN_GBUS : std_logic;
signal s_RDY_N           : std_logic;
signal s_READF           : std_logic;

-- Clock period definitions
constant CLK_period      : time := 25 ns; --CLK = 40MHz
constant CLK_period_2    : time := 125 ns; --CLK = 8MHz
constant t1              : time := 10 ns; --Tiempo minimo de establecimiento 10 ns tiempo máximo 90ns.

procedure ciclo_memoria (
    signal S_CICLO_MEMORIA : in std_logic_vector(4 downto 1);
    signal PULSO_GEN_25NS  : in std_logic;
    signal RDY_N           : in std_logic;
    signal GUARDA_WE4A1    : in std_logic;
    signal RESET_INT       : out std_logic;
    signal TMI_RAW         : out std_logic;
    signal WE4_1_N_RAW     : out std_logic_vector(4 downto 1);
    signal EQ              : out std_logic
)
is
variable v_TIME : time := 0 ns; -- Se asigna una variable para determinar el tiempo de ejecucion.
begin
    -- Comienzo el ciclo de escritura con un desfase t1 del flanco reloj.
    -- Momento cero del handshake con el CP.
    -- Aca el master coloca los valores de GA (por lo tanto EQ) y la señal TMI_N_REG.
    wait until rising_edge(PULSO_GEN_25NS);
    RESET_INT <= '1';
    v_TIME := now;
    wait for t1;
    TMI_RAW <= '0';
    EQ      <= '1';

    -- El master coloca los valores de WE4_1_N.
    wait until rising_edge(PULSO_GEN_25NS);
    wait for t1;
    WE4_1_N_RAW <= S_CICLO_MEMORIA;

    -- El tiempo mínimo en cuanto TMI puede hacerse '1' es 250ns.
    wait until s_RDY_N = '0';
    wait until rising_edge(PULSO_GEN_25NS);
    wait for t1;
    TMI_RAW <= '1';

```

```

    wait until RDY_N = '1' for 300 ns;
    v_TIME := now - v_TIME;
    report "Duracion del periodo de lectura = " & time'image(v_TIME);
    EQ <= '0';

end procedure;

begin

-- Instantiate the Unit Under Test (UUT)
 uut : TAC port map(
    RESET_INT      => s_RESET_INT,
    CLK_40M        => s_CLK_40,
    TMI_N_REG      => s_TMI_N_REG,
    EQ             => s_EQ,
    PULSO_GEN_25NS => s_PULSO_GEN_25NS,
    WE4_1_N       => s_WE4_1_N,
    INICIA_MEMO   => s_INICIA_MEMO,
    GUARDA_WE4A1  => s_GUARDA_WE4A1,
    DATO_VALIDO_EN_GBUS => s_DATO_VALIDO_EN_GBUS,
    RDY_N         => s_RDY_N,
    READF         => s_READF
  );

-- Clock process definitions
CLK_process : process
begin
    s_CLK_40 <= '1';
    wait for CLK_period/2;
    s_CLK_40 <= '0';
    wait for CLK_period/2;
end process;

CLK_8_process : process
begin
    s_PULSO_GEN_25NS <= '1';
    wait for CLK_period_2/2;
    s_PULSO_GEN_25NS <= '0';
    wait for CLK_period_2/2;
end process;

-- TMI_N_REG Tiene que estar tres ciclos reloj retrasada respecto a TMI_N_RAW.
--PROCEDEMOS A REGISTRAR LA SEÑAL TMI_N QUE ES LA QUE INICIA UN CICLO DE MEMORIA
registra_TMI_N : process (s_CLK_40)
begin

    --LA señal TMI-N cuando pasa a '0' inicia un ciclo de memoria

```

```

--es por ello que se la registra SIEMPRE eh indefinidamente
--ademas se filtran los glitches con OR
if rising_edge(s_CLK_40) then
    s_TMI_N_ant1 <= s_TMI_RAW;
    s_TMI_N_ant2 <= s_TMI_N_ant1;
    s_TMI_N_ant3 <= s_TMI_N_ant2;
end if;
end process;

--GENERO UNA SEÑAL DE TMI REGISTRADA Y FILTRADA DE GLITCHES
s_TMI_N_REG <= (s_TMI_N_ant1 or s_TMI_N_ant2 or s_TMI_N_ant3);

_*****
--A CONTINUACIÓN SE REGISTRA LA ACCIÓN A REALIZAR POR LA MEMORIA CONTENIDA EN
--LOS PINES GW
registra_ACCION_MEMO : process (s_CLK_40, s_GUARDA_WE4A1, s_WE4_1_N_RAW, S_GW4A1_N_ANT1, S_GW4A1_N_ANT2)
begin

    if rising_edge(s_CLK_40) then
        if s_GUARDA_WE4A1 = '0' then --SI NO INICIAMOS UN CICLO DE MEMO ENTONCES
            --MANTENEMOS CAPTURANDO LOS PINES DE SELECCIÓN DE OPERACIÓN DE LA MEMO
            S_GW4A1_N_ANT1 <= s_WE4_1_N_RAW;
            S_GW4A1_N_ANT2 <= S_GW4A1_N_ANT1;
        end if;
    end if;
end process;

s_WE4_1_N <= not(S_GW4A1_N_ANT2);--RECORDEMOS QUE WE ES GW NEGADO
-- Stimulus process
stim_secuencia_inicio : process
    variable v_TIME : time := 0 ns; -- Se asigna una variable para determinar el tiempo de ejecucion.
begin
    -- hold reset state for 100 ns.
    s_RESET_INT <= '0';
    s_TMI_RAW <= '1';
    s_WE4_1_N_RAW <= "UUUU";
    s_EQ <= '0';

    -- Comienzo el ciclo de lectura con un desfase t1 del flanco reloj.
    wait until rising_edge(s_PULSO_GEN_25NS);

    -- Momento cero del handshake con el CP.
    -- Aca el master coloca los valores de GA (por lo tanto EQ) y la señal TMI_N_REG.
    wait until rising_edge(s_PULSO_GEN_25NS);
    v_TIME := now;
    s_RESET_INT <= '1';
    wait for t1;
    s_TMI_RAW <= '0';

```

```

s_EQ      <= '1';

-- El master coloca los valores de WE4_1_N.
wait until rising_edge(s_PULSO_GEN_25NS);
wait for t1;
s_WE4_1_N_RAW <= "0000";

-- El tiempo mínimo en cuanto TMI puede hacerse '1' es 250ns.
wait until s_RDY_N = '0';
wait until rising_edge(s_PULSO_GEN_25NS);
wait for t1;
s_TMI_RAW <= '1';

wait until s_RDY_N = '1' for 300 ns;
v_TIME := now - v_TIME;
report "Duracion del periodo de lectura = " & time'image(v_TIME);
s_EQ <= '0';

-- Escritura total.
wait until rising_edge(s_PULSO_GEN_25NS);
wait until rising_edge(s_PULSO_GEN_25NS);
s_CICLO_MEMORIA <= "1111";
ciclo_memoria(s_CICLO_MEMORIA, s_PULSO_GEN_25NS, s_RDY_N, s_GUARDA_WE4A1, s_RESET_INT, s_TMI_RAW,
s_WE4_1_N_RAW, s_EQ);

-- Escritura de palabra superior.
wait until rising_edge(s_PULSO_GEN_25NS);
wait until rising_edge(s_PULSO_GEN_25NS);
s_CICLO_MEMORIA <= "1100";
ciclo_memoria(s_CICLO_MEMORIA, s_PULSO_GEN_25NS, s_RDY_N, s_GUARDA_WE4A1, s_RESET_INT, s_TMI_RAW,
s_WE4_1_N_RAW, s_EQ);

-- Escritura de palabra inferior.
wait until rising_edge(s_PULSO_GEN_25NS);
wait until rising_edge(s_PULSO_GEN_25NS);
s_CICLO_MEMORIA <= "0011";
ciclo_memoria(s_CICLO_MEMORIA, s_PULSO_GEN_25NS, s_RDY_N, s_GUARDA_WE4A1, s_RESET_INT, s_TMI_RAW,
s_WE4_1_N_RAW, s_EQ);
wait;
end process;

end;

```

## A.5 PARIDAD\_V0.vhd

```
-- FILEPATH: /d:/OneDrive - ss.z/Proyecto Final/2-Verificacion del HDL/verif_modelsim_RAM-MU-16/test-
Gen_par_v0-RAM-MU16/PARIDAD_v0_tb.vhd
library IEEE;
use IEEE.std_logic_1164.all;

entity PARIDAD_v0_tb is
end PARIDAD_v0_tb;

architecture Behavioral of PARIDAD_v0_tb is

-- Component declaration
component PARIDAD_v0
port (
    GBUS_IN      : in std_logic_vector (24 downto 1);
    SRAM_IN      : in std_logic_vector (32 downto 1);
    RESM_N       : in std_logic;
    CLK          : in std_logic;
    READF        : in std_logic;
    PR_N         : in std_logic;
    PULSO_READF_25NS : in std_logic;
    SRAM_OUT     : out std_logic_vector (32 downto 1);
    GBUS_OUT     : out std_logic_vector (24 downto 1);
    PF           : out std_logic;
    PF_LED       : out std_logic
);
end component;

-- Signal declarations
signal GBUS_IN_tb      : std_logic_vector (24 downto 1);
signal SRAM_IN_tb     : std_logic_vector (32 downto 1);
signal RESM_N_tb     : std_logic;
signal CLK_tb         : std_logic;
signal READF_tb      : std_logic;
signal PR_N_tb       : std_logic;
signal PULSO_READF_25NS_tb : std_logic;
signal SRAM_OUT_tb   : std_logic_vector (32 downto 1);
signal GBUS_OUT_tb   : std_logic_vector (24 downto 1);
signal PF_tb         : std_logic;
signal PF_LED_tb     : std_logic;

signal s_PARIDAD_SRAM_OUT : std_logic_vector (4 downto 1);

constant paridad_par      : std_logic := '1';
constant paridad_impar    : std_logic := '0';
```

```

constant readf_escritura : std_logic := '0';
constant readf_lectura  : std_logic := '1';
begin

s_PARIDAD_SRAM_OUT <= SRAM_OUT_tb(28 downto 25);

-- Instantiate the module under test
UUT : PARIDAD_v0
port map(
    GBUS_IN      => GBUS_IN_tb,
    SRAM_IN      => SRAM_IN_tb,
    RESM_N       => RESM_N_tb,
    CLK          => CLK_tb,
    READF        => READF_tb,
    PR_N         => PR_N_tb,
    PULSO_READF_25NS => PULSO_READF_25NS_tb,
    SRAM_OUT     => SRAM_OUT_tb,
    GBUS_OUT     => GBUS_OUT_tb,
    PF           => PF_tb,
    PF_LED       => PF_LED_tb
);

-- clock process
clk_proc : process
begin
    CLK_tb <= '0';
    wait for 25 ns;
    CLK_tb <= '1';
    wait for 25 ns;
end process;

-- Create READF synced with CLK
READF_sync_proc : process (CLK_tb)
begin
    if rising_edge(CLK_tb) then
        PULSO_READF_25NS_tb <= READF_tb;
    end if;
end process;

-- Stimulus process
stim_proc : process
begin
    RESM_N_tb <= '0';
    -- For loop 5 times waiting for rising_edge.
    wait until rising_edge(CLK_tb);
    wait until rising_edge(CLK_tb);
    wait until rising_edge(CLK_tb);
    wait until rising_edge(CLK_tb);

```

```

wait until rising_edge(CLK_tb);

-- Initialize inputs
RESM_N_tb <= '1';
PR_N_tb   <= paridad_impar;
READF_tb <= readf_escritura;
GBUS_IN_tb <= (others => '0');
SRAM_IN_tb <= (others => 'X');
wait until rising_edge(CLK_tb);
wait until rising_edge(CLK_tb);

-- Test 1: Checkeo de paridad en un ciclo de escritura con paridad par.
GBUS_IN_tb <= x"41154B";
PR_N_tb   <= paridad_par;
READF_tb <= readf_escritura;
wait until rising_edge(CLK_tb);
assert s_PARIDAD_SRAM_OUT = "1011" report "ERROR: PARIDAD_SRAM_OUT = 1000" severity failure;
assert PF_tb = '0' report "ERROR: PF_tb = '0'" severity failure;
wait until rising_edge(CLK_tb);

-- Test 2: Checkeo de paridad en un ciclo de escritura con paridad impar.
GBUS_IN_tb <= x"41154B";
PR_N_tb   <= paridad_impar;
READF_tb <= readf_escritura;
wait until rising_edge(CLK_tb);
assert s_PARIDAD_SRAM_OUT = "0100" report "ERROR: PARIDAD_SRAM_OUT = 0111" severity failure;
assert PF_tb = '0' report "ERROR: PF_tb = '0'" severity failure;
wait until rising_edge(CLK_tb);

wait until rising_edge(CLK_tb);

-- Test 3: Checkeo de paridad en un ciclo de escritura con paridad par, sin fallo de paridad.
SRAM_IN_tb <= x"04A7A4DC";
PR_N_tb   <= paridad_par;
READF_tb <= readf_lectura;
wait until rising_edge(CLK_tb);
READF_tb <= '0';
wait until rising_edge(CLK_tb);
assert GBUS_OUT_tb = x"A7A4DC" report "ERROR: GBUS_OUT_tb != A7A4DC" severity failure;
assert PF_tb = '0' report "ERROR: PF_tb != '0'" severity failure;
wait until rising_edge(CLK_tb);

-- Test 4: Checkeo de paridad en un ciclo de escritura con paridad impar, sin fallo de paridad.
SRAM_IN_tb <= x"0BA7A4DC";
PR_N_tb   <= paridad_impar;
READF_tb <= readf_lectura;
wait until rising_edge(CLK_tb);
READF_tb <= '0';

```

```

wait until rising_edge(CLK_tb);
assert GBUS_OUT_tb = x"A7A4DC" report "ERROR: GBUS_OUT_tb != A7A4DC" severity failure;
assert PF_tb = '0' report "ERROR: PF_tb != '0'" severity failure;
wait until rising_edge(CLK_tb);
wait until rising_edge(CLK_tb);

-- Test 5: Checkeo de paridad en un ciclo de escritura con paridad par, con fallo de paridad.
SRAM_IN_tb <= x"00FDCACE";
PR_N_tb <= paridad_par;
READF_tb <= readf_lectura;
wait until rising_edge(CLK_tb);
READF_tb <= '0';
wait until rising_edge(CLK_tb);
wait until rising_edge(CLK_tb);
assert GBUS_OUT_tb = x"FDCACE" report "ERROR: GBUS_OUT_tb != FDCACE" severity failure;
assert PF_tb = '1' report "ERROR: PF_tb != '0'" severity error;
wait until rising_edge(CLK_tb);
wait until rising_edge(CLK_tb);

-- Reinicio el PFF.
RESM_N_tb <= '0';
wait until rising_edge(CLK_tb);
wait until rising_edge(CLK_tb);
RESM_N_tb <= '1';

-- Test 6: Checkeo de paridad en un ciclo de escritura con paridad impar, con fallo de paridad.
SRAM_IN_tb <= x"00FDCACE";
PR_N_tb <= paridad_impar;
READF_tb <= readf_lectura;
wait until rising_edge(CLK_tb);
READF_tb <= '0';
wait until rising_edge(CLK_tb);
wait until rising_edge(CLK_tb);
assert GBUS_OUT_tb = x"FDCACE" report "ERROR: GBUS_OUT_tb != FDCACE" severity failure;
assert PF_tb = '1' report "ERROR: PF_tb != '0'" severity error;
wait until rising_edge(CLK_tb);
wait until rising_edge(CLK_tb);
wait;
end process;

end Behavioral;

```

## A.6 generar\_paridad\_SMR-MU.py

```
import random

def generate_gbus_data():
    """
    Genera un dato hexadecimal aleatorio de 24 bits y devuelve una cadena.
    """
    return hex(random.randint(0, 0xFFFFFFFF)[2:].upper().zfill(6))

def calculate_parity_bits(hex_data, parity_invert=False):
    """
    Calcula los bits de paridad para los datos hexadecimales de 24 bits utilizando el esquema de paridad
    especificado en
    la documentacion del SMR-MU.

    Args:
        hex_data: Una cadena que representa los datos hexadecimales de 24 bits.
        parity_invert: Un booleano que indica si se debe utilizar paridad par (False) o paridad impar
        (True).

    Returns:
        Una cadena con los bits de paridad calculados en formato binario.
    """
    data = int(hex_data, 16) # Convierte la cadena hexadecimal en un entero
    groups = [
        (data >> 16) & 0xFF, # Primeros 8 bits
        (data >> 12) & 0xF, # Segundos 4 bits
        (data >> 8) & 0xF, # Terceros 4 bits
        data & 0xFF, # Últimos 8 bits
    ]

    parity_bits = ""
    for group in groups:
        parity_bit = 0
        for bit in bin(group)[2:].zfill(8): # Extrae y procesa cada bit
            parity_bit ^= int(bit)
        parity_bit ^= parity_invert # Invierte el bit de paridad si es necesario
        parity_bits += str(parity_bit)

    return parity_bits

hex_data = generate_gbus_data()
parity_bits_par = calculate_parity_bits(hex_data, parity_invert=True) # Bit de paridad par.
parity_bits_impar = calculate_parity_bits(hex_data, parity_invert=False) # Bit de paridad impar.

print("Original hex data:", hex_data)
print("Original binary data:", bin(int(hex_data, 16))[2:].zfill(24))
```

```
print("Parity bits (PAR):", parity_bits_par)
print("Parity bits (IMPAR):", parity_bits_impar)
```

## A.7 test\_PFF\_v0.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_PFF_v0 IS
END tb_PFF_v0;

ARCHITECTURE behavior OF tb_PFF_v0 IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT PFF_v0
    Port ( RESM_N :          in  STD_LOGIC;
          CLK       :          in  STD_LOGIC;
          READF     :          in  STD_LOGIC;
          PFA       :          in  STD_LOGIC;
          PULSO_READF_25NS : in  STD_LOGIC; --viene de RAM_MU
          PF        :          out STD_LOGIC;
          PF_LED    :          out STD_LOGIC
        );
    END COMPONENT;

    --Inputs
    signal s_resm_n : std_logic := 'U';
    signal s_clk    : std_logic := 'U';
    signal s_readf  : std_logic := 'U';
    signal s_pfa    : std_logic := 'U';
    signal s_pulso_readf_25ns : std_logic := 'U';

    --Outputs
    signal s_PF : std_logic := 'U';
    signal s_PF_LED : std_logic := 'U';

    -- Clock period definitions
    constant clk_period : time := 25 ns; --40MHz

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: PFF_v0 PORT MAP (
        RESM_N => s_resm_n,
```

```

        CLK => s_clk,
        READF => s_readf,
        PFA => s_pfa,
        PULSO_READF_25NS => s_pulso_readf_25ns,
        PF => s_PF,
        PF_LED => s_PF_LED
    );

-- Clock process definitions
clk_process : process
begin
    s_clk <= '1';
    wait for clk_period/2;
    s_clk <= '0';
    wait for clk_period/2;
end process;

-- Stimulus process
stim_proc: process
begin
    -- La unica combinacion que debería hacer s_PF y PF_LED '1' es:
    -- PFA = 1 - READF = 0 - PULSO_READF_25ns = 1

    wait for 100 ns;
    s_resm_n <= '0';

    wait for 200 ns;
    s_resm_n <= '1';

    wait for 100 ns;
    s_pfa <= '1';
    s_readf <= '0';
    s_pulso_readf_25ns <= '1';

    wait for 200 ns;
    s_pfa <= '0';
    s_readf <= '0';
    s_pulso_readf_25ns <= '1';

    wait;
end process;

END;
```

## A.8 test\_Gen\_par\_v0.vhd

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_GEN_PAR_v0 IS
END tb_GEN_PAR_v0;

ARCHITECTURE behavior OF tb_GEN_PAR_v0 IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT GEN_PAR_v0
    Port ( PR_N :      in  STD_LOGIC;
          --PR_N = 0 paridad IMPAR, PR_N = 1 paridad PAR
          DATOS_IN  : in  STD_LOGIC_VECTOR (24 downto 1);
          --entrada de datos para controlar paridad
          DATOS_OUT : out STD_LOGIC_VECTOR (32 downto 1)
          --salida de 24 bits de datos mas 4 bits de paridad
          );
    END COMPONENT;

    --Inputs
    signal s_PR_N : std_logic := 'U';
    signal s_DATOS_IN : STD_LOGIC_VECTOR (24 downto 1) := (others => 'U');

    --Outputs
    signal s_DATOS_OUT : STD_LOGIC_VECTOR (32 downto 1) := (others => 'U');

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: GEN_PAR_v0 PORT MAP (
        PR_N => s_PR_N,
        DATOS_IN => s_DATOS_IN,
        DATOS_OUT => s_DATOS_OUT
    );

    -- Stimulus process
    stim_proc: process
    begin
        -- PR_N=1 paridad PAR PR_N=0 paridad IMPAR

        wait for 10 ns;

        s_PR_N <= '0';
    
```

```

wait for 10 ns;

s_DATOS_IN(8 downto 1) <= "0000011"; -- Paridad par 0
s_DATOS_IN(12 downto 9) <= "0010"; -- Paridad par 1
s_DATOS_IN(16 downto 13) <= "0101"; -- Paridad par 0
s_DATOS_IN(24 downto 17) <= "00000110"; -- Paridad par 0

wait for 50 ns;

assert s_DATOS_OUT(25) = '0' report "ERROR DE PARIDAD." severity note;
assert s_DATOS_OUT(26) = '1' report "ERROR DE PARIDAD." severity note;
assert s_DATOS_OUT(27) = '0' report "ERROR DE PARIDAD." severity note;
assert s_DATOS_OUT(28) = '0' report "ERROR DE PARIDAD." severity note;

wait for 10 ns;

s_PR_N <= '1';

wait for 50 ns;

assert s_DATOS_OUT(25) = '1' report "ERROR DE PARIDAD." severity note;
assert s_DATOS_OUT(26) = '0' report "ERROR DE PARIDAD." severity note;
assert s_DATOS_OUT(27) = '1' report "ERROR DE PARIDAD." severity note;
assert s_DATOS_OUT(28) = '1' report "ERROR DE PARIDAD." severity note;

wait;
end process;

END;
```

## A.9 Prueba y maneja SRAM.vhd

```

-- FILEPATH: /d:/OneDrive - ss.z/Proyecto Final/2-Verificacion del HDL/verif_modelsim_RAM-MU-16/test-
Prueba_FSM_y_ManejaSRAM-RAM-MU16/Prueba_FSM_y_MANEJASRAM_tb.vhd
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Prueba_FSM_y_MANEJASRAM_tb is
end Prueba_FSM_y_MANEJASRAM_tb;

architecture Behavioral of Prueba_FSM_y_MANEJASRAM_tb is

-- Component declaration
```

```

component Prueba_FSM_y_MANEJASRAM is
  Port (
    CLK_40M : in STD_LOGIC;
    RESM_N : in STD_LOGIC;
    WE4_1_N : in STD_LOGIC_VECTOR (4 downto 1);
    INICIA : in STD_LOGIC;
    DATOS_DESDE_PARITY : in STD_LOGIC_VECTOR (32 downto 1);
    GA14_01_N : in STD_LOGIC_VECTOR (14 downto 1);
    DATO_VALIDO_EN_GBUS : in STD_LOGIC;
    DATOS_SRAM_IN : in STD_LOGIC_VECTOR (8 downto 1);
    CE_N : out STD_LOGIC;
    OE_N : out STD_LOGIC;
    WE_N : out STD_LOGIC;
    OE_FPGA_A_SRAM : out STD_LOGIC;
    DATOS_HACIA_PARITY : out STD_LOGIC_VECTOR (32 downto 1);
    A : out STD_LOGIC_VECTOR (17 downto 1);
    DATOS_SRAM_OUT : out STD_LOGIC_VECTOR (8 downto 1)
  );
end component;

-- Signal declarations
signal s_CLK_40M : STD_LOGIC := '0';
signal s_RESM_N : STD_LOGIC := '0';
signal s_WE4_1_N : STD_LOGIC_VECTOR (4 downto 1) := (others => '0');
signal s_INICIA : STD_LOGIC := '0';
signal s_DATOS_DESDE_PARITY : STD_LOGIC_VECTOR (32 downto 1) := (others => '0');
signal s_GA14_01_N : STD_LOGIC_VECTOR (14 downto 1) := (others => '0');
signal s_DATO_VALIDO_EN_GBUS : STD_LOGIC := '0';
signal s_DATOS_SRAM_IN : STD_LOGIC_VECTOR (8 downto 1) := (others => '0');
signal s_CE_N : STD_LOGIC;
signal s_OE_N : STD_LOGIC;
signal s_WE_N : STD_LOGIC;
signal s_OE_FPGA_A_SRAM : STD_LOGIC;
signal s_DATOS_HACIA_PARITY : STD_LOGIC_VECTOR (32 downto 1);
signal s_A : STD_LOGIC_VECTOR (17 downto 1);
signal s_DATOS_SRAM_OUT : STD_LOGIC_VECTOR (8 downto 1);

begin

-- Instantiate the DUT
 uut: Prueba_FSM_y_MANEJASRAM
  port map (
    CLK_40M => s_CLK_40M,
    RESM_N => s_RESM_N,
    WE4_1_N => s_WE4_1_N,
    INICIA => s_INICIA,
    DATOS_DESDE_PARITY => s_DATOS_DESDE_PARITY,
    GA14_01_N => s_GA14_01_N,
  );
end;

```

```

DATO_VALIDO_EN_GBUS => s_DATO_VALIDO_EN_GBUS,
DATOS_SRAM_IN => s_DATOS_SRAM_IN,
CE_N => s_CE_N,
OE_N => s_OE_N,
WE_N => s_WE_N,
OE_FPGA_A_SRAM => s_OE_FPGA_A_SRAM,
DATOS_HACIA_PARITY => s_DATOS_HACIA_PARITY,
A => s_A,
DATOS_SRAM_OUT => s_DATOS_SRAM_OUT
);

-- Clock process
process
begin
    while now < 4000 ns loop
        s_CLK_40M <= '0';
        wait for 25 ns;
        s_CLK_40M <= '1';
        wait for 25 ns;
    end loop;
    wait;
end process;

-- Stimulus process
process
begin
    s_RESM_N <= '0';
    wait until rising_edge(s_CLK_40M);
    wait until rising_edge(s_CLK_40M);
    wait until rising_edge(s_CLK_40M);
    s_RESM_N <= '1';
    wait until rising_edge(s_CLK_40M);

    --! Ciclo de escritura completa.
    s_WE4_1_N <= "0000"; --! Write cycle.
    s_DATOS_DESDE_PARITY <= x"1000001"; --! Datos con el checksum hecho (a escribir).
    s_GA14_01_N <= "000000000001"; --! Direccion de memoria.
    s_DATO_VALIDO_EN_GBUS <= '1';
    s_DATOS_SRAM_IN <= x"01"; --! Dato en SRAM direccion GA14_01_N + dos bits.
    wait until rising_edge(s_CLK_40M);
    s_INICIA <= '1';
    wait until rising_edge(s_CLK_40M);
    wait until rising_edge(s_CLK_40M);
    wait for 1 ns;
    assert s_DATOS_SRAM_OUT = s_DATOS_DESDE_PARITY(8 downto 1) report "Error en el dato leído de la
SRAM" severity error;
    for i in 1 to 2 loop
        wait until rising_edge(s_CLK_40M);

```

```

end loop;
wait for 1 ns;
assert s_DATOS_SRAM_OUT = s_DATOS_DESDE_PARITY(16 downto 9) report "Error en el dato leído de la
SRAM" severity error;
for i in 1 to 2 loop
    wait until rising_edge(s_CLK_40M);
end loop;
wait for 1 ns;
assert s_DATOS_SRAM_OUT = s_DATOS_DESDE_PARITY(24 downto 17) report "Error en el dato leído de la
SRAM" severity error;
for i in 1 to 2 loop
    wait until rising_edge(s_CLK_40M);
end loop;
wait for 1 ns;
assert s_DATOS_SRAM_OUT = s_DATOS_DESDE_PARITY(32 downto 25) report "Error en el dato leído de la
SRAM" severity error;
for i in 1 to 11 loop
    wait until rising_edge(s_CLK_40M);
end loop;
s_INICIA <= '0';

for i in 1 to 4 loop
    wait until rising_edge(s_CLK_40M);
end loop;

--! Ciclo de lectura.
s_WE4_1_N <= "1111"; --! Ciclo de lectura
s_DATOS_DESDE_PARITY <= x"1000001"; --! Datos con el checksum hecho (a escribir).
s_GA14_01_N <= "000000000001"; --! Direccion de memoria.
s_DATO_VALIDO_EN_GBUS <= '0';
s_DATOS_SRAM_IN <= x"AA"; --! Dato en SRAM direccion GA14_01_N + dos bits.
wait until rising_edge(s_CLK_40M);
s_INICIA <= '1';
for i in 1 to 2 loop
    wait until rising_edge(s_CLK_40M);
end loop;
wait for 1 ns;
assert s_DATOS_HACIA_PARITY(8 downto 1) = s_DATOS_SRAM_IN report "Error en el dato leído de la SRAM"
severity error;
wait until rising_edge(s_CLK_40M);
s_DATOS_SRAM_IN <= x"BB";
wait until rising_edge(s_CLK_40M);
wait for 1 ns;
assert s_DATOS_HACIA_PARITY(16 downto 9) = s_DATOS_SRAM_IN report "Error en el dato leído de la
SRAM" severity error;
wait until rising_edge(s_CLK_40M);
s_DATOS_SRAM_IN <= x"CC";
wait until rising_edge(s_CLK_40M);

```

```

wait for 1 ns;
assert s_DATOS_HACIA_PARITY(24 downto 17) = s_DATOS_SRAM_IN report "Error en el dato leído de la
SRAM" severity error;
wait until rising_edge(s_CLK_40M);
s_DATOS_SRAM_IN <= x"DD";
wait until rising_edge(s_CLK_40M);
wait for 1 ns;
assert s_DATOS_HACIA_PARITY(32 downto 25) = s_DATOS_SRAM_IN report "Error en el dato leído de la
SRAM" severity error;
wait until rising_edge(s_CLK_40M);
s_INICIA <= '0';

for i in 1 to 4 loop
    wait until rising_edge(s_CLK_40M);
end loop;

--! Ciclo de escritura parcial.
s_WE4_1_N <= "1100"; --! Ciclo escritura de la primera mitad.
s_DATOS_DESDE_PARITY <= x"00FFFAAA"; --! Datos con el checksum hecho (a escribir).
s_GA14_01_N <= "0000000000100"; --! Direccion de memoria.
s_DATO_VALIDO_EN_GBUS <= '1';
s_DATOS_SRAM_IN <= x"00"; --! Dato en SRAM direccion GA14_01_N + dos bits.
wait until rising_edge(s_CLK_40M);
s_INICIA <= '1';
for i in 1 to 2 loop
    wait until rising_edge(s_CLK_40M);
end loop;
wait for 1 ns;
assert s_DATOS_HACIA_PARITY(8 downto 1) = s_DATOS_SRAM_IN report "Error en el dato leído de la SRAM"
severity error;
wait until rising_edge(s_CLK_40M);
s_DATOS_SRAM_IN <= x"D0";
wait until rising_edge(s_CLK_40M);
wait for 1 ns;
assert s_DATOS_HACIA_PARITY(16 downto 9) = s_DATOS_SRAM_IN report "Error en el dato leído de la
SRAM" severity error;
wait until rising_edge(s_CLK_40M);
s_DATOS_SRAM_IN <= x"DD";
wait until rising_edge(s_CLK_40M);
wait for 1 ns;
assert s_DATOS_HACIA_PARITY(24 downto 17) = s_DATOS_SRAM_IN report "Error en el dato leído de la
SRAM" severity error;
wait until rising_edge(s_CLK_40M);
s_DATOS_SRAM_IN <= x"00";
wait until rising_edge(s_CLK_40M);
wait for 1 ns;
assert s_DATOS_HACIA_PARITY(32 downto 25) = s_DATOS_SRAM_IN report "Error en el dato leído de la
SRAM" severity error;

```

```

for i in 1 to 3 loop
    wait until rising_edge(s_CLK_40M);
end loop;
assert s_DATOS_SRAM_OUT = x"AA" report "Error en el dato leído de la SRAM" severity error;

for i in 1 to 2 loop
    wait until rising_edge(s_CLK_40M);
end loop;
assert s_DATOS_SRAM_OUT = x"DA" report "Error en el dato leído de la SRAM" severity error;

for i in 1 to 2 loop
    wait until rising_edge(s_CLK_40M);
end loop;
assert s_DATOS_SRAM_OUT = x"DD" report "Error en el dato leído de la SRAM" severity error;

for i in 1 to 2 loop
    wait until rising_edge(s_CLK_40M);
end loop;
assert s_DATOS_SRAM_OUT = x"00" report "Error en el dato leído de la SRAM" severity error;

for i in 1 to 2 loop
    wait until rising_edge(s_CLK_40M);
end loop;

s_INICIA <= '0';

wait;
end process;

end Behavioral;

```

## A.10 RAM\_MU.vhd

```

-----
-- Testbench para RAM_MU v2_10 v4
-- Además de alternar ciclos de lectura/escritura, se agregan condiciones de paridad PAR e IMPAR.
-----
-----
library ieee;
use ieee.std_logic_1164.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use ieee.numeric_std.all;

```

```

entity tb_RAM_MU_v0 is
end tb_RAM_MU_v0;

architecture behavior of tb_RAM_MU_v0 is

    -- Component Declaration for the Unit Under Test (UUT)

    component RAM_MU
        port (
            GW_N : in std_logic_vector(4 downto 1);
            -- REFCL : IN std_logic;
            TMI_N : in std_logic;
            RESM_N : in std_logic;
            GA13A_N : in std_logic;
            GA14A_N : in std_logic;
            DA : in std_logic_vector(4 downto 1);
            PR_N : in std_logic;
            MIH_N : in std_logic;
            GEN : in std_logic;
            CLK_40M : in std_logic;
            GD_N : inout std_logic_vector(24 downto 1);
            GA_N : in std_logic_vector(16 downto 1);
            DATO_SRAM : inout std_logic_vector(8 downto 1);
            READF1A4 : in std_logic_vector(4 downto 1);
            OE_SRAM_N : out std_logic;
            CE_SRAM_N : out std_logic;
            WE_SRAM_N : out std_logic;
            DIR_BUFF_BI : out std_logic;
            OE_READF1 : out std_logic;
            OE_READF2 : out std_logic;
            OE_READF3 : out std_logic;
            OE_READF4 : out std_logic;
            --OE_BUFF_BI_N : OUT std_logic;
            PFB : out std_logic;
            RDY_N : out std_logic;
            PF_N : out std_logic;
            REFS : out std_logic;
            READF : out std_logic;
            --LED2 : OUT std_logic;
            ADDRESS_SRAM : out std_logic_vector(17 downto 1)
        );
    end component;

    --Inputs
    signal GW_N : std_logic_vector(4 downto 1) := (others => 'U');
    -- signal REFCL : std_logic := '0';
    signal TMI_N : std_logic := 'U';
    signal RESM_N : std_logic := 'U';
    signal GA13A_N : std_logic := 'U';

```

```

signal GA14A_N : std_logic           := 'U';
signal DA      : std_logic_vector(4 downto 1) := (others => 'U');
signal PR_N    : std_logic           := 'U';
signal MIH_N   : std_logic           := 'U';
signal GEN     : std_logic           := '0';
signal CLK_40M : std_logic           := '0';
signal GA_N    : std_logic_vector(16 downto 1) := (others => 'U');
signal READF1A4 : std_logic_vector(4 downto 1);

--BiDirs
signal GD_N      : std_logic_vector(24 downto 1) := (others => 'U');
signal DATO_SRAM : std_logic_vector(8 downto 1) := (others => 'U');

--Outputs
signal OE_SRAM_N : std_logic;
signal CE_SRAM_N : std_logic;
signal WE_SRAM_N : std_logic;
signal DIR_BUFF_BI : std_logic;
-- signal OE_BUFF_BI_N : std_logic;
signal OE_READF1 : std_logic;
signal OE_READF2 : std_logic;
signal OE_READF3 : std_logic;
signal OE_READF4 : std_logic;
signal PFB       : std_logic;
signal RDY_N     : std_logic;
signal PF_N      : std_logic;
signal REFS      : std_logic;
signal READF     : std_logic;
--signal LED2 : std_logic;
signal ADDRESS_SRAM : std_logic_vector(17 downto 1);

-- Clock period definitions
constant CLK_40M_period : time := 25 ns; --40MHz
constant DEMORA_NS      : time := 130 ns; --desfaso las seales de reloj
--150 ns -> Flanco ascendente de GEN coincide con el ascendente de CLK
--162.5 ns -> Flanco ascendente de GEN coincide con el descendente de CLK
--140 ns -> GEN adelanta a CLK
--130 ns -> CLK adelanta a GEN
constant GEN_period : time := 125 ns;
constant t1         : time := 85 ns;

constant esc_total      : std_logic_vector(4 downto 1) := not("0000");
constant lect_total     : std_logic_vector(4 downto 1) := not("1111");
constant esc_mitad_superior : std_logic_vector(4 downto 1) := not("0011");
constant esc_mitad_inferior : std_logic_vector(4 downto 1) := not("1100");
constant esc_mitad_word  : std_logic_vector(4 downto 1) := not("1001");
constant esc_char_superior : std_logic_vector(4 downto 1) := not("0111");
constant esc_char_inferior : std_logic_vector(4 downto 1) := not("1110");

```

```

constant dato_sram_parity_fault : std_logic_vector(32 downto 1) := x"01020806";
constant dato_sram_valido      : std_logic_vector(32 downto 1) := x"05020806";

-- 0011 -> --escritura mitad superior
-- 1100 -> --escritura mitad inferior
-- 0111 -> -- escritura caracter superior
-- 1001 -> -- escritura mitad palabra
-- 1110 -> -- escritura bit final

signal SimulacionActiva_IN : std_logic := '0';
signal comienzo_ciclo      : std_logic := '0';
signal s_dato_guardado_sram : std_logic_vector(32 downto 1);

procedure ciclo_lectura (
    signal GEN          : in std_logic;
    signal CLK_40M      : in std_logic;
    signal RDY_N        : in std_logic;
    signal MIH_N        : in std_logic;
    signal CE_SRAM_N    : in std_logic;
    signal DATO_GUARDADO_SRAM : in std_logic_vector(32 downto 1);
    signal TMI_N        : out std_logic;
    signal GA13A_N      : out std_logic;
    signal GA14A_N      : out std_logic;
    signal DA           : out std_logic_vector(4 downto 1);
    signal GA_N         : out std_logic_vector(16 downto 1);
    signal DATO_SRAM    : out std_logic_vector(8 downto 1)
) is
    constant t1 : time := 85 ns;

    variable v_TIME : time := 0 ns; -- Se asigna una variable para determinar el tiempo de ejecucion.

begin

    wait until GEN = '1' and MIH_N = '1';
    wait for t1;

    -----
    --INICIA CICLO DE LECTURA DE PALABRA COMPLETA-----
    -----

    v_TIME := now;
    TMI_N  <= '0';
    GA13A_N <= '0';
    GA14A_N <= '0';
    DA     <= "0011";          --DA1 Y DA2 SON "11"
    GA_N   <= "1100000000000000"; --GA16 Y GA15 SON "11"

    -- Se espera hasta que comience la FSM.
    wait until falling_edge(CE_SRAM_N);

```

```

DATO_SRAM <= DATO_GUARDADO_SRAM(8 downto 1);

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= DATO_GUARDADO_SRAM(16 downto 9);

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= DATO_GUARDADO_SRAM(24 downto 17);

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= DATO_GUARDADO_SRAM(32 downto 25);

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= "ZZZZZZZZ";

-- Tiempo restante para llegar a los 500nS + el desfase de TMI.
wait until rising_edge(GEN);
wait until rising_edge(GEN);

TMI_N <= '1';

-- Se detecta cuando RDY_N sea 1 y se calcula el tiempo de respuesta.
wait until (RDY_N = '1') for 200ns;
v_TIME := now - V_TIME;
report "Duracion del periodo de lectura = " & time'image(v_TIME);

end procedure ciclo_lectura;

procedure ciclo_lectura_fail (
    signal GEN      : in std_logic;
    signal RDY_N    : in std_logic;
    signal MIH_N    : in std_logic;
    signal TMI_N    : out std_logic;
    signal GA13A_N  : out std_logic;
    signal GA14A_N  : out std_logic;
    signal DA       : out std_logic_vector(4 downto 1);
    signal GA_N     : out std_logic_vector(16 downto 1);
    signal DATO_SRAM : out std_logic_vector(8 downto 1)

) is
    constant t1 : time := 85 ns;

```

```

variable v_TIME : time := 0 ns; -- Se asigna una variable para determinar el tiempo de ejecucion.

begin

wait until GEN = '1' and MIH_N = '1';
wait for t1;

-----
--INICIA CICLO DE LECTURA DE PALABRA COMPLETA-----
-----

v_TIME := now;
TMI_N  <= '0';
GA13A_N <= '0';
GA14A_N <= '0';
DA      <= "0011";          --DA1 Y DA2 SON "11"
GA_N    <= "1100000000000000"; --GA16 Y GA15 SON "11"

-- Se espera hasta que comience la FSM.
wait for 22 ns + t1;

wait for 125 ns - (22 ns + t1); --PARA SACAR A TIEMPO LOS DATOS POR LA SRAM
DATO_SRAM <= X"06";

wait for 50 ns; --PARA SACAR A TIEMPO LOS DATOS POR LA SRAM

DATO_SRAM <= X"08";

wait for 50 ns; --PARA SACAR A TIEMPO LOS DATOS POR LA SRAM

DATO_SRAM <= X"02";

wait for 50 ns; --PARA SACAR A TIEMPO LOS DATOS POR LA SRAM

DATO_SRAM <= X"06";

wait for 50 ns; --PARA SACAR A TIEMPO LOS DATOS POR LA SRAM

DATO_SRAM <= "ZZZZZZZZ";

-- Tiempo restante para llegar a los 500nS + el desfase de TMI.
wait for 137.5 ns + t1;

TMI_N <= '1';

-- Se detecta cuando RDY_N sea 1 y se calcula el tiempo de respuesta.
wait until (RDY_N = '1') for 200ns;
v_TIME := now - v_TIME;
report "Duracion del periodo = " & time'image(v_TIME);

```

```

end procedure ciclo_lectura_fail;

procedure ciclo_escritura (
    signal GEN      : in std_logic;
    signal CLK_40M  : in std_logic;
    signal RDY_N    : in std_logic;
    signal MIH_N    : in std_logic;
    signal CE_SRAM_N : in std_logic;
    signal TMI_N    : out std_logic;
    signal GA13A_N  : out std_logic;
    signal GA14A_N  : out std_logic;
    signal DA       : out std_logic_vector(4 downto 1);
    signal GA_N     : out std_logic_vector(16 downto 1);
    signal DATO_SRAM : inout std_logic_vector(8 downto 1)

) is
    constant t1      : time := 10 ns;
    variable v_TIME  : time := 0 ns;

begin
    --! Inicio el ciclo de memoria con un pequeno delay entre el flanco ascendente de GEN
    --! y TMI_N = 0.
    wait until GEN = '1'; --
    wait for t1;
    v_TIME := now;
    TMI_N  <= '0';
    GA13A_N <= '0';
    GA14A_N <= '0';
    DA     <= "0011";          --DA1 Y DA2 SON "11"
    GA_N   <= "1100000000000000"; --GA16 Y GA15 SON "11"

    wait until falling_edge(CE_SRAM_N);

    DATO_SRAM <= x"FF";

    for i in 0 to 10 loop
        wait until rising_edge(CLK_40M);
    end loop;

    DATO_SRAM <= "ZZZZZZZZ";

    --125 ns despues RDY deberia hacerse 0

    wait until rising_edge(GEN);
    wait until rising_edge(GEN);
    TMI_N <= '1';

```

```

-- Se detecta cuando RDY_N sea 1 y se calcula el delta.
wait until RDY_N = '1' for 200ns;
v_TIME := now - V_TIME;
report "Duracion del periodo = " & time'image(v_TIME);

end procedure ciclo_escritura;

begin

-- Instantiate the Unit Under Test (UUT)
 uut : RAM_MU port map
 (
   GW_N => GW_N,
   -- REFCL => REFCL,
   TMI_N => TMI_N,
   RESM_N => RESM_N,
   GA13A_N => GA13A_N,
   GA14A_N => GA14A_N,
   DA => DA,
   PR_N => PR_N,
   MIH_N => MIH_N,
   GEN => GEN,
   CLK_40M => CLK_40M,
   GD_N => GD_N,
   GA_N => GA_N,
   DATO_SRAM => DATO_SRAM,
   READF1A4 => READF1A4,
   OE_SRAM_N => OE_SRAM_N,
   CE_SRAM_N => CE_SRAM_N,
   WE_SRAM_N => WE_SRAM_N,
   DIR_BUFF_BI => DIR_BUFF_BI,
   -- OE_BUFF_BI_N => OE_BUFF_BI_N,
   OE_READF1 => OE_READF1,
   OE_READF2 => OE_READF2,
   OE_READF3 => OE_READF3,
   OE_READF4 => OE_READF4,
   PFB => PFB,
   RDY_N => RDY_N,
   PF_N => PF_N,
   REFS => REFS,
   READF => READF,
   --LED2 => LED2,
   ADDRESS_SRAM => ADDRESS_SRAM
 );

READF1A4 <= READF & READF & READF & READF;

-- Clock process definitions

```

```

CLK_process : process
begin
    CLK_40M <= '0';
    wait for CLK_40M_period/2;
    CLK_40M <= '1';
    wait for CLK_40M_period/2;
end process;

GEN_process : process
begin
    wait for DEMORA_NS;
    while (SimulacionActiva_IN = '1') loop
        GEN <= '0';
        wait for GEN_period / 2;
        GEN <= '1';
        wait for GEN_period / 2;
    end loop;
    GEN <= '0';
    wait; -- Proceso en espera permanente
end process GEN_process;

-----
-- Se crea un proceso para generar estímulos al DUT.
-- Se intercalan ciclos de escritura y lectura de manera alternada.
-----

stim_proc : process
begin

    -- hold reset state for 100 ns.
    RESM_N <= '0'; --estado actual = rest

    wait for 100 ns;
    SimulacionActiva_IN <= '1';
    RESM_N <= '0'; --estado actual = rest
    DATO_SRAM <= "ZZZZZZZZ"; --POR AHORA NO LA USO...
    GA_N <= "1111111111111111"; --GA16 Y GA15 SON "11"

    wait for 100 ns;
    RESM_N <= '1'; --estado actual = rest
    TMI_N <= '1';
    PR_N <= '1';

    wait for 900 ns;
    GD_N <= x"ABCDEDF";
    GW_N <= esc_total;
    ciclo_escritura(GEN, CLK_40M, RDY_N, MIH_N, CE_SRAM_N, TMI_N, GA13A_N, GA14A_N, DA, GA_N, DA-
TO_SRAM);

    -- Ciclo de lectura con fallo de paridad.

```

```

wait for 375 ns;
GW_N          <= lect_total; -- Lectura
GD_N          <= (others => 'Z');
s_dato_guardado_sram <= dato_sram_parity_fault;
ciclo_lectura(GEN, CLK_40M, RDY_N, MIH_N, CE_SRAM_N, s_dato_guardado_sram, TMI_N, GA13A_N, GA14A_N,
DA, GA_N, DATO_SRAM);
wait until rising_edge(PFB);
wait for 125 ns;
RESM_N <= '0'; --estado actual = rest
wait for 375 ns;
RESM_N <= '1'; --estado actual = rest
---

pruebas : for i in 0 to 1 loop
    wait for 375 ns;
    GD_N <= x"134405";
    GW_N <= esc_mitad_inferior;
    ciclo_escritura(GEN, CLK_40M, RDY_N, MIH_N, CE_SRAM_N, TMI_N, GA13A_N, GA14A_N, DA, GA_N, DA-
TO_SRAM);

    wait for 375 ns;
    GW_N <= esc_mitad_superior;
    ciclo_escritura(GEN, CLK_40M, RDY_N, MIH_N, CE_SRAM_N, TMI_N, GA13A_N, GA14A_N, DA, GA_N, DA-
TO_SRAM);

    wait for 375 ns;
    GW_N <= esc_mitad_word;
    ciclo_escritura(GEN, CLK_40M, RDY_N, MIH_N, CE_SRAM_N, TMI_N, GA13A_N, GA14A_N, DA, GA_N, DA-
TO_SRAM);

    wait for 375 ns;
    GW_N <= esc_char_superior;
    ciclo_escritura(GEN, CLK_40M, RDY_N, MIH_N, CE_SRAM_N, TMI_N, GA13A_N, GA14A_N, DA, GA_N, DA-
TO_SRAM);

    wait for 375 ns;
    GW_N <= esc_char_inferior;
    ciclo_escritura(GEN, CLK_40M, RDY_N, MIH_N, CE_SRAM_N, TMI_N, GA13A_N, GA14A_N, DA, GA_N, DA-
TO_SRAM);

    PR_N <= not(PR_N);
end loop; -- pruebas
wait;
end process stim_proc;

-----
-- Se genera un proceso para alternar MIH_N de manera periodica, con un duty
-- cycle variable.

```

```

-----
mih_proc : process
    constant periodo    : time := 10us;
    constant duty_cycle : real := 0.1;

begin

    MIH_N <= '0';
    wait for periodo * duty_cycle;
    MIH_N <= '1';
    wait for periodo;

end process mih_proc;

end;

```

## A.11 tb\_VHDL\_ISE\_pagger.vhd

```

-----
-- Testbench para RAM_MU v11_3.
-- Este testbench pone a prueba el pin de parity fault.
-----

library ieee;
use ieee.std_logic_1164.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use ieee.numeric_std.all;

entity tb_VHDL11_ISE_pagger is
end tb_VHDL11_ISE_pagger;

architecture behavior of tb_VHDL11_ISE_pagger is

    -- Component Declaration for the Unit Under Test (UUT)

    component RAM_MU
        port (
            GW_N      : in std_logic_vector(4 downto 1);
            --REFCL    : in std_logic;
            TMI_N     : in std_logic;
            RESM_N    : in std_logic;
            GA13A_N   : in std_logic;
            GA14A_N   : in std_logic;

```

```

    DA      : in std_logic_vector(4 downto 1);
    PR_N    : in std_logic;
    MIH_N   : in std_logic;
    GEN     : in std_logic;
    CLK_40M : in std_logic;
    GD_N    : inout std_logic_vector(24 downto 1);
    GA_N    : in std_logic_vector(16 downto 1);
    DATO_SRAM : inout std_logic_vector(8 downto 1);
    READF1A4 : in std_logic_vector(4 downto 1);
    MODO_64K : in std_logic;
    OE_SRAM_N : out std_logic;
    CE_SRAM_N : out std_logic;
    WE_SRAM_N : out std_logic;
    DIR_BUFF_BI : out std_logic;
    OE_BUFF_BI_N : out std_logic;
    PFB      : out std_logic;
    RDY_N    : out std_logic;
    PF_N     : out std_logic;
    REFS     : out std_logic;
    READF    : out std_logic;
    --LED2 : OUT std_logic;
    ADDRESS_SRAM : out std_logic_vector(19 downto 1)
);
end component;
--Inputs
signal GW_N      : std_logic_vector(4 downto 1) := (others => 'U');
signal REFCL    : std_logic                    := '0';
signal TMI_N    : std_logic                    := 'U';
signal RESM_N   : std_logic                    := 'U';
signal GA13A_N  : std_logic                    := 'U';
signal GA14A_N  : std_logic                    := 'U';
signal DA      : std_logic_vector(4 downto 1) := (others => 'U');
signal PR_N    : std_logic                    := 'U';
signal MIH_N   : std_logic                    := 'U';
signal GEN     : std_logic                    := '0';
signal CLK_40M : std_logic                    := '0';
signal GA_N    : std_logic_vector(16 downto 1) := (others => 'U');
signal READF1A4 : std_logic_vector(4 downto 1);
signal MODO_64K : std_logic := 'U';

--BiDirs
signal GD_N      : std_logic_vector(24 downto 1) := (others => 'U');
signal DATO_SRAM : std_logic_vector(8 downto 1) := (others => 'U');

--Outputs
signal OE_SRAM_N : std_logic;
signal CE_SRAM_N : std_logic;
signal WE_SRAM_N : std_logic;

```

```

signal DIR_BUFF_BI : std_logic;
signal OE_BUFF_BI_N : std_logic;
signal PFB : std_logic;
signal RDY_N : std_logic;
signal PF_N : std_logic;
signal REFS : std_logic;
signal READF : std_logic;
--signal LED2 : std_logic;
signal ADDRESS_SRAM : std_logic_vector(19 downto 1);

-- Clock period definitions
constant CLK_40M_period : time := 25 ns; --40MHz
constant DEMORA_NS : time := 140 ns; --desfaso las seales de reloj
--150 ns -> Flanco ascendente de GEN coincide con el ascendente de CLK
--162.5 ns -> Flanco ascendente de GEN coincide con el descendente de CLK
--140 ns -> GEN adelanta a CLK
--130 ns -> CLK adelanta a GEN
constant GEN_period : time := 125 ns;
constant t1 : time := 85 ns; -- t1 es el delta entre la bajada de TMI y el rising edge del reloj
de 8MHZ.

constant paridad : std_logic := '1'; -- 0 Impar 1 Par.

constant escritura_parcial : std_logic_vector(4 downto 1) := not("1100"); --escritura mitad superior
-- 0011 -> --escritura mitad superior
-- 1100 -> --escritura mitad inferior
-- 0111 -> -- escritura caracter superior
-- 1001 -> -- escritura mitad palabra
-- 1110 -> -- escritura bit final

signal SimmulacionActiva_IN : std_logic := '0';
signal comienzo_ciclo : std_logic := '0';

begin

-- Instantiate the Unit Under Test (UUT)
ut : RAM_MU port map(
    GW_N => GW_N,
    --REFCL => REFCL,
    TMI_N => TMI_N,
    RESM_N => RESM_N,
    GA13A_N => GA13A_N,
    GA14A_N => GA14A_N,
    DA => DA,
    PR_N => PR_N,
    MIH_N => MIH_N,
    GEN => GEN,
    CLK_40M => CLK_40M,

```

```

GD_N      => GD_N,
GA_N      => GA_N,
DATO_SRAM => DATO_SRAM,
READF1A4 => READF1A4,
MODO_64K => MODO_64K,
OE_SRAM_N => OE_SRAM_N,
CE_SRAM_N => CE_SRAM_N,
WE_SRAM_N => WE_SRAM_N,
DIR_BUFF_BI => DIR_BUFF_BI,
OE_BUFF_BI_N => OE_BUFF_BI_N,
PFB       => PFB,
RDY_N     => RDY_N,
PF_N      => PF_N,
REFS      => REFS,
READF     => READF,
--LED2 => LED2,
ADDRESS_SRAM => ADDRESS_SRAM
);

READF1A4 <= READF & READF & READF & READF;

-- Clock process definitions
CLK_process : process
begin
    CLK_40M <= '0';
    wait for CLK_40M_period/2;
    CLK_40M <= '1';
    wait for CLK_40M_period/2;
end process;

GEN_process : process
begin
    wait for DEMORA_NS;
    while (SimulacionActiva_IN = '1') loop
        GEN <= '0';
        wait for GEN_period / 2;
        GEN <= '1';
        wait for GEN_period / 2;
    end loop;
    GEN <= '0';
    wait; -- Proceso en espera permanente
end process GEN_process;

-- Stimulus process
stim_proc : process
begin

    -- hold reset state for 100 ns.
    wait for 100 ns;

```

```

SimulacionActiva_IN <= '1';
RESM_N <= '0'; --estado actual = rest
MIH_N <= '0'; --cuando RESM_N es '0' MIH_N tambien es '0'
DATO_SRAM <= "ZZZZZZZ"; --POR AHORA NO LA USO...
GA_N <= "1111111111111111"; --GA16 Y GA15 SON "11"
GD_N <= (others => 'Z');

wait for 250 ns;

RESM_N <= '1'; --0ns
TMI_N <= '1';
MIH_N <= '1';
PR_N <= paridad; --paridad PAR. Le envio este valor al bloque de paridad para que genere paridad.

GW_N <= "0000";
wait for 100 ns; --

--! Se pone a prueba el filtro de glitches.
MODO_64K <= '0';
wait for 500 ns;
MODO_64K <= '1';
wait for 500 ns;

wait until falling_edge(GEN);
MIH_N <= '0';

-- Secuencia de la captura VHDL_11_2
-----
--INICIA CICLO DE LECTURA DE PALABRA COMPLETA-----
-----

wait for 1460 ns; -- Se prueba con un desfase entre -10 y -85 ns.
MIH_N <= '1';
TMI_N <= '0';
GA_N <= x"1FFF"; --GA16 Y GA15 SON "11"

--PONGO EL BUS G EN Z para evitar
GD_N <= (others => 'Z');

GW_N <= "0000"; --MODO LECTURA

wait for 30 ns;

TMI_N <= '1';

wait for 30 ns;

TMI_N <= '0';

```

```

-- Se espera a que la maquina de estados se encuentre en el momento de lectura.
wait for 65 ns;
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
DATO_SRAM <= X"EF";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"CD";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"AB";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"04";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= "ZZZZZZZZ";

wait for 137.5 ns; --separo escritura de lectura

-- MIH_N NO VARIA, queda como antes
-- PR_N NO VARIA, queda como antes
-- RESM_N NO VARIA, queda como antes

TMI_N <= '1';
--! Se espera hasta que RDY_N indice el fin del ciclo de memoria.
--! En ese momento se deshabilita la memoria.
wait until RDY_N = '1';
wait for 70 ns;
MIH_N <= '0';

wait for 100 ns;

-----
--INICIA CICLO DE ESCRITURA DE PALABRA COMPLETA-----
-----

wait until rising_edge(GEN);
wait for t1; -- Se prueba con un desfase entre 10 y 85 ns.
MIH_N <= '1';

```

```

TMI_N <= '0';
GA13A_N <= '0';
GA14A_N <= '0';
DA <= "0011";           --DA1 Y DA2 SON "11"
GA_N <= "1111111111111100"; --GA16 Y GA15 SON "11"
GW_N <= "1111";         --ESCRITURA COMPLETA

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= "01010101";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(GEN);

GD_N <= x"5A5A5A"; -- Poner en hex.

wait until RDY_N = '0' for 250 ns;

DATO_SRAM <= "ZZZZZZZZ";

--125 ns despues RDY deberia hacerse 0

wait for 147.5 ns;

TMI_N <= '1';

wait for 100 ns;

wait for 500 ns;

-- Secuencia de la captura original
-----
--INICIA CICLO DE ESCRITURA DE PALABRA COMPLETA-----
-----
-- Se prueba si cae TMI antes que un rising edge de GEN.
wait until falling_edge(GEN);
wait for 40 ns; -- Se prueba con un desfase entre -10 y -85 ns.

```

```

TMI_N  <= '0';
GA13A_N <= '0';
GA14A_N <= '0';
DA     <= "0011";           --DA1 Y DA2 SON "11"
GA_N   <= "11111111111111100"; --GA16 Y GA15 SON "11"
GW_N   <= "1111";         --ESCRITURA COMPLETA

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= "01010101";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

GD_N <= x"5A5A5A"; -- Poner en hex.

wait until RDY_N = '0' for 250 ns;

DATO_SRAM <= "ZZZZZZZ";

--125 ns despues RDY deberia hacerse 0

wait for 147.5 ns;

TMI_N <= '1';

wait for 100 ns;

-----
--INICIA CICLO DE ESCRITURA DE PALABRA COMPLETA-----
-----
--! Se prueba una condicion normal.
wait until rising_edge(GEN);
wait for t1; -- Se prueba con un desfase entre 10 y 85 ns.

TMI_N  <= '0';
GA13A_N <= '0';
GA14A_N <= '0';
DA     <= "0011";           --DA1 Y DA2 SON "11"

```

```

GA_N <= "1111111111111100"; --GA16 Y GA15 SON "11"
GW_N <= "1111"; --ESCRITURA COMPLETA

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= "01010101";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

GD_N <= x"A5A5A5"; -- Poner en hex.

wait until RDY_N = '0' for 250 ns;

DATO_SRAM <= "ZZZZZZZZ";

--125 ns despues RDY deberia hacerse 0

wait for 147.5 ns;

TMI_N <= '1';

--! Se espera hasta que RDY_N indice el fin del ciclo de memoria.
--! En ese momento se deshabilita la memoria.
wait until RDY_N = '1';
MIH_N <= '0';

-----
--INICIA CICLO DE LECTURA DE PALABRA COMPLETA-----
-----

wait until rising_edge(GEN);
TMI_N <= '0';
-- Se esperan 108 segundos como en la captura para volver a habilitar la memoria.
wait for 108 ns;
MIH_N <= '1';

GA_N <= "11111111110101111"; --GA16 Y GA15 SON "11"

--PONGO EL BUS G EN Z para evitar
GD_N <= (others => 'Z');

```

```

GW_N <= "0000"; --MODULO LECTURA

-- Se espera a que la maquina de estados se encuentre en el momento de lectura.
wait until rising_edge(GEN);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
DATO_SRAM <= X"EF";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"CD";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"AB";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"04";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= "ZZZZZZZZ";

wait for 137.5 ns; --separo escritura de lectura

--   MIH_N NO VARIA, queda como antes
--   PR_N NO VARIA, queda como antes
--   RESM_N NO VARIA, queda como antes

TMI_N <= '1';

--! Se espera hasta que RDY_N indice el fin del ciclo de memoria.
--! En ese momento se deshabilita la memoria.
wait until RDY_N = '1';
MIH_N <= '0';

-----
--INICIA CICLO DE LECTURA DE PALABRA COMPLETA-----
-----

wait for 108 ns; -- Se prueba con un desfase entre -10 y -85 ns.

```

```

MIH_N <= '1';
TMI_N <= '0';
GA_N  <= "1111111110101111"; --GA16 Y GA15 SON "11"

--PONGO EL BUS G EN Z para evitar
GD_N <= (others => 'Z');

GW_N <= "0000"; --MODO LECTURA

-- Se espera a que la maquina de estados se encuentre en el momento de lectura.
wait until rising_edge(GEN);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
DATO_SRAM <= X"EF";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"CD";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"AB";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"04";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= "ZZZZZZZZ";

wait for 137.5 ns; --separo escritura de lectura

-- MIH_N NO VARIA, queda como antes
-- PR_N NO VARIA, queda como antes
-- RESM_N NO VARIA, queda como antes

TMI_N <= '1';
--! Se espera hasta que RDY_N indice el fin del ciclo de memoria.
--! En ese momento se deshabilita la memoria.
wait until RDY_N = '1';
wait for 100 ns;
MIH_N <= '0';

```

```

-----
--INICIA CICLO DE LECTURA DE PALABRA COMPLETA-----
-----

wait for 400 ns;
MIH_N <= '1';
TMI_N <= '0';
GA_N <= "111111110101111"; --GA16 Y GA15 SON "11"

--PONGO EL BUS G EN Z para evitar
GD_N <= (others => 'Z');

GW_N <= "0000"; --MODO LECTURA

-- Se espera a que la maquina de estados se encuentre en el momento de lectura.
wait until rising_edge(GEN);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);
DATO_SRAM <= X"EF";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"CD";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"AB";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= X"03";

wait until rising_edge(CLK_40M);
wait until rising_edge(CLK_40M);

DATO_SRAM <= "ZZZZZZZ";

wait for 137.5 ns; --separo escritura de lectura

-- MIH_N NO VARIA, queda como antes
-- PR_N NO VARIA, queda como antes
-- RESM_N NO VARIA, queda como antes

TMI_N <= '1';

```

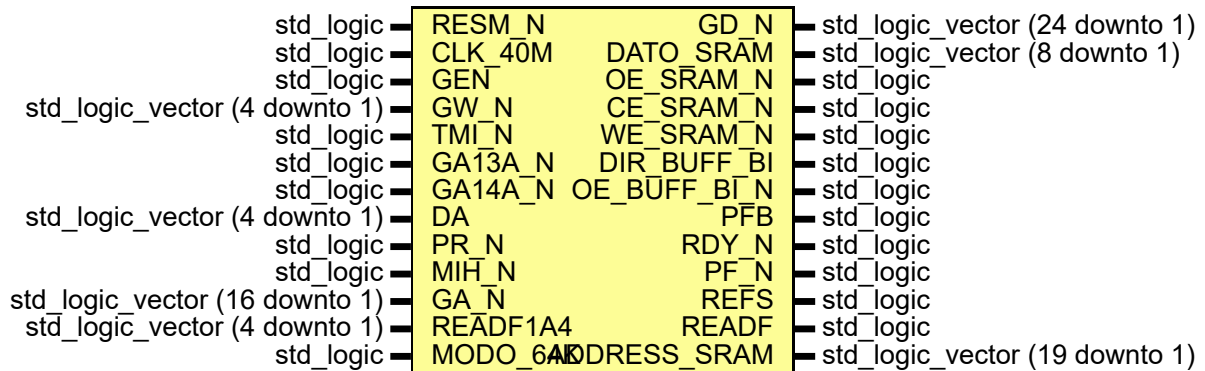
```
wait for 100 ns;  
  
wait;  
  
end process;  
  
end;
```

## Anexo B - Diseño Digital - RAM-MU64

### Entity: RAM\_MU

- File: RAM\_MU.vhd

### Diagram



### Description

Este conjunto es el bloque principal del proyecto. Es la estructura que engloba todos los módulos creados para replicar el comportamiento de la memoria original. Se encarga, además, de implementar el temporizado y habilitaciones de los puertos E/S de: La FPGA, la SRAM y los buffers de traslación de nivel. Recibe, a través de estos últimos la E/S del bus del sistema, y las asocia con los respectivos bloques. Además, dentro de este bloque, se introdujo la modificación de controlar el sentido de transmisión de datos de los buffers bidireccionales en función del estado de las señales READF1-4, respetando la forma de control de la memoria original, favoreciendo la adquisición de datos en los ciclos de escritura parciales.

### Ports

Port name	Direction	Type	Description
RESM_N	in	std_logic	Reset global.
CLK_40M	in	std_logic	Reloj de 40MHz.
GEN	in	std_logic	Reloj del GBUS a 8MHz.
GW_N	in	std_logic_vector (4 downto 1)	Señales que indican el tipo de ciclo de memoria (lectura, escritura o escritura parcial).
TMI_N	in	std_logic	Señal del GBUS que indica inicio de un ciclo normal de memoria.
GA13A_N	in	std_logic	Señal del GBUS que se utiliza para discriminar los módulos de memoria RAM presentes en el GBUS.
GA14A_N	in	std_logic	Señal del GBUS que se utiliza para discriminar los módulos de memoria RAM presentes en el GBUS.
DA	in	std_logic_vector (4 downto 1)	Identificador de la dirección del dispositivo (Device Adress).
PR_N	in	std_logic	Entrada proveniente del GBUS e indica si la paridad utilizada es par ('1' lógico) o impar ('0' lógico).
MIH_N	in	std_logic	MIH_N: (Memory inhibit): Se usa para inhibir POR '0' la operación normal de memoria y causa una acción de refresco.
GA_N	in	std_logic_vector (16 downto 1)	Dirección de la palabra de 24 bits que se quiere escribir/leer con el formato original de 16 bits del GBUS.
READF1A4	in	std_logic_vector (4 downto 1)	Señales del GBUS que indican cuando el procesador leerá un dato de los buffers de la memoria.
MODO_64K	in	std_logic	Entrada proveniente del hardware que determina si el funcionamiento es como módulo de 16k estado lógico '0' o 64k estado lógico '1'.
GD_N	inout	std_logic_vector (24 downto 1)	Dato entrante o saliente perteneciente del GBUS, perteneciente a la dirección GA_N.
DATO_SRAM	inout	std_logic_vector (8 downto 1)	Datos de cada dirección de memoria SRAM.

Port name	Direction	Type	Description
OE_SRAM_N	out	std_logic	Señal para manejar el pin output enable de la memoria SRAM.
CE_SRAM_N	out	std_logic	Señal para manejar el pin chip enable de la memoria SRAM.
WE_SRAM_N	out	std_logic	Bits para determinar el tipo de acción de memoria (escritura, lectura o escritura parcial).
DIR_BUFF_BI	out	std_logic	Control de la dirección de los buffers bidireccionales que adaptan los daots GD_N. Por defecto se encuentra en modo receptivo, es decir, desde el BUS G hacia la FPGA y ante un ciclo de lectura de la memoria se habilitan para adaptar desde la FPGA hacia el BUS G.
OE_BUFF_BI_N	out	std_logic	Control de las salidas (Output Enable) de los buffers bidireccionales que adaptan los daots GD_N. Se presenta alta impedancia 'Z' en los pines de los buffers con un '1' lógico y se permite la adaptación con un '0' lógico.
PFB	out	std_logic	Señal que le indica maneja el LED de fallo de paridad.
RDY_N	out	std_logic	Señal del handshake entre el GBUS y la memoria. A través de RDY_N se informa al GBUS si la memoria se encuentra ejecutando un ciclo de memoria.
PF_N	out	std_logic	Señal que le indica falla de paridad al GBUS.
REFS	out	std_logic	Señal que indica al procesador si la memoria se está refrescando ('1' lógico) o no ('0' lógico).
READF	out	std_logic	Esta señal informa al CPIO que los datos en los buffers de salida están listos a presentarse en el BUS-G.
ADDRESS_SRAM	out	std_logic_vector (19 downto 1)	Dirección de la memoria SRAM.

## Signals

Name	Type	Description
S_MIH_N_REGISTRADA	std_logic	Señal interna del pin MIH_N libre de ruidos.
S_RESET_INTERNO	std_logic	Señal interna del pin RESM_N libre de ruidos.
S_TMI_N_REGISTRADA	std_logic	Señal interna del pin TMI_N libre de ruidos.
S_GA13A_N_REGISTRADA	std_logic	Señal interna del pin GA13A_N libre de ruidos.
S_GA14A_N_REGISTRADA	std_logic	Señal interna del pin GA14A_N libre de ruidos.
S_MODAL64K_REGISTRADA	std_logic	Señal interna del pin MODO_64K libre de ruidos.
S_DA_REGISTRADA	std_logic_vector (4 downto 1)	Señal interna de los pines DA libre de ruidos.
S_PR_N_REGISTRADA	std_logic	Señal interna de los pines DA libre de ruidos.
S_GA_N_REGISTRADA	std_logic_vector (16 downto 1)	Señal interna de los pines DA libre de ruidos.
S_WE4A1_N_REGISTRADA	std_logic_vector (4 downto 1)	Señal interna de los pines DA libre de ruidos.
S_PULSO_GEN_25NS	std_logic	Señal interna del reloj del GBUS sincronizado con la FPGA.
S_GA16_A_GA15_N	std_logic_vector (1 downto 0)	Señal interna que corresponde a los bits 16 y 15 de la dirección del dato GA.
S_GA14A_A_GA13A_N	std_logic_vector (1 downto 0)	Señal interna que corresponde a los bits 14 y 13 de la dirección del dato GA.
S_EQ	std_logic	Señal interna EQ que indica cuando una petición de lectura/escritura debe ser aceptada.
S_GBUS_IN	std_logic_vector (24 downto 1)	Señal interna que se utiliza para enviar un dato desde el GBUS hacia la memoria.
S_SRAM_OUT	std_logic_vector (32 downto 1)	Señal interna que representa un dato del GBUS con el formato de 32 bits utilizado internamente en este diseño.
S_GBUS_OUT	std_logic_vector (24 downto 1)	Señal interna que se utiliza para enviar un dato desde la memoria hacia el GBUS.
S_PF_N	std_logic	Señal interna para conectar con la salida PF_N e indicar cuando hay falla de paridad.
S_LED1	std_logic	Señal interna para conectar con la salida PFB e indicar cuando hay falla de paridad.
S_PULSO_READF_25NS	std_logic	Señal interna para detectar un flanco descendente en READF.

Name	Type	Description
S_READF	std_logic	Señal interna para conectar con la salida READF e indicar cuando hay falla de paridad.
S_DATO_VALIDO_EN_GBUS	std_logic	Señal interna que indica cuando el dato disponible en el GBUS está establecido de manera correcta.
S_DETECT_READF	std_logic	Señal interna para detectar un flanco descendente de READF.
S_INICIA_MEMO	std_logic	Señal interna que indica el comienzo de un ciclo de memoria.
S_GUARDA_WE4A1	std_logic	Señal hacia registrar_entradas que indica cuando se debe mantener los valores guardados de WE4A1.
S_RDY_N	std_logic	Señal interna para conectar con la salida RDY_N e indicar cuando hay falla de paridad.
S_DATOS_OUT_RAM_MU	std_logic_vector (32 downto 1)	Datos de cada dirección de memoria SRAM.
S_CE_SRAM_N	std_logic	Señal auxiliar del pin CE_SRAM_N.
S_OE_SRAM_N	std_logic	Señal auxiliar del pin OE_SRAM_N.
S_WE_SRAM_N	std_logic	Señal auxiliar del pin WE_SRAM_N.
S_OE_FPGA_A_SRAM	std_logic	Señal auxiliar para indicar cuando se va a escribir hacia la SRAM y configurar DATO_SRAM como salida.
S_ADDRESS_SRAM	std_logic_vector (19 downto 1)	Señal interna para conectar con la salida ADDRESS_SRAM.
S_DATO_SRAM_IN	std_logic_vector (8 downto 1)	Señal interna para conectar a DATO_SRAM en sentido SRAM a FPGA.
S_DATO_SRAM_OUT	std_logic_vector (8 downto 1)	Señal interna para conectar a DATO_SRAM en sentido FPGA a SRAM.
S_DIR_BUFF_BI	std_logic	
S_DIR_FPGA_IO	std_logic	

## Processes

- Detector\_de\_flanco\_READF: ( CLK\_40M, S\_READF )
  - Description**  
Proceso secuencial que registra la entrada READF.  
La salida de este registro se la utiliza para generar una señal interna que indica cuando hubo un flanco descendente en este pin.
- BUFFERS\_BUS\_G\_EN\_ESCRITURA: ( S\_MIH\_N\_REGISTRADA, CLK\_40M, S\_READF, READF1A4 )
  - Description**  
EL PROBLEMA A RESOLVER ES QUE NUNCA PUEDEN ESTAR EN SALIDA LOS PINES DE LA FPGA AL MISMO TIEMPO QUE LOS BUFFERS DEL BUS G ESTAN EN ENTRADA, SE QUEMA. LOS BUFERS SIEMPRE ESTAN EN LECTURA DEL BUS G Y ESCRITURA DE LA FPGA, Y LA FPGA ESTA EN ENTRADA.  
PRIMERO DEBEMOS COLOCAR LOS BUFERS EN ESCRITURA DEL BUS G Y LECTURA DE LA FPGA PARA LUEGO UN TIEMPO DESPUES PONER LA FPGA EN ESCRITURA

## Instantiations

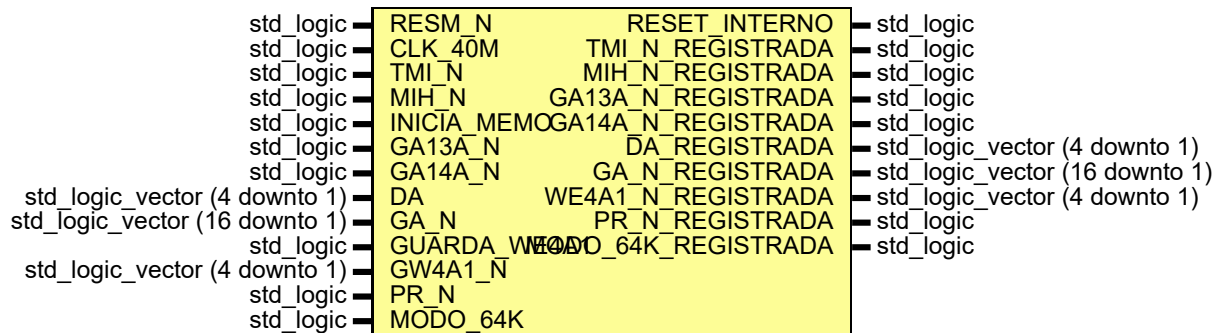
- bloque\_registrar\_entradas\_v1: registrar\_entradas\_v1
  - Este bloque permite eliminar problemas de meta-estabilidad que se dan cuando se manejan dos señales de reloj con flancos distintos y variables, favoreciendo a conseguir un mayor grado de confiabilidad sobre el funcionamiento de la memoria. Además, sirve para filtrar ruido eléctrico presentes en las señales que llegan hasta la placa de circuito impreso.
- bloque\_GEN: GEN\_v0
  - Este módulo se encarga de sincronizar el reloj de 8MHz del GBUS con el reloj interno de la FPGA de 40MHz.
- bloque\_MAD: MAD\_v0
  - El bloque MAD es el encargado, al igual que en el diseño original, de identificar a que módulo en particular son dirigidas las acciones ejecutadas por el CPU. Para ello, se utilizan distintas entradas del GBUS: GA13A, GA14A, GA15\_N, GA16\_N y DA. La señal EQ se utiliza en el resto del diseño para activar o no los procedimientos de lectura o escritura. La característica nueva que agrega esta versión es la posibilidad de responder ante todas las peticiones de un conjunto de cuatro módulos, habilitando así que se pueda elevar la capacidad de un sólo módulo de 16K a 64K. Dado que este diseño debe utilizarse en ciertas ocasiones en 16k de manera obligatoria, se agrega una entrada extra llamada MODO\_64K la cual indica como debe comportarse el MAD.
- bloque\_TAC: TAC

- El circuito de sincronización y control (TAC) procesa las señales necesarias para secuenciar los distintos ciclos de la memoria que se envían como señal de handshake a la unidad que está utilizando la memoria. En este bloque se detecta cuando se inicia un ciclo de memoria (INICIA\_MEMO = '1'). Además, se implementa una lógica que permite inferir el momento que está el valor correcto de GW y, así, indicar la acción a realizar con la memoria. También, se divide, para un ciclo de escritura, el momento donde los datos están establecidos en el bus (DATO\_VALIDO\_EN\_GBUS), para poder enviar al bloque correspondiente la señal de habilitación de los flip-flops que toman y retienen dichos datos. Por último, se implementa una lógica para evitar que DATO\_VALIDO\_EN\_GBUS = '1' si nos encontramos en un ciclo de lectura. Finalmente, se generan las señales de aviso RDY y READF con el temporizado específico del handshake original, según corresponda.
- bloque\_PARIDAD: PARIDAD\_v0
  - El componente Paridad\_v0 se encarga de generar los bits de paridad correspondientes para la escritura de datos, de comprobar si existen fallos de paridad en la lectura y de generar la señal PF (Parity Fault) del G-BUS, al igual que controlar el LED indicador de fallo de paridad PF\_LED. Para esto hace uso de dos bloques más PFF\_v0 y GEN\_PAR\_v0.
- bloque\_Maneja\_SRAM: Prueba\_FSM\_y\_MANEJASRAM
  - Este módulo se encarga de administrar los datos que se almacenan en la memoria y, por ende, del manejo del integrado de memoria SRAM. Está compuesto por dos componentes: una máquina de 18 estados y una implementación de registros de datos que contiene toda la lógica necesaria para para realizar la descomposición de la palabra de 32 bits (24 bits de dato + 4 bits de paridad
- 4 bits en '0'). Este diseño no agrega lógica adicional, solo interconecta las entradas y salidas pertinentes a los componentes internos y externos.

## Entity: registrar\_entradas\_v1

- File: registrar\_entradas\_v1.vhd

## Diagram



## Description

Este bloque permite eliminar problemas de meta-estabilidad que se dan cuando se manejan dos señales de reloj con flancos distintos y variables, favoreciendo a conseguir un mayor grado de confiabilidad sobre el funcionamiento de la memoria. Además, sirve para filtrar ruido eléctrico presentes en las señales que llegan hasta la placa de circuito impreso.

A este componente se conectan las entradas crudas provenientes desde los pines de la memoria y luego se distribuyen al resto del diseño las señales internas libres de glitches. Las entradas MODO\_64K y MIH\_N se filtran a través del componente Filtro\_Glitches\_v2. En el resto de las señales se utilizan soluciones a medida, ya que se deben agregar otras condiciones como la retención de datos ante un ciclo de memoria.

## Ports

Port name	Direction	Type	Description
RESM_N	in	std_logic	Reset global.
CLK_40M	in	std_logic	Reloj de 40MHz.
TMI_N	in	std_logic	Señal del GBUS que indica inicio de un ciclo normal de memoria.
MIH_N	in	std_logic	Inhibición de memoria. Esta señal se utiliza para inhibir las acciones normales de memoria.
INICIA_MEMO	in	std_logic	Señal que indica cuando la memoria debe iniciar un ciclo.
GA13A_N	in	std_logic	Señal del GBUS que se utiliza para discriminar los módulos de memoria RAM presentes en el GBUS.

Port name	Direction	Type	Description
GA14A_N	in	std_logic	Señal del GBUS que se utiliza para discriminar los módulos de memoria RAM presentes en el GBUS.
DA	in	std_logic_vector (4 downto 1)	Conjunto de bits indican la dirección del dispositivo (Device Adress) presente en el GBUS.
GA_N	in	std_logic_vector (16 downto 1)	Señales que indican la dirección en memoria de un dato.
GUARDA_WE4A1	in	std_logic	Señal que indica cuando se debe retener el valor presente en GW_4A1_N.
GW4A1_N	in	std_logic_vector (4 downto 1)	Señales que indican el tipo de ciclo de memoria (lectura, escritura o escritura parcial).
PR_N	in	std_logic	Invertir paridad. Esta señal indica si se debe usar paridad par o impar en la comprobación de los módulos.
MODO_64K	in	std_logic	Señal que indica si la memoria debe comportarse en 16K24 o 64k24 bits.
RESET_INTERNO	out	std_logic	Señal reset libre de glitches que se utiliza en el resto del diseño.
TMI_N_REGISTRADA	out	std_logic	Señal TMI_N libre de glitches que se utiliza en el resto del diseño.
MIH_N_REGISTRADA	out	std_logic	Señal MIH_N libre de glitches que se utiliza en el resto del diseño.
GA13A_N_REGISTRADA	out	std_logic	Señal GA13_A_N libre de glitches que se utiliza en el resto del diseño.
GA14A_N_REGISTRADA	out	std_logic	Señal GA14_A_N libre de glitches que se utiliza en el resto del diseño.
DA_REGISTRADA	out	std_logic_vector (4 downto 1)	Señales DA libres de glitches que se utiliza en el resto del diseño.
GA_N_REGISTRADA	out	std_logic_vector (16 downto 1)	Señales GA_N libres de glitches que se utiliza en el resto del diseño.
WE4A1_N_REGISTRADA	out	std_logic_vector (4 downto 1)	Señales WE4A1_N libres de glitches que se utiliza en el resto del diseño.
PR_N_REGISTRADA	out	std_logic	Señal PR_N libre de glitches que se utiliza en el resto del diseño.
MODO_64K_REGISTRADA	out	std_logic	Señal MODO_64_K libre de glitches que se utiliza en el resto del diseño.

## Signals

Name	Type	Description
s_resm_anterior	std_logic	
s_resm_anterior2	std_logic	
s_resm_anterior3	std_logic	
s_resm_anterior4	std_logic	
s_resm_anterior5	std_logic	
S_RESET_INTERNO	std_logic	
s_TMI_N_ant1	std_logic	
s_TMI_N_ant2	std_logic	
s_TMI_N_ant3	std_logic	
s_MIH_N_registrada	std_logic	
s_GA13A_N_ant1	std_logic	
s_GA13A_N_ant2	std_logic	
s_GA14A_N_ant1	std_logic	
s_GA14A_N_ant2	std_logic	
s_DA_ant1	std_logic_vector(4 downto 1)	
s_DA_ant2	std_logic_vector(4 downto 1)	
s_GA_N_ant1	std_logic_vector(16 downto 1)	
s_GA_N_ant2	std_logic_vector(16 downto 1)	
S_GW4A1_N_ANT1	std_logic_vector(4 downto 1)	
S_GW4A1_N_ANT2	std_logic_vector(4 downto 1)	
s_PR_N_anterior	std_logic	
s_PR_N_anterior2	std_logic	

Name	Type	Description
s_PR_N_REGISTRADA	std_logic	
s_MODO_64K_REGISTRADA	std_logic	

## Processes

- proc\_elimina\_glitch\_RESM\_N: ( CLK\_40M, RESM\_N, s\_resm\_anterior )
  - **Description**  
Se filtran los posibles glitches en el pin RESM\_N con un filtro de 5 registros y se los compara con una compuerta OR. De esta manera hace falta que la señal RESM\_N se mantenga en '0' durante 125nS para que haya un reset.
- registra\_TMI\_N: ( CLK\_40M )
  - **Description**  
Se filtran los posibles glitches en el pin TMI\_N con un filtro de 3 registros y se los compara con una compuerta OR. Teniendo esto en cuenta, hace falta que la señal RESM\_N se mantenga en '0' durante 75nS para que TMI\_N cambie de '1' a '0'.
- registra\_DIR: ( CLK\_40M, INICIA\_MEMO, GA13A\_N, s\_GA13A\_N\_ant1, GA14A\_N, s\_GA14A\_N\_ant1, DA, s\_DA\_ant1, GA\_N, s\_GA\_N\_ant1 )
  - **Description**  
Se registran los pines GA13A\_N, GA14A\_N, DA y GA\_N con 3 flip flops en cascada y la salida del último registro se la asigna a la señal x\_REGISTRADA correspondiente. Además, estos registros poseen un pin de habilitación (enable) gobernado por la entrada INICIA\_MEMO, la cual indica cuando se tienen que retener los datos que estan presentes en el GBUS.
- registra\_ACCION\_MEMO: ( CLK\_40M, GUARDA\_WE4A1, GW4A1\_N, S\_GW4A1\_N\_ANT1 )
  - **Description**  
Se registran las entradas GW4A1\_N con 2 flip flops en cascada y la salida negada del último registro se la asigna a la señal WE4A1\_N\_REGISTRADA Además, estos registros poseen un pin de habilitación (enable) gobernado por la entrada GUARDA\_WE4A1, la cual indica el momento en el que se deben retener los valores.
- registrar: ( CLK\_40M, PR\_N, s\_PR\_N\_anterior )
  - **Description**  
Se registra la entrada PR\_N con 2 flip flops en cascada y la salida del último registro se la asigna a la señal PR\_N\_REGISTRADA.

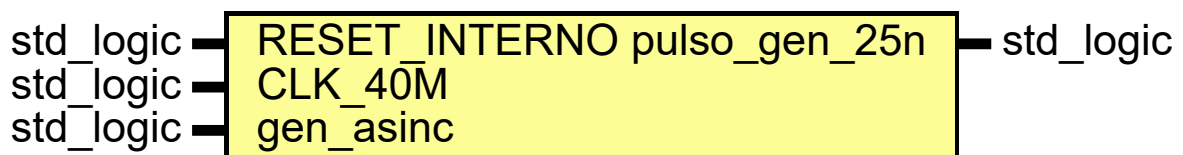
## Instantiations

- Filtro\_Glitches\_MODO\_64K: Filtro\_Glitches\_v2
  - Filtro de glitches simetrico para MODO\_64K de 10 registros.
- Filtro\_Glitches\_MIH\_N: Filtro\_Glitches\_v2
  - Filtro de glitches simetrico para el MIH\_N de 3 registros.

## Entity: GEN\_v0

- **File:** GEN\_v0.vhd

## Diagram



## Description

Este módulo se encarga de sincronizar el reloj de 8MHz del GBUS con el reloj interno de la FPGA de 40MHz.

Para ello se utilizan dos registros en cascada a la entrada de reloj asincrónica y se toman las salidas de estos registros para detectar un flanco ascendente. La salida del reloj sincronizado toma el valor '0' por defecto y '1' ante cada flanco ascendente. Se utilizan las salidas pertenecientes a los

registros para evitar metastabilidad. Aunque ahorra lógica, no se puede utilizar señales que no sean la de salida del circuito corrector de metaestabilidad formado por dos FF.

## Ports

Port name	Direction	Type	Description
RESET_INTERNO	in	std_logic	Reset de los registros.
CLK_40M	in	std_logic	Reloj de 40 MHz.
gen_asinc	in	std_logic	Reloj de 8 MHz desincronizado.
pulso_gen_25n	out	std_logic	Reloj de 8 MHz sincronizado.

## Signals

Name	Type	Description
s_gen_sincro	std_logic	Primer registro de gen_asinc (t-1).
s_gen_sincro_1	std_logic	Segundo registro de gen_asinc (t-2).

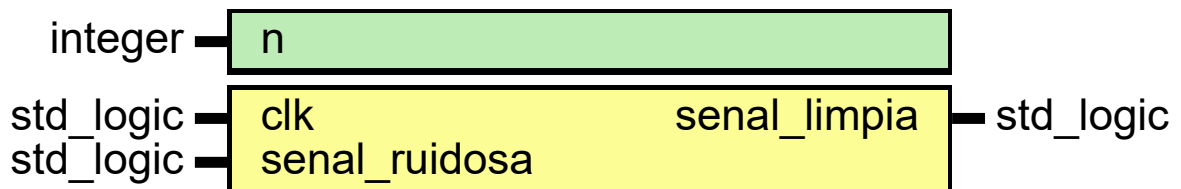
## Processes

- REGISTRO\_GEN: ( RESET\_INTERNO, CLK\_40M )
  - **Description**  
Se describen dos flip flops en cascada para utilizarlos en la sincronización de las señales.

## Entity: Filtro\_Glitches\_v2

- **File:** Filtro\_Glitches\_v2.vhd

## Diagram



## Description

Filtro de glitches, basado en la implementación de Cypress <https://www.cypress.com/file/130886/download>. La señal limpia se crea de tal manera que un cambio es hecho sólo si se mantuvo el valor de entrada durante  $n$  ciclos. Por ejemplo, si  $n=3$  la señal ruidosa de entrada tiene que mantener ese valor durante 3 ciclos para que la señal filtrada tome el mismo valor. En esta segunda versión se realiza un cambio en la salida cuando se detectan  $n$  '0' o '1' consecutivos, de lo contrario se mantiene el valor de salida.

## Generics

Generic name	Type	Value	Description
n	integer	3	Numero ciclos reloj filtrados.

## Ports

Port name	Direction	Type	Description
clk	in	std_logic	Reloj.
senal_ruidosa	in	std_logic	Maximo ruido filtrado es de $n \cdot \text{periodo\_reloj}$ segundos.
senal_limpiar	out	std_logic	Señal filtrada.

## Signals

Name	Type	Description
registros	std_logic_vector(n - 1 downto 1)	Vector con los $n$ valores anteriores.

Name	Type	Description
salida_actual	std_logic	Señal que se utiliza para registrar el valor de senal_limpia.
salida_proxima	std_logic	Señal que se utiliza para asignarle un valor a senal_limpia.

## Constants

Name	Type	Value	Description
unos	std_logic_vector(n - 1 downto 1)	(others => '1')	Vector de la misma longitud que registros. Solo contiene '1'.
ceros	std_logic_vector(n - 1 downto 1)	(others => '0')	Vector de la misma longitud que registros. Solo contiene '0'.

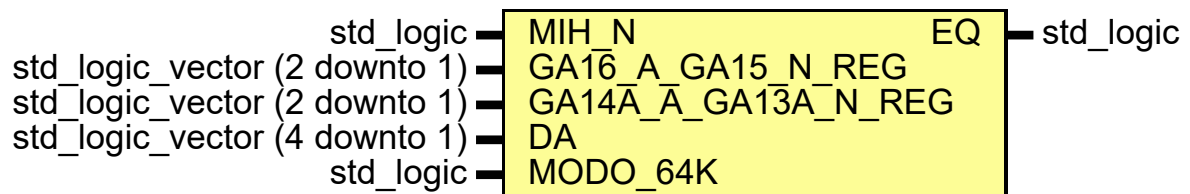
## Processes

- modificar\_salida: ( clk )
  - **Description**  
Registro para la salida del filtro. Se le asigna a salida\_actual el valor de salida\_proxima. De esta manera es posible saber el valor que actualmente posee senal\_limpia.
- registros\_entradas: ( clk )
  - **Description**  
Se genera la cascada de n-1 registros que serán utilizados para filtrar la señal.
- comparacion: ( senal\_ruidosa, registros, salida\_actual )
  - **Description**  
Se crea la logica para determinar la salida en funcion de las entradas anteriores guardadas en registros. Para ello se modifica la señal salida\_proxima. Si la entrada actual es '1' y los valores anteriores tambien lo fueron salida\_proxima es '1'. Si la entrada actual es '0' y los valores anteriores tambien lo fueron salida\_proxima es '0'. En caso contrario el valor de salida\_proxima es salida\_actual.

## Entity: MAD\_v0

- File: MAD\_v0.vhd

## Diagram



## Description

El bloque MAD es el encargado, al igual que en el diseño original, de identificar a que módulo en particular son dirigidas las acciones ejecutadas por el CPU. Para ello, se utilizan distintas entradas del GBUS: GA13A, GA14A, GA15\_N, GA16\_N y DA. La señal EQ se utiliza en el resto del diseño para activar o no los procedimientos de lectura o escritura. La característica nueva que agrega esta versión es la posibilidad de responder ante todas las peticiones de un conjunto de cuatro módulos, habilitando así que se pueda elevar la capacidad de un sólo módulo de 16K a 64K. Dado que este diseño debe utilizarse en ciertas ocasiones en 16k de manera obligatoria, se agrega una entrada extra llamada MODO\_64K la cual indica como debe comportarse el MAD.

El conjunto de cuatro módulos de memoria se identifica comparando DA(3) con GA13A y DA(4) con GA14A. Si estos valores son idénticos, las ordenes sobre el GBUS son para ese conjunto en particular. Para identificar cada uno de los cuatro módulos de manera individual se utilizan las señales restantes: DA(1) se compara con GA15\_N y DA(2) con GA16\_N. Con esto, es posible determinar si el módulo debe responder o no ante cada acción de escritura o lectura.

## Ports

Port name	Direction	Type	Description
MIH_N	in	std_logic	MIH_N: (Memory inhibit): Se usa para inhibir POR '0' la operación normal de memoria y causa una acción de refresco.
GA16_A_GA15_N_REG	in	std_logic_vector (2 downto 1)	Bits más significativos que se utilizan para identificar los módulos.

Port name	Direction	Type	Description
GA14A_A_GA13A_N_REG	in	std_logic_vector (2 downto 1)	Bits menos significativos que se utilizan para identificar los módulos.
DA	in	std_logic_vector (4 downto 1)	Indentificador de la dirección del dispositivo (Device Adress).
MODO_64K	in	std_logic	Entrada proveniente del hardware que determina si el funcionamiento es como módulo de 16k estado lógico '0' o 64k estado lógico '1'.
EQ	out	std_logic	Señal interna para indicar si el módulo debe responder (estado lógico '1') o no (estado lógico '0') ante las peticiones en el GBUS.

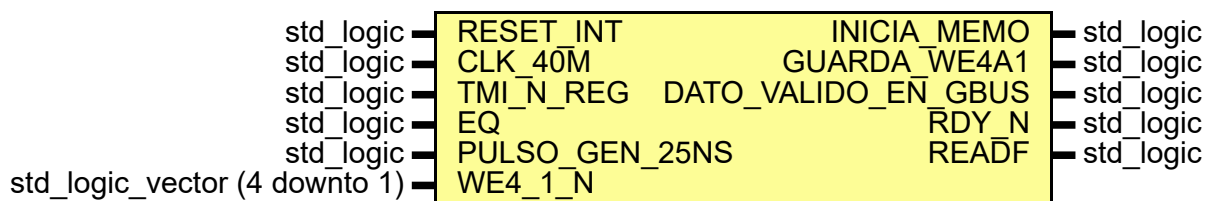
## Signals

Name	Type	Description
s_comp_msb	std_logic	Señal interna producto de la comparación de DA(4) y DA(3) con GA14A_N más GA13A.
s_comp_lsb	std_logic	Señal interna producto de la comparación de DA(2) y DA(1) con GA16_N más GA15_N.

## Entity: TAC

- File: TAC.vhd

## Diagram



## Description

El circuito de sincronización y control (TAC) procesa las señales necesarias para secuenciar los distintos ciclos de la memoria que se envían como señal de handshake a la unidad que está utilizando la memoria. En este bloque se detecta cuando se inicia un ciclo de memoria (INICIA\_MEMO = '1'). Además, se implementa una lógica que permite inferir el momento que está el valor correcto de GW y, así, indicar la acción a realizar con la memoria. También, se divide, para un ciclo de escritura, el momento donde los datos están establecidos en el bus (DATO\_VALIDO\_EN\_GBUS), para poder enviar al bloque correspondiente la señal de habilitación de los flip-flops que toman y retienen dichos datos. Por último, se implementa una lógica para evitar que DATO\_VALIDO\_EN\_GBUS = '1' si nos encontramos en un ciclo de lectura. Finalmente, se generan las señales de aviso RDY y READF con el temporizado específico del handshake original, según corresponda.

## Ports

Port name	Direction	Type	Description
RESET_INT	in	std_logic	Reset global.
CLK_40M	in	std_logic	Reloj de 40MHz.
TMI_N_REG	in	std_logic	Señal TMI_N libre de glitches proveniente del componente "Registrar Entradas" que indica el inicio de un ciclo normal de memoria (transport memory internal).
EQ	in	std_logic	Señal interna para indicar si el módulo debe responder (estado lógico '1') o no (estado lógico '0') ante las peticiones en el GBUS.
PULSO_GEN_25NS	in	std_logic	Reloj de 8 MHz sincronizado con el de 40MHz.
WE4_1_N	in	std_logic_vector (4 downto 1)	Bits para determinar el tipo de acción de memoria (escritura, lectura o escritura parcial).
INICIA_MEMO	out	std_logic	Señal que indica el comienzo de un ciclo de memoria.
GUARDA_WE4A1	out	std_logic	Señal hacia registrar_entradas que indica cuando se debe mantener los valores guardados de WE4A1. De esta manera se previene que cambien durante un ciclo de memoria.
DATO_VALIDO_EN_GBUS	out	std_logic	Señal que indica cuando el dato disponible en el GBUS está establecido de

Port name	Direction	Type	Description
			manera correcta y se puede extraer su valor.
RDY_N	out	std_logic	Señal del handshake entre el GBUS y la memoria. A través de RDY_N se informa al GBUS si la memoria se encuentra ejecutando un ciclo de memoria.
READF	out	std_logic	Esta señal informa al CPIO que los datos en los buffers de salida están listos a presentarse en el BUS-G.

## Signals

Name	Type	Description
S_INICIA_MEMO	std_logic	Señal auxiliar para informar a la FSM que se debe iniciar un ciclo de memoria.
S_ACCION_MEMO	std_logic	Señal auxiliar de GUARDA_WE4A1.
S_DATO_VALIDO_GBUS	std_logic	Señal auxiliar de DATO_VALIDO_GBUS.
S_RDY_N	std_logic	Señal auxiliar de RDY_N.
S_READF	std_logic	Señal auxiliar de READF.
S_READF_FLANCO_EXTRA	std_logic	Señal auxiliar que está retrasada un flanco reloj de GEN respecto a S_READF.

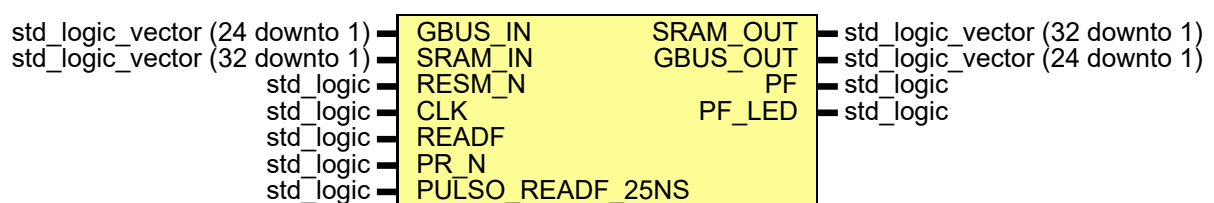
## Processes

- DETECTO\_INICIO\_SECUENCIA\_MEMO: ( RESET\_INT, CLK\_40M, PULSO\_GEN\_25NS )
  - Description**  
Se detecta la secuencia de inicio de memoria y se define el valor de S\_INICIA\_MEMO.
- DETERMINO\_ACCION\_MEMO: ( RESET\_INT, CLK\_40M, PULSO\_GEN\_25NS, S\_INICIA\_MEMO )
  - Description**  
Se define el valor de S\_ACCION\_MEMO y por ende de GUARDA\_WE4A1. Este proceso permite inferir el momento en el que se debe retener el valor en los pines GW4A1.
- DATO\_VALIDO\_GBUS: ( RESET\_INT, CLK\_40M, PULSO\_GEN\_25NS, S\_ACCION\_MEMO )
  - Description**  
Se determina el valor que toma la señal S\_DATO\_VALIDO\_GBUS a partir de la cual se generan DATO\_VALIDO\_EN\_GBUS, S\_RDY\_N y RDY\_N. Además, se la utiliza en otros procesos para sincronizar eventos.
- PULSO\_READF: ( RESET\_INT, CLK\_40M, PULSO\_GEN\_25NS, S\_DATO\_VALIDO\_GBUS )
  - Description**  
Se genera la señal S\_READF para luego generar la salida READF. Esta señal indica cuando se está ejecutando un ciclo de lectura de memoria.
- PULSO\_EXTRA\_READF: ( RESET\_INT, CLK\_40M, PULSO\_GEN\_25NS, S\_READF )
  - Description**  
Se crea la lógica para la señal S\_READF\_FLANCO\_EXTRA, la cual debe mantener el valor de S\_READF durante un flanco más de GEN.

## Entity: PARIDAD\_v0

- File: PARIDAD\_v0.vhd

## Diagram



## Description

El componente Paridad\_v0 se encarga de generar los bits de paridad correspondientes para la escritura de datos, de comprobar si existen fallos de paridad en la lectura y de generar la señal PF (Parity Fault) del G-BUS, al igual que controlar el LED indicador de fallo de paridad PF\_LED. Para esto hace uso de dos bloques más PFF\_v0 y GEN\_PAR\_v0.

Cuando un dato proviene del G-BUS (ciclo de escritura), se generan los 4 bits de paridad a partir de los 24 bits de datos (GBUS\_IN). En la operación inversa (ciclo de lectura), se comprueba que los bits de paridad de los datos almacenados (SRAM\_IN) sean iguales a los que fueron generados al momento de la escritura.

Para determinar si se encuentra en un ciclo de lectura se utiliza la entrada READF.

READF toma el valor '1' lógico en un ciclo de lectura y a partir de esto, se habilita

la comprobación de paridad. La entrada PULSO\_READF\_25NS es un registro de READF y se utiliza para detectar un flanco descendente en READF.

## Ports

Port name	Direction	Type	Description
GBUS_IN	in	std_logic_vector (24 downto 1)	24 bits de datos que salen del GBUS y entran en este bloque para generar paridad.
SRAM_IN	in	std_logic_vector (32 downto 1)	32 bits de datos que vienen desde la SRAM y entran en este bloque para comprobar la paridad.
RESM_N	in	std_logic	Reset.
CLK	in	std_logic	Reloj de 40MHz.
READF	in	std_logic	Entrada proveniente del bloque TAC que se utiliza como bandera para determinar si hay un ciclo de lectura o escritura. En '1' lógico indica ciclo de lectura, por lo tanto, se habilita el bloque de comprobación de paridad. Si esta en '0' indica ciclo de escritura, por lo tanto, se deshabilita el bloque de paridad.
PR_N	in	std_logic	Entrada proveniente del GBUS e indica si la paridad utilizada es par ('1' lógico) o impar ('0' lógico).
PULSO_READF_25NS	in	std_logic	Se la utiliza para detectar un flanco descendente en READF. Viene de RAM_MU y entra a PFF_v0
SRAM_OUT	out	std_logic_vector (32 downto 1)	32 bits de datos que salen de este bloque y van a la memoria con la generación de paridad incluida.
GBUS_OUT	out	std_logic_vector (24 downto 1)	24 bits de datos que salen de este bloque y se envían al GBUS.
PF	out	std_logic	Señal que le indica falla de paridad al GBUS.
PF_LED	out	std_logic	Señal para encender el LED si hubo falla de paridad.

## Signals

Name	Type	Description
s_SRAM_IN	std_logic_vector (24 downto 1)	Señal interna que comprende los 24 bits (menos significativos) de datos que se obtienen desde la entrada SRAM_IN y se utilizarán para generar paridad y comprobar que los datos no estén corrompidos.
s_SRAM_IN_PAR	std_logic_vector (32 downto 1)	Es el resultado de la señal de entrada luego de pasar por el generador de paridad.
s_PFA	std_logic	Señal interna que determina si existe o no falla de paridad en la lectura de datos desde la SRAM.
s_SRAM_OUT_32	std_logic_vector (32 downto 1)	Señal interna de 32 bits (24 de datos más 4 de paridad) que se obtienen al pasar la señal GBUS_IN por el generador de paridad obteniendo los datos más los 4 bits de paridad.

## Instantiations

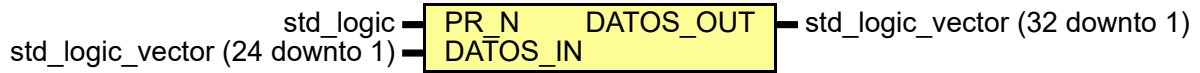
- GEN\_PAR\_GBUS: GEN\_PAR\_v0
  - Se realiza una instancia del componente GEN\_PAR\_v0 para generar los bits de paridad a partir de los datos del GBUS y escribirlos en la memoria SRAM.
- CHEQUEO\_PAR\_SRAM: GEN\_PAR\_v0
  - Se realiza otra instancia del componente GEN\_PAR\_v0 para comprobar la paridad de los datos obtenidos desde la memoria SRAM. Se comparan los 4 bits generados a partir de los 24 bits guardados y los 4 bits que se generaron en la escritura. A partir de esto, se da valor a la señal s\_PFA en '0' cuando hay fallo de paridad (datos distintos) y '1' cuando no lo hay (datos iguales).
- PFF: PFF\_v0

- Se realiza una instancia del componente PFF\_v0 para sincronizar la señal interna de parity fault con el handshake del GBUS.

## Entity: GEN\_PAR\_v0

- **File:** GEN\_PAR\_v0.vhd

### Diagram



### Description

Este módulo se encarga de generar los 4 bits de paridad a partir de una palabra de 24 bits, de acuerdo al funcionamiento de la memoria original SMR-MU.

Las entrada PR\_N indica si se debe generar paridad par o impar a partir de los 24 bits de DATOS\_IN. La salida DATOS\_OUT posee los valores de DATOS\_IN más los cuatro bits de paridad generados. Como la memoria SRAM guarda datos de a 8 bits, se debe generar una señal de 32 bits los cuales los últimos 4 no se usan y se completan con '0'.

La paridad se genera de la siguiente manera, los primeros ocho bits (1 a 8 de DATOS\_IN) se asignan al primer bit de paridad. Los siguiente cuatro (9 a 12 de DATOS\_IN) se utilizan para el segundo bit y el tercero se realiza a partir de próximos cuatro (13 a 16 de DATOS\_IN). El último bit de paridad se realiza con los últimos ocho bits de entrada (17 a 24 de DATOS\_IN).

### Ports

Port name	Direction	Type	Description
PR_N	in	std_logic	Entrada que indica el tipo de paridad a ejecutar (Par o Impar).
DATOS_IN	in	std_logic_vector (24 downto 1)	Entrada de datos para controlar paridad.
DATOS_OUT	out	std_logic_vector (32 downto 1)	Salida de 24 bits de datos mas 4 bits de paridad.

### Signals

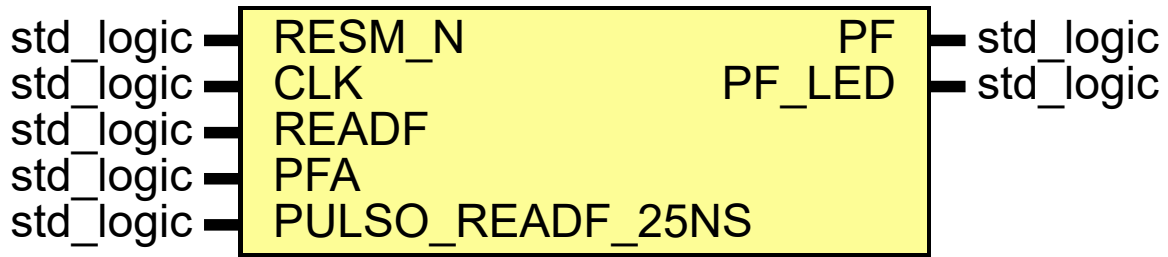
Name	Type	Description
a	std_logic	Señales internas para generar el primer bit de paridad
b	std_logic	Señales internas para generar el primer bit de paridad
c	std_logic	Señales internas para generar el primer bit de paridad
d	std_logic	Señales internas para generar el primer bit de paridad
e	std_logic	Señales internas para generar el primer bit de paridad
f	std_logic	Señales internas para generar el primer bit de paridad
g	std_logic	Señales internas para generar el primer bit de paridad
h	std_logic	Señales internas para generar el segundo bit de paridad.
i	std_logic	Señales internas para generar el segundo bit de paridad.
j	std_logic	Señales internas para generar el segundo bit de paridad.
k	std_logic	Señales internas para generar el tercer bit de paridad.
l	std_logic	Señales internas para generar el tercer bit de paridad.
m	std_logic	Señales internas para generar el tercer bit de paridad.
n	std_logic	Señales internas para generar el cuarto bit de paridad.
o	std_logic	Señales internas para generar el cuarto bit de paridad.
p	std_logic	Señales internas para generar el cuarto bit de paridad.
q	std_logic	Señales internas para generar el cuarto bit de paridad.
r	std_logic	Señales internas para generar el cuarto bit de paridad.
s	std_logic	Señales internas para generar el cuarto bit de paridad.
t	std_logic	Señales internas para generar el cuarto bit de paridad.

Name	Type	Description
Bit_Paridad	std_logic_vector (4 downto 1)	Paridad, PD(4 a 1) o PO(4 a 1).
DATOS_OUT_signal	std_logic_vector(32 downto 1)	Señal auxiliar.

## Entity: PFF\_v0

- File: PFF\_v0.vhd

### Diagram



### Description

V11\_3

Esté módulo se encarga de generar las señales de Parity Fault (PF) y del led indicador de Parity Fault (PF\_LED). La entrada PFA indica si existe un fallo en la comprobación de paridad ('1' falla de paridad y '0' sin falla) y las señales READF y PULSO\_READF\_25NS se utilizan para sincronizar la salida PF con el handshake del GBUS.

### Ports

Port name	Direction	Type	Description
RESM_N	in	std_logic	Reset.
CLK	in	std_logic	Reloj de 40MHZ
READF	in	std_logic	Señal que indica que la memoria se encuentra lista para escribir en el GBUS.
PFA	in	std_logic	Señal que indica si existió un fallo de paridad.
PULSO_READF_25NS	in	std_logic	Salida del registro de READF, está atrasada 25ns.
PF	out	std_logic	Salida hacia RAM_MU.vhd, informa un '1' lógico cuando hay fallo de paridad y '0' cuando no lo hay.
PF_LED	out	std_logic	Led de falla de paridad, igual comportamiento que PF.

### Signals

Name	Type	Description
s_PF	std_logic	Señal interna para PF.
s_PF_LED	std_logic	Señal interna para PF_LED.

### Processes

- proc\_PFF: ( RESM\_N, CLK, PFA, READF, PULSO\_READF\_25NS )

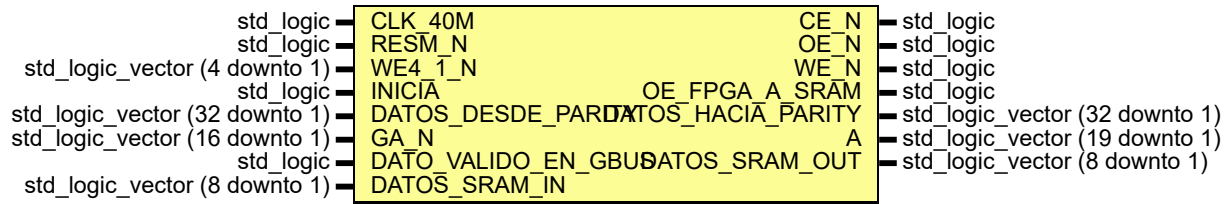
- **Description**

Este proceso se encarga de sincronizar la señal de fallo de paridad de acuerdo al handshake de la memoria original. Para ello, se detecta un flanco descendente en READF y si hubo un fallo de paridad se informa. Se debe tener en cuenta que la única manera de realizar PF = '0' es con un reset de memoria.

## Entity: Prueba\_FSM\_y\_MANEJASRAM

- File: Prueba\_FSM\_y\_MANEJASRAM.vhd

## Diagram



## Description

Este módulo se encarga de administrar los datos que se almacenan en la memoria y , por ende, del manejo del integrado de memoria SRAM. Está compuesto por dos componentes: una máquina de 18 estados y una implementación de registros de datos que contiene toda la lógica necesaria para para realizar la descomposición de la palabra de 32 bits (24 bits de dato + 4 bits de paridad + 4 bits en '0'). Este diseño no agrega lógica adicional, solo interconecta las entradas y salidas pertinentes a los componentes internos y externos.

La forma de escribir o leer se realiza de a 8 bits, ya que de esta manera funciona el integrado de memoria SRAM (el manejo de datos lo hace mediante 8 pines de entrada-salida). El bloque recibe los 4 bits (WE1 a WE4) que le indican el formato de la palabra que se irán a escribir o leer de la memoria. Del bloque TAC recibe la señal INICIA\_MEMO que le indica cuando iniciar un ciclo de memoria, junto con la señal DATO\_VALIDO\_EN\_GBUS. Del bloque PARIDAD obtiene los datos provenientes del G-BUS junto con la paridad generada sobre estos. Además, si el GBUS quiere leer datos desde la memoria, este bloque interactuará con el bloque de PARIDAD para que genere paridad nuevamente y compare los bits de paridad previamente guardados en memoria con los bits de paridad obtenidos al leer el dato de la misma, y finalmente copiara esos datos en el G-BUS para que pueda ser leídos. Por otro lado, se encargará de escribir el dato como se lo indiquen las señales WE (ya sea, escribir la palabra completa, la mitad superior de la palabra, el carácter inferior, entre otros casos).

## Ports

Port name	Direction	Type	Description
CLK_40M	in	std_logic	Reloj de 40MHz.
RESM_N	in	std_logic	Reset global
WE4_1_N	in	std_logic_vector (4 downto 1)	Bits para determinar el tipo de acción de memoria (escritura, lectura o escritura parcial).
INICIA	in	std_logic	Señal que indica el comienzo de un ciclo de memoria.
DATOS_DESDE_PARITY	in	std_logic_vector (32 downto 1)	24 bits de datos a guardarse dentro de la SRAM más los 4 bits de paridad correspondientes.
GA_N	in	std_logic_vector (16 downto 1)	Dirección de la palabra de 24 bits que se quiere escribir/leer con el formato original de 16 bits del GBUS.
DATO_VALIDO_EN_GBUS	in	std_logic	Señal que indica cuando el dato disponible en el GBUS es válido y se puede extraer su valor.
DATOS_SRAM_IN	in	std_logic_vector (8 downto 1)	Byte de datos provenientes de cada dirección de la memoria SRAM.
CE_N	out	std_logic	Señal para manejar el pin chip enable de la memoria SRAM.
OE_N	out	std_logic	Señal para manejar el pin output enable de la memoria SRAM.
WE_N	out	std_logic	Señal para manejar el pin write enable de la memoria SRAM.
OE_FPGA_A_SRAM	out	std_logic	Señal para manejar el buffer bidireccional de 8 bits entre la FPGA y el IC de SRAM.
DATOS_HACIA_PARITY	out	std_logic_vector (32 downto 1)	24 bits de datos a leerse de la SRAM + 4 bits de paridad correspondientes.
A	out	std_logic_vector (19 downto 1)	Dirección de cada dato de la memoria SRAM.
DATOS_SRAM_OUT	out	std_logic_vector (8 downto 1)	Datos de cada dirección de memoria SRAM.

## Signals

Name	Type	Description
S_A1y2	std_logic_vector (2 downto 1)	Direcciones de cada byte dentro de la plabra de 32 bits en la memoria SRAM.

Name	Type	Description
S_RETENGO_VALOR_FF_GBUS	std_logic	Se controla la retención de los datos presentes en el GBUS para que no cambien durante la escritura de la SRAM.
S_hab_escrit_ff_sram_byte1	std_logic	Habilitación de escritura de los FF usados para LEER DATOS DE LA SRAM
S_hab_escrit_ff_sram_byte2	std_logic	Habilitación de escritura de los FF usados para LEER DATOS DE LA SRAM
S_hab_escrit_ff_sram_byte3	std_logic	Habilitación de escritura de los FF usados para LEER DATOS DE LA SRAM
S_hab_escrit_ff_sram_byte4	std_logic	Habilitación de escritura de los FF usados para LEER DATOS DE LA SRAM

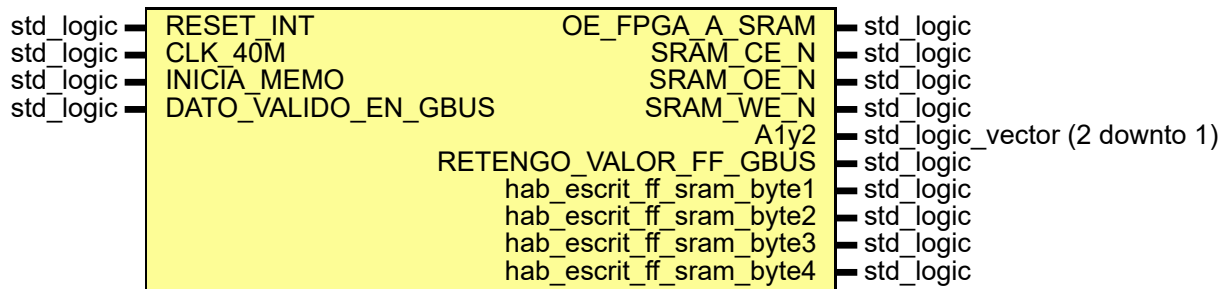
## Instantiations

- FSM: FSM\_LEEYESCRIBE4B
  - Peter pan.
- REGISTROS\_IO\_SRAM: HW\_REGISTROS\_IO\_SRAM

## Entity: FSM\_LEEYESCRIBE4B

- File: FSM\_LEEYESCRIBE4B.vhd

## Diagram



## Description

Este bloque describe la maquina de estados (FSM) que se complementa con HW\_REGISTROS para implementar la lógica de los ciclos de escritura y lectura en memoria SRAM y el control del hardware. Cada vez que se inicia un ciclo de memoria por medio de la entrada INICIA\_MEMO, se extraen los datos de memoria modificando los pines de control de la SRAM y la dirección A1y2 que compone cada dato. En caso que se trate de un ciclo de escritura o escritura parcial la FSM sigue con el proceso de colocar cada byte del dato en la SRAM.

## Ports

Port name	Direction	Type	Description
RESET_INT	in	std_logic	Reset global
CLK_40M	in	std_logic	Clock de 40 MHz.
INICIA_MEMO	in	std_logic	Señal que indica el comienzo de un ciclo de memoria.
DATO_VALIDO_EN_GBUS	in	std_logic	Señal que indica cuando el dato disponible en el GBUS es válido y se puede extraer su valor.
OE_FPGA_A_SRAM	out	std_logic	Señal para manejar el buffer bidireccional de 8 bits entre la FPGA y el IC de SRAM.
SRAM_CE_N	out	std_logic	Señal para manejar el pin chip enable de la memoria SRAM.
SRAM_OE_N	out	std_logic	Señal para manejar el pin output enable de la memoria SRAM.
SRAM_WE_N	out	std_logic	Señal para manejar el pin write enable de la memoria SRAM.
A1y2	out	std_logic_vector (2 downto 1)	Direcciones de cada byte dentro de la palabra de 32 bits en la memoria SRAM.
RETENGO_VALOR_FF_GBUS	out	std_logic	Se controla la retención de los datos presentes en el GBUS para que no cambien durante la escritura de la SRAM.
hab_escrit_ff_sram_byte1	out	std_logic	Habilitación de escritura de los FF usados para LEER DATOS DE LA SRAM

Port name	Direction	Type	Description
hab_escrit_ff_sram_byte2	out	std_logic	Habilitación de escritura de los FF usados para LEER DATOS DE LA SRAM
hab_escrit_ff_sram_byte3	out	std_logic	Habilitación de escritura de los FF usados para LEER DATOS DE LA SRAM
hab_escrit_ff_sram_byte4	out	std_logic	Habilitación de escritura de los FF usados para LEER DATOS DE LA SRAM

## Signals

Name	Type	Description
estado_act	STATE_TYPE	Señal para que indica el estado actual de la FSM.
estado_sig	STATE_TYPE	Señal para que indica el proximo estado de la FSM.
s_cond	std_logic_vector (3 downto 0)	Señal que sirve para determina en que condiciones se realizan los cambios de estado.
s_hab_escrit_ff_sram_byte1	std_logic	Señal auxiliar de la salida hab_escrit_ff_sram_byte1.
s_hab_escrit_ff_sram_byte2	std_logic	Señal auxiliar de la salida hab_escrit_ff_sram_byte2.
s_hab_escrit_ff_sram_byte3	std_logic	Señal auxiliar de la salida hab_escrit_ff_sram_byte3.
s_hab_escrit_ff_sram_byte4	std_logic	Señal auxiliar de la salida hab_escrit_ff_sram_byte4.

## Types

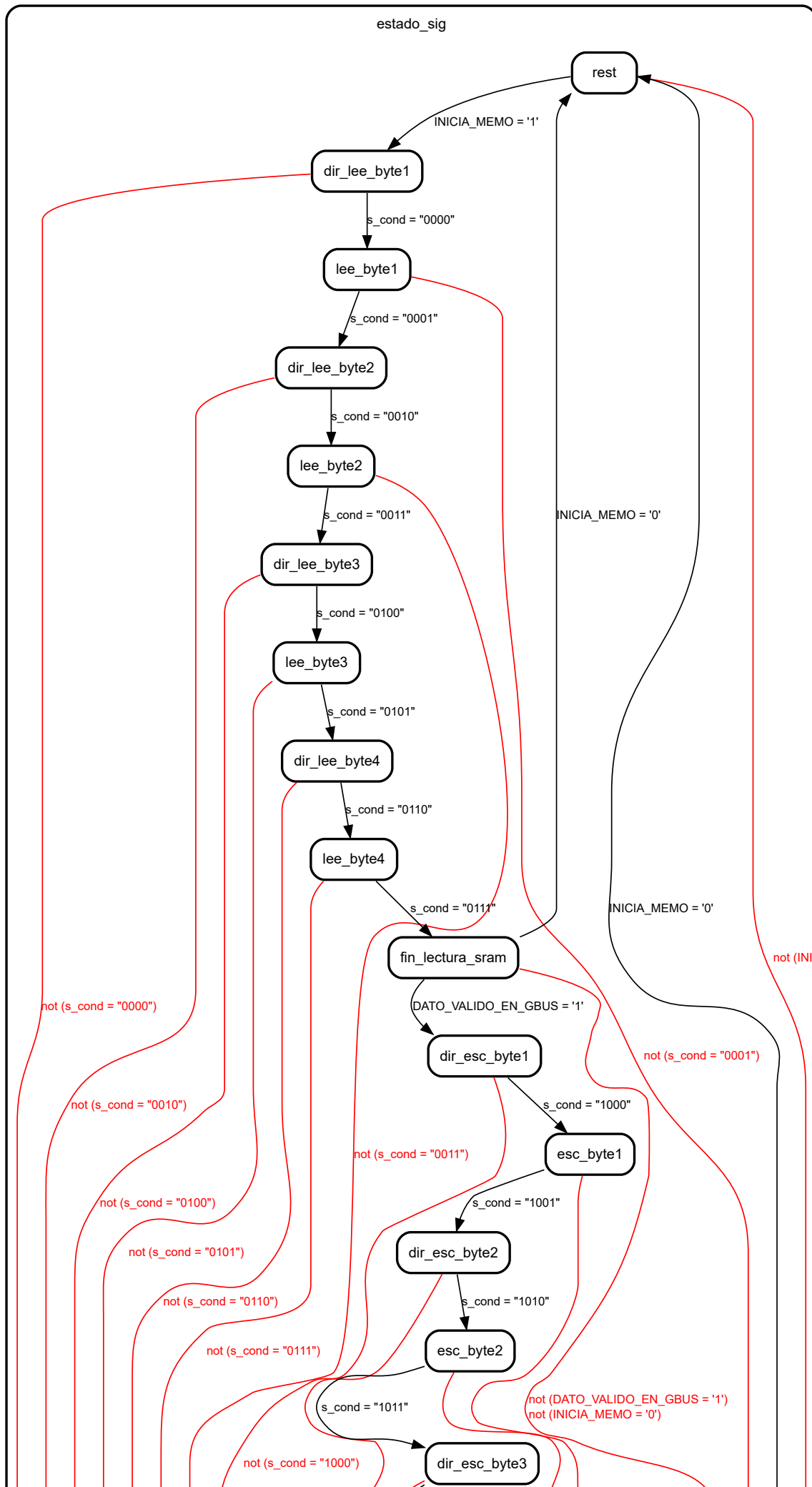
Name	Type	Description
STATE_TYPE	( rest, dir_lee_byte1, lee_byte1, dir_lee_byte2, lee_byte2, dir_lee_byte3, lee_byte3, dir_lee_byte4, lee_byte4, fin_lectura_sram, dir_esc_byte1, esc_byte1, dir_esc_byte2, esc_byte2, dir_esc_byte3, esc_byte3, dir_esc_byte4, esc_byte4 )	Estados para los ciclos de memoria.

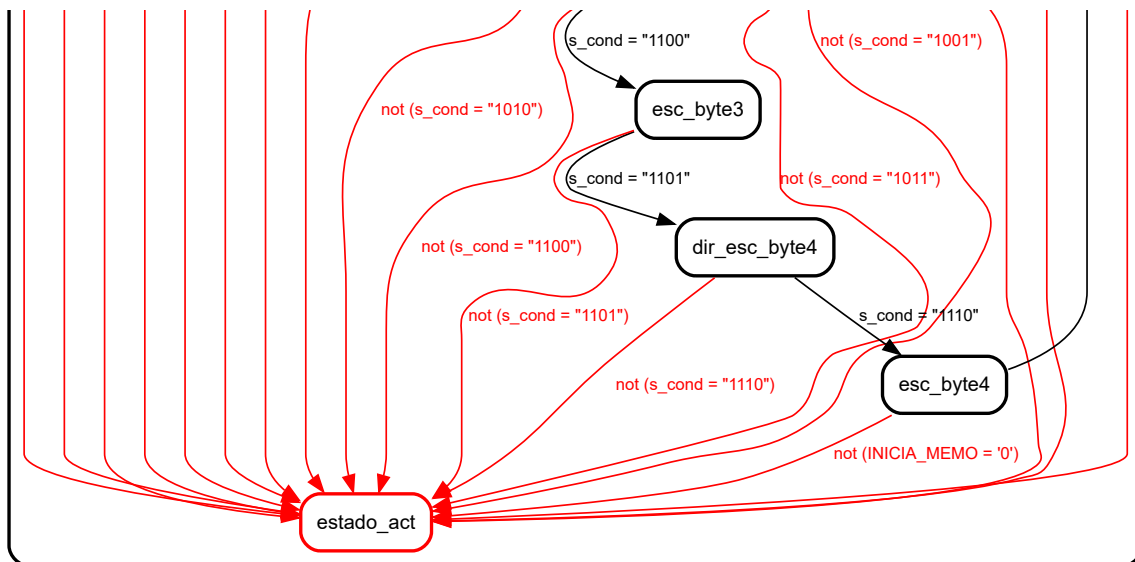
## Processes

- proc\_inicial: ( RESET\_INT, CLK\_40M, estado\_act, estado\_sig )
  - **Description**  
Proceso encargado de volver al estado inicial la máquina cuando recibe un pulso de RESET\_INT y de cambiar el estado de manera sincronica.
- proc\_prox\_estado: ( estado\_act, estado\_sig, INICIA\_MEMO, s\_cond, DATO\_VALIDO\_EN\_GBUS )
  - **Description**  
Se describen las condiciones para las transiciones de estado de la FSM.
- proc\_salidas\_segun\_estado: ( estado\_act, INICIA\_MEMO )
  - **Description**  
Este proceso es el encargado de asignarle los distintos valores a las salidas del bloque en función del estado actual.

## State machines

- Se describen las condiciones para las transiciones de estado de la FSM.

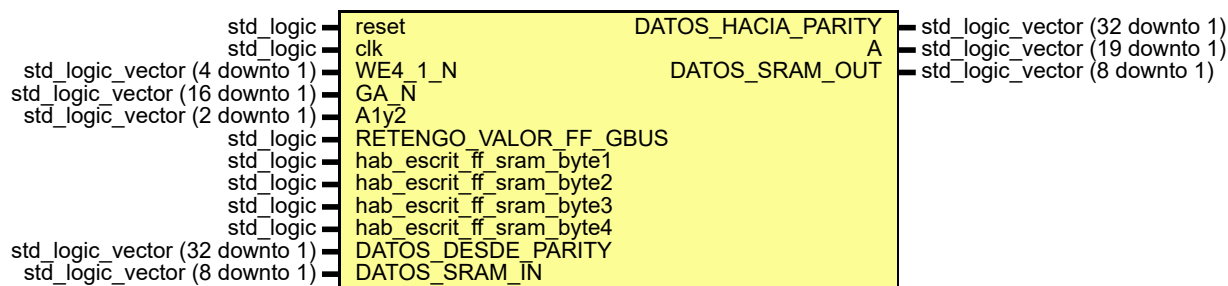




## Entity: HW\_REGISTROS\_IO\_SRAM

- File: HW\_REGISTROS\_IO\_SRAM.vhd

### Diagram



### Description

V11\_3

Este módulo es el encargado de interactuar a bajo nivel con el chip de SRAM. Coloca y extrae valores, tanto en el módulo de paridad como en la memoria SRAM, en función de las entradas que recibe del bloque FSM\_LEEYESCRIBE4B. Esta maquina de estados accede a los datos guardados en la SRAM en todas las acciones, debido a que en las escrituras parciales no se deben sobrescribir los datos almacenados. Teniendo en cuenta que el chip de memoria SRAM puede ser accedido de a 1 byte a la vez, se deben escribir los 28 bits del dato en bloques de 32 bits. De esta manera un dato queda dividido en SRAM en 4 partes iguales de 8 bits. Para poder direccionar estos, se deben crear una nueva dirección que traduzca la dirección original de 16 bits a una de 18. Los bytes extraídos de cada dirección se envían al bloque encargado de comprobar la paridad.

En primer lugar se define, para cada grupo de 8 bits que componen el dato de 32 bits a almacenar en SRAM, mascarar de bits que serán utilizadas para discriminar los bits utilizados en ese ciclo de memoria. Esta operación es particularmente importante para el caso de las escrituras parciales, ya que se desea mantener los datos previamente escritos. Luego, se realiza la operación AND entre los datos que ingresan desde el bloque de paridad y las mascarar. El resultado de estas operaciones son registradas por flip flops, los cuales poseen una entrada enable gobernada por la maquina de estados. De esta manera se retiene el dato proveniente del bloque de paridad cuando es necesario.

Para el caso de los datos provenientes de la SRAM se realiza el mismo procedimiento que se describió anteriormente. En este caso, las mascarar se deben conformar con el valor correspondiente para poner en '0' la parte de la palabra que debe ser sobre-escrita por los datos que vienen del GBUS, es decir desde el modulo de paridad.

A la hora de colocar los datos finales al bloque de paridad se copian directamente desde los flip flops que almacenan los valores leídos desde la SRAM. Para la copia de valores desde la entrada del parity hacia la SRAM se realiza una operación XOR a partir de los datos almacenados en los registros del GBUS y la memoria SRAM. De esta manera, se asegura que sólo sean modificados los bits necesarios en cada dirección de la memoria.

### Ports

Port name	Direction	Type	Description
reset	in	std_logic	Reset global
clk	in	std_logic	Clock
WE4_1_N	in	std_logic_vector (4 downto 1)	Bits para determinar el tipo de acción de memoria (escritura, lectura o escritura parcial).
GA_N	in	std_logic_vector (16 downto 1)	Dirección de la palabra de 24 bits que se quiere escribir/leer con el formato original de 16 bits del GBUS.
A1y2	in	std_logic_vector (2 downto 1)	Bits extra para la dirección de la memoria SRAM. Son dos ya que cada dato almacenado en SRAM se divide en 4 bytes.
RETENGO_VALOR_FF_GBUS	in	std_logic	Entrada que indica cuando es necesario retener el dato presente en el GBUS porque se inicia un ciclo de escritura en memoria.
hab_escrit_ff_sram_byte1	in	std_logic	Entrada que indica cuando se puede escribir el byte 1 perteneciente al dato en la SRAM.
hab_escrit_ff_sram_byte2	in	std_logic	Entrada que indica cuando se puede escribir el byte 2 perteneciente al dato en la SRAM.
hab_escrit_ff_sram_byte3	in	std_logic	Entrada que indica cuando se puede escribir el byte 3 perteneciente al dato en la SRAM.
hab_escrit_ff_sram_byte4	in	std_logic	Entrada que indica cuando se puede escribir el byte 4 perteneciente al dato en la SRAM.
DATOS_DESDE_PARITY	in	std_logic_vector (32 downto 1)	24 bits de datos a guardarse dentro de la SRAM más los 4 bits de paridad correspondientes.
DATOS_SRAM_IN	in	std_logic_vector (8 downto 1)	Byte de datos provenientes de cada dirección de la memoria SRAM.
DATOS_HACIA_PARITY	out	std_logic_vector (32 downto 1)	24 bits de datos a leerse de la SRAM + 4 bits de paridad correspondientes.
A	out	std_logic_vector (19 downto 1)	Dirección de cada dato de la memoria SRAM.
DATOS_SRAM_OUT	out	std_logic_vector (8 downto 1)	Datos de cada dirección de memoria SRAM.

## Signals

Name	Type	Description
s_and_gbus_1	std_logic_vector (8 downto 1)	Mascara de bits para discriminar los bits que se deben modificar en una operacion de memoria.
s_and_gbus_2	std_logic_vector (8 downto 1)	Mascara de bits para discriminar los bits que se deben modificar en una operacion de memoria.
s_and_gbus_3	std_logic_vector (8 downto 1)	Mascara de bits para discriminar los bits que se deben modificar en una operacion de memoria.
s_and_gbus_4	std_logic_vector (8 downto 1)	Mascara de bits para discriminar los bits que se deben modificar en una operacion de memoria.
s_ffi_parity_1	std_logic_vector (8 downto 1)	Señales entrantes a los FF de salida de datos hacia la SRAM.
s_ffi_parity_2	std_logic_vector (8 downto 1)	Señales entrantes a los FF de salida de datos hacia la SRAM.
s_ffi_parity_3	std_logic_vector (8 downto 1)	Señales entrantes a los FF de salida de datos hacia la SRAM.
s_ffi_parity_4	std_logic_vector (8 downto 1)	Señales entrantes a los FF de salida de datos hacia la SRAM.
s_ffo_parity_1	std_logic_vector (8 downto 1)	Señales salientes de los FF de salida de datos hacia la SRAM.
s_ffo_parity_2	std_logic_vector (8 downto 1)	Señales salientes de los FF de salida de datos hacia la SRAM.
s_ffo_parity_3	std_logic_vector (8 downto 1)	Señales salientes de los FF de salida de datos hacia la SRAM.
s_ffo_parity_4	std_logic_vector (8 downto 1)	Señales salientes de los FF de salida de datos hacia la SRAM.
s_xor1	std_logic_vector (8)	Señal auxiliar de 8 bit para conformar las XOR de salida de datos a la SRAM

Name	Type	Description
	downto 1)	
s_xor2	std_logic_vector (8 downto 1)	Señal auxiliar de 8 bit para conformar las XOR de salida de datos a la SRAM
s_xor3	std_logic_vector (8 downto 1)	Señal auxiliar de 8 bit para conformar las XOR de salida de datos a la SRAM
s_xor4	std_logic_vector (8 downto 1)	Señal auxiliar de 8 bit para conformar las XOR de salida de datos a la SRAM
s_iosram_o	std_logic_vector (8 downto 1)	Señal de 8 bit para conformar el mux de salida de datos de los 4 flip flops que tienen cargados los valores tomados del modulo de paridad provenientes del G-BUS a la SRAM.
s_iosram_i	std_logic_vector (8 downto 1)	Señal de 8 bit para conformar el mux de entrada de datos a los 4 FF que van a cargar los valores que se toman de la SRAM.
s_ff1_sram_1	std_logic_vector (8 downto 1)	Señal que conecta la entrada de los registros de SRAM.
s_ffo_sram_1	std_logic_vector (8 downto 1)	Señal que conecta la salida de los registros de SRAM.
s_and_sram_1	std_logic_vector (8 downto 1)	Mascara de bits utilizada para discriminar los bits provenientes de la SRAM.
s_ff1_sram_2	std_logic_vector (8 downto 1)	Señal que conecta la entrada de los registros de SRAM.
s_ffo_sram_2	std_logic_vector (8 downto 1)	Señal que conecta la salida de los registros de SRAM.
s_and_sram_2	std_logic_vector (8 downto 1)	Mascara de bits utilizada para discriminar los bits provenientes de la SRAM.
s_ff1_sram_3	std_logic_vector (8 downto 1)	Señal que conecta la entrada de los registros de SRAM..
s_ffo_sram_3	std_logic_vector (8 downto 1)	Señal que conecta la salida de los registros de SRAM.
s_and_sram_3	std_logic_vector (8 downto 1)	Mascara de bits utilizada para discriminar los bits provenientes de la SRAM.
s_ff1_sram_4	std_logic_vector (8 downto 1)	Señal que conecta la entrada de los registros de SRAM.
s_ffo_sram_4	std_logic_vector (8 downto 1)	Señal que conecta la salida de los registros de SRAM.
s_and_sram_4	std_logic_vector (8 downto 1)	Mascara de bits utilizada para discriminar los bits provenientes de la SRAM.
s_xor_sram_1	std_logic_vector (8 downto 1)	Señal auxiliar de 8 bit para conformar las XOR de salida de datos provenientes de la SRAM.
s_xor_sram_2	std_logic_vector (8 downto 1)	Señal auxiliar de 8 bit para conformar las XOR de salida de datos provenientes de la SRAM.
s_xor_sram_3	std_logic_vector (8 downto 1)	Señal auxiliar de 8 bit para conformar las XOR de salida de datos provenientes de la SRAM.
s_xor_sram_4	std_logic_vector (8 downto 1)	Señal auxiliar de 8 bit para conformar las XOR de salida de datos provenientes de la SRAM.

## Processes

- proc\_ff1\_escr: ( reset, clk, RETENGO\_VALOR\_FF\_GBUS )
  - **Description**  
Se sintetiza el primer registro de los cuatro que componen el dato a almacenar en SRAM.  
La señal de entrada es s\_ff1\_parity\_1 que se la asigna a s\_ffo\_parity\_1.  
Sólo se habilita el funcionamiento del registro (copiar la entrada a la salida) cuando RETENGO\_VALOR\_FF\_GBUS es '1'.
- proc\_ff2\_escr: ( reset, clk, RETENGO\_VALOR\_FF\_GBUS )
  - **Description**  
Se sintetiza el segundo registro de los cuatro que componen el dato a almacenar en SRAM.  
La señal de entrada es s\_ff2\_parity\_2 que se la asigna a s\_ffo\_parity\_2.  
Sólo se habilita el funcionamiento del registro (copiar la entrada a la salida) cuando RETENGO\_VALOR\_FF\_GBUS es '1'.

- proc\_ff3\_escri: ( reset, clk, RETENGO\_VALOR\_FF\_GBUS )
  - **Description**  
 Se sintetiza el tercer registro de los cuatro que componen el dato a almacenar en SRAM.  
 La señal de entrada es s\_ff\_i\_parity\_3 que se la asigna a s\_ffo\_parity\_3.  
 Sólo se habilita el funcionamiento del registro (copiar la entrada a la salida)  
 cuando RETENGO\_VALOR\_FF\_GBUS es '1'.
- proc\_ff4\_escri: ( reset, clk, RETENGO\_VALOR\_FF\_GBUS )
  - **Description**  
 Se sintetiza el cuarto registro de los cuatro que componen el dato a almacenar en SRAM.  
 La señal de entrada es s\_ff\_i\_parity\_4 que se la asigna a s\_ffo\_parity\_4.  
 Sólo se habilita el funcionamiento del registro (copiar la entrada a la salida)  
 cuando RETENGO\_VALOR\_FF\_GBUS es '1'.
- demux\_bytes\_SRAM: ( A1y2, s\_iosram\_i )
  - **Description**  
 Demultiplexor para dar valores a s\_ff\_i\_sram\_x en funcion del byte que se esté leyendo en ese momento.
- proc\_ff1\_lectura: ( reset, clk, hab\_escrit\_ff\_sram\_byte1 )
  - **Description**  
 Se sintetiza el primer registro de los cuatro que componen el dato proveniente de la SRAM.  
 La señal de entrada es s\_ff\_i\_sram1 que se la asigna a s\_ffo\_sram1.  
 Sólo se habilita el funcionamiento del registro (copiar la entrada a la salida)  
 cuando hab\_escrit\_ff\_sram\_byte1 es '1'.
- proc\_ff2\_lectura: ( reset, clk, hab\_escrit\_ff\_sram\_byte2 )
  - **Description**  
 Se sintetiza el segundo registro de los cuatro que componen el dato proveniente de la SRAM.  
 La señal de entrada es s\_ff\_i\_sram2 que se la asigna a s\_ffo\_sram2.  
 Sólo se habilita el funcionamiento del registro (copiar la entrada a la salida)  
 cuando hab\_escrit\_ff\_sram\_byte2 es '1'.
- proc\_ff3\_lectura: ( reset, clk, hab\_escrit\_ff\_sram\_byte3 )
  - **Description**  
 Se sintetiza el tercer registro de los cuatro que componen el dato proveniente de la SRAM.  
 La señal de entrada es s\_ff\_i\_sram3 que se la asigna a s\_ffo\_sram3.  
 Sólo se habilita el funcionamiento del registro (copiar la entrada a la salida)  
 cuando hab\_escrit\_ff\_sram\_byte3 es '1'.
- proc\_ff4\_lectura: ( reset, clk, hab\_escrit\_ff\_sram\_byte4 )
  - **Description**  
 Se sintetiza el cuarto registro de los cuatro que componen el dato proveniente de la SRAM.  
 La señal de entrada es s\_ff\_i\_sram4 que se la asigna a s\_ffo\_sram4.  
 Sólo se habilita el funcionamiento del registro (copiar la entrada a la salida)  
 cuando hab\_escrit\_ff\_sram\_byte4 es '1'.