DISEÑO DE UN SISTEMA HW PARA EVALUAR EL CONSUMO DE POTENCIA DE MEMORIAS SRAM CON ECC

DESIGN OF A HARDWARE SYSTEM FOR EVALUATING POWER CONSUMPTION OF SRAM MEMORIES WITH ECC

TRABAJO FIN DE GRADO
CURSO 2019-2020

AUTORES
GONZALO FERNÁNDEZ-DÍEZ PONTE
RAQUEL RAMOS CORRAL
BEATRIZ VILLEGAS SÁNCHEZ

CODIRECTORES
JUAN ANTONIO CLEMENTE BARREIRA
JUAN CARLOS FABERO JIMÉNEZ

GRADO EN INGENIERÍA INFORMÁTICA Y COMPUTADORES
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

DISEÑO DE UN SISTEMA HW PARA EVALUAR EL CONSUMO DE POTENCIA DE MEMORIAS SRAM CON ECC

DESIGN OF A HARDWARE SYSTEM FOR EVALUATING POWER CONSUMPTION OF SRAM MEMORIES WITH ECC

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA Y COMPUTADORES
DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

AUTORES
GONZALO FERNÁNDEZ-DÍEZ PONTE
RAQUEL RAMOS CORRAL
BEATRIZ VILLEGAS SÁNCHEZ

CODIRECTORES
JUAN ANTONIO CLEMENTE BARREIRA
JUAN CARLOS FABERO JIMÉNEZ

CONVOCATORIA: SEPTIEMBRE 2020
CALIFICACIÓN: NOTA

GRADO EN INGENIERÍA INFORMÁTICA Y COMPUTADORES
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

18 DE SEPTIEMBRE DE 2020

DEDICATORIA

A todos los caídos durante el proceso de creación de este TFG.

AGRADECIMIENTOS

Agradecemos a todas las personas que han estado presentes durante el desarrollo del proyecto por aguantarnos y ayudarnos en todo momento.

En especial gracias a Ania, Damián, Laura, Juan Bautista, Alejandro y a nuestras familias.

RESUMEN

Este Trabajo de Fin de Grado se eligió por la oportunidad de aprender más acerca de diseño hardware, ya que a lo largo de la carrera hay temas dentro de este ámbito que no se ven en profundidad. A ello hay que sumar, además, el hecho de poder diseñar una placa de circuito impreso (PCB), algo nuevo para todos los integrantes del equipo, a los que también les interesó el hecho de poder participar en un estudio donde se aplica radiación sobre una memoria SRAM (*Static Random Access Memory*) para estudiar su comportamiento.

En este TFG se aborda la temática de los efectos de la radiación natural sobre dispositivos comerciales o *Commercial-Off-The-Shelf* (COTS), la cual provoca los *Single Event Effects* (SEEs). Esta problemática es especialmente de interés en entornos como el aeroespacial. Por otro lado, en dichos entornos, reducir el consumo de energía es también importante, para lo cual se suele utilizar una técnica conocida como *Dynamic Voltage Scaling* (DVS). En trabajos relacionados, se ha demostrado que si se reduce la tensión de alimentación de ciertos componentes de un sistema (como memorias volátiles) aumenta su sensibilidad a la radiación frente a los SEE, anteriormente mencionados. Para contrarrestar este efecto, un método efectivo es realizar lecturas periódicas, o ciclos de refresco, de todo su contenido cuyo periodo debe ser calculado según el número de errores detectados durante este proceso. Esto es efectivo porque la práctica totalidad de componentes, y en particular, memorias COTS ya implementan técnicas de corrección y detección de errores (*Error Correcting Codes* o ECC). Dicho ECC, al leer una palabra de memoria que ha sufrido un error, la corrige automáticamente, lo cual puede ser utilizado para aumentar la fiabilidad de datos en sistemas y dispositivos. Hay toda una colección de técnicas ECC para detectar y corregir errores simples y múltiples (con diversa multiplicidad) en una misma palabra.

Con el fin de poder estudiar estos efectos, se ha diseñado un sistema que realiza lecturas periódicas sobre una memoria COTS. Como caso de estudio se ha elegido la memoria SRAM CY62167GE30-45ZXIES[1], la cual implementa un ECC que cuando se realiza la lectura de una palabra de memoria corrige automáticamente los errores simples, de un solo bit por palabra, pero no los errores MCU (*Multiple Cell Upsets*), de múltiples bits por palabra. Para poder ver el número de errores producidos, se ha implementado la recepción y transmisión de datos por consola con el uso de una UART (*Universal Asynchronous Receiver-Transmitter*). Dicho sistema consiste en una FPGA (que controla la operación del sistema) y una placa de expansión donde se coloca la memoria ya mencionada. Ambos elementos se comunican a través de un conector, para el cual, además, se tuvo que diseñar una PCB.

[1] En el resto del documento se abreviará con CY62167GE30

Asimismo, se ha implementado una funcionalidad que permite modificar el período de los ciclos refresco de la memoria pero de forma manual, ya que se pretendía realizar un estudio de la cantidad de errores generados a lo largo del proceso de lectura aplicando DVS (*Dynamic Voltage Scaling*) durante del proceso de radiación, pero no fue posible debido a los inconvenientes causados por el confinamiento.

El resultado final es un diseño hardware que controla las lecturas y escrituras de la memoria, los errores detectados durante la lectura y muestra por consola información relacionada con las lecturas realizadas.

Palabras clave

PCB, *hardware*, radiación natural, SRAM, *Single Event Effects* (SEEs), ECC, DVS.

ABSTRACT

This Bachelor thesis was chosen because it offered to us the opportunity to learn more about hardware design, since throughout the degree there are topics within this field that are not seen in depth. In addition, it allowed us to design a printed circuit board (PCB), which was something new for all the members of the team, who were also interested in being able to participate in a study where radiation is applied to a SRAM (Static Random Access Memory) to study its behavior.

This Bachelor thesis addresses the effects of natural radiation on commercial off-the-shelf (COTS) devices, which causes Single Event Effects (SEEs). This problem is especially interesting in environments such as aerospace. On the other hand, in such environments, reducing energy consumption is also important, for which a technique known as Dynamic Voltage Scaling (DVS) is commonly used. In related works, it has been shown that reducing the supply voltage to certain components of a system (such as volatile memories) increases their sensitivity to radiation against the previously mentioned SEEs. To counteract this effect, an effective method consists in performing periodic readings, or refresh cycles, of all its contents whose period should be calculated according to the number of errors detected during this process. This is effective because almost all components, and in particular, COTS memories, already implement error correction and detection techniques (Error Correcting Codes or ECC). This ECC, when reading a memory word that has suffered an error, corrects it automatically, which can be used to increase data reliability in systems and devices. There exists a whole plethora of ECC techniques in the literature to detect and correct single and multiple errors (with different multiplicity) in the same word.

In order to study these effects, a system has been designed that performs periodic readings on a COTS memory. As a case study, the SRAM CY62167GE30-45ZXIES memory has been chosen, which implements an ECC that, when reading a memory word, automatically corrects simple errors, of a single bit per word, but not MCUs (Multiple Cell Upsets), of multiple bits per word. In order to see the number of errors produced, data reception and transmission by console has been implemented using a UART (Universal Asynchronous Receiver-Transmitter). This system consists of an FPGA (which controls the operation of the system) and an expansion board where the already mentioned memory is placed. Both elements communicate through a connector, for which, in addition, a PCB had to be designed.

Likewise, a functionality has been implemented that allows modifying the period of the memory refresh cycles. This is controlled manually, since the intention was to carry out a study of the amount of errors generated throughout the reading process by applying DVS (Dynamic Voltage Scaling) during the radiation process. However, this was not possible due to the inconveniences caused by the confinement.

The final result is a hardware design that controls the reading and writing operations of the memory, the errors detected during the reading processes and displays relevant information about the reading cycles by console.

Keywords
Design, hardware, radiation, SRAM, errors, ECC, MCU, DVS

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Motivación

El interés por realizar un proyecto de *hardware* pero sin dejar de lado la parte de programación, así como la gran versatilidad y flexibilidad para ser usada en múltiples ámbitos diferentes que ofrece una FPGA (*Field-Programmable Gate Array)*[1], nos motivó a escoger esta como la base de nuestro proyecto.

Una vez decidido que queríamos realizar el proyecto con una FPGA decidimos escoger este tema por dos motivos: el primero es la oportunidad que nos ofrece de colaborar y formar parte de un proyecto de investigación nacional, con referencia TIN2017-87237, concedido por el Ministerio de Ciencia, Innovación y Universidades [2]. El segundo es la posibilidad de que nuestro trabajo pueda ser usado para paliar los efectos de la radiación en la exploración de entornos tan hostiles como puede ser el espacio.

1.2 Objetivos

El objetivo de este proyecto es estudiar el posible uso de memorias comerciales o COTS (*Commercial Off-The-Shelf*) en un entorno de radiación (como, por ejemplo, la aeronáutica o el sector espacio) con el objetivo de ahorrar costes ya que los componentes certificados para espacio representan costes más elevados, al no fabricarse éstos en volúmenes tan grandes como los componentes COTS. Otro motivo para utilizar estos últimos en semejantes entornos es su mayor capacidad de cálculo con respecto a componentes certificados para espacio (buenos ejemplos de ello son las misiones Exomars y GAIA de la Agencia espacial europea (ESA) [3, 4]) y que muchos de ellos ya incorporan mecanismos de detección y corrección de errores (también conocidos como *Error Correcting Codes* o ECC) que pueden ser utilizados para paliar de los efectos de la radiación [5 – 7].

Además, se desea realizarlo de manera eficiente energéticamente. Por ello se desea aplicar *Dynamic Voltage Scaling* (DVS), lo cual consiste en reducir la tensión de alimentación de ciertos componentes de un sistema (como memorias volátiles) mientras no se estén utilizando para así ahorrar energía. Sin embargo, si se reduce la tensión de alimentación, aumenta la sensibilidad frente a radiación, como ya se ha demostrado en numerosos estudios frente a varias fuentes de partículas como protones, neutrones o iones pesados [8 – 11].

Para contrarrestar este efecto, un método muy efectivo en memorias que implementan ECC es realizar lecturas periódicas de todo su contenido (a lo que, en adelante, llamaremos "ciclos de refresco"), cuyo periodo debe ser calculado según el número errores producidos en la memoria en el entorno en que está trabajando. El ECC corrige automáticamente errores simples cuando se realiza la lectura de una palabra de memoria, y además el dispositivo normalmente incluye

un pin de salida que notifica que tal acción se tuvo que realizar, por lo que es posible contar el número de errores que se corrigieron durante un ciclo de refresco. Esto permite poder ajustar la frecuencia de los ciclos de refresco, aumentándola si se detectaron muchos errores, o viceversa.

Sin embargo, durante dicho refresco se produce un aumento de consumo de energía, por lo que se debe realizar un estudio para que se realice eficazmente y no aumente excesivamente el consumo por realizar demasiados refrescos o se produzcan muchos errores por no realizar los suficientes.

Para realizar dicho estudio, hemos utilizado una memoria SRAM (*Static Random Access Memory)* [12] en concreto la memoria usada es CY62167GE30, fabricada por *Cypress Semiconductor* [13] usando como controlador la FPGA Artix-7 XC7A35T-1CPG236C [14] existente en una placa Cmod A7-35T de 48 pines [15]. Ésta realiza lecturas y escrituras en dicha memoria y mantiene el conteo de los errores detectados.

Finalmente, para facilitar la conexión física entre los pines de la placa Cmod A7-35T y la memoria CY62167GE30, mediante el software libre KiCad [16] se ha diseñado una PCB (*Printed Circuit Board*) [17], es decir, una placa de circuito impreso. Esta PCB permite establecer la conexión entre los dos componentes principales que forman parte de nuestro proyecto[2] de forma sencilla y limpia.

1.3 Plan de trabajo

La mayor parte del desarrollo e implementación del proyecto se realizó en grupo y en caso contrario normalmente mínimo había dos integrantes.

Gonzalo Fernández-Díez Ponte

Al comienzo del proyecto se realizó por separado un estudio de la documentación que nos había sido proporcionada por los tutores sobre la memoria CY62167GE30 y la placa Cmod A7-35T y realizamos individualmente un resumen y traducción, ya que toda la documentación está en inglés, de las partes más importantes para poder comenzar con la implementación. Además, se realizó un repaso de VHDL utilizando, sobre todo, los apuntes de la asignatura Tecnología de Computadores.

Tras finalizar el estudio de la documentación el equipo comenzó a reunirse en la biblioteca de la Facultad de Informática, aproximadamente 2 veces por semana en reuniones de dos horas, para comenzar con el desarrollo en VHDL de la primera versión del proyecto.

Con el comienzo de la pandemia no se pudo seguir realizando reuniones presenciales por lo que se tuvo que comenzar a realizarlas por videollamada,

[2] FPGA y memoria SRAM

normalmente 5 reuniones semanales de dos horas, en las que uno de los miembros realizaba el desarrollo en Vivado compartiendo pantalla al resto para poder colaborar, aunque todas las pruebas y depuraciones sobre la FPGA y la memoria las tenía que realizar Beatriz ya que era la única que disponía de ellas.

Durante este periodo se terminó en VHDL el desarrollo de la lectura, escritura, control de errores de la memoria y la cuenta de los ciclos que se tarda en realizar una lectura, además de la implementación de la UART para poder comunicarse con la placa y mostrar los datos obtenidos.

Una vez realizada esta parte se comenzó con el diseño de la PCB en KiCad para establecer las conexiones entre la FPGA y la memoria de forma sencilla y ordenada. Para esta parte del proyecto se siguió usando la misma metodología de trabajo. Se pudo terminar y mandar a imprimir la PCB ya en periodo de post-confinamiento.

Para el desarrollo de la memoria, ya que el proyecto se había realizado en común en su mayor parte, se decidió repartir entre los miembros las diferentes partes de la memoria por sorteo, correspondiéndole a Gonzalo los puntos: 1.1 Motivación, 2.2 FPGA Artix-7 XC7A35T-1CPG236C, 2.3 Cmod A7-35T, 3.3 Termite y 4.1 Desarrollo en Vivado, esta última parte la realizó junto con Raquel. Además, Gonzalo se encargó de la realización de la bibliografía y las referencias usando el gestor de referencias bibliográficas *Mendeley* [18].

Beatriz Villegas Sánchez

Realizó el estudio e investigación de los *datasheets* de los componentes que se iban a utilizar en el Trabajo de Fin de Grado así como de documentación adicional como artículos proporcionados por los codirectores, manuales, etc.

Durante el desarrollo del proyecto investigó cuando fue necesario sobre el lenguaje VHDL para realizar correctamente la implementación del proyecto. En general el tiempo de participación en el proyecto desde el mes de enero fue de media 2 horas 5 días a la semana.

Desarrolló con todo el equipo un módulo que simulaba la memoria física y un controlador básico funcional con los estados de lectura y escritura.

Al llegar la pandemia, Beatriz era la que tenía los componentes. Esto hizo que fuese complicado hacer un reparto equitativo de tareas, por lo que se conectaba todo el equipo para continuar con la implementación. Se buscaron los módulos de componentes necesarios como el fichero de restricciones para la herramienta de síntesis correspondiente a la placa Cmod A7-35T y Beatriz, en particular, buscó el *debouncer* que fue añadido por ella. Beatriz comprobó que pines se iban a utilizar y lo implementó, en este caso solo se usaban los botones 0 y 1 y los leds 1 y 2.

Para poder depurar la implementación del módulo de la memoria anteriormente mencionado, Beatriz asignó la funcionalidad de lectura al botón 0 y la funcionalidad de escritura al botón 1. Además, implementó que se activase el led 1 mientras se realizaba una lectura y el led 2 para la escritura, así se podría comprobar desde el exterior que la implementación del controlador funcionaba. También realizó pruebas aisladas para controlar los leds y botones de la Cmod A7-35T.

Con todo el equipo realizó la implementación del contador de ciclos de lectura, se buscó un conversor de binario a BCD, añadido por Beatriz para mostrar los ciclos a través de la UART, la cual buscó Beatriz y añadió para poder transmitir datos por consola. Se modificó en grupo para ajustarlo a las especificaciones del proyecto. Más tarde Beatriz implementó el contador de errores MCU e implementó con todo el equipo los conversores de binario a BCD para mostrar los errores ECC y MCU. Se realizaron las modificaciones pertinentes en el controlador de la memoria CY62167GE30.

Al tener casi terminada esta implementación, se depuraron los errores de funcionamiento con todo el equipo y al final Beatriz detectó que el problema de que apareciesen errores MCU cuando realmente se escribía y leía bien de la memoria CY62167GE30, era provocado por el mal contacto de los cables, los colocó de nuevo y ya funcionó. Antes de descubrir esta solución se realizaron numerosas pruebas mostrando datos por la UART para comprobar que las funcionalidades eran correctas e ir descartando posibles fallos. Durante todo este proceso en varias ocasiones Beatriz tuvo que realizar pruebas y depuraciones en solitario, ya que Vivado tardaba mucho tiempo en sintetizar, implementar y generar el *bitstream* y así se agilizaba el desarrollo del proyecto.

Una vez acabadas estas modificaciones Beatriz le dio los componentes a Raquel. Beatriz investigó en manuales y tutoriales cómo diseñar una PCB y entre todo el equipo se llevó a cabo el diseño de esta y se mandó a fabricar.

Se pidió añadir una funcionalidad nueva que permitiese establecer un período de refresco en el cual se realizarían lecturas automáticas. Beatriz y Raquel implementaron los contadores y registros de refresco y de las frecuencias, además de establecer unos períodos de refresco coherentes en la tabla de refresco.

Beatriz y Raquel investigaron cómo adaptar la UART existente para añadir la funcionalidad de recepción de datos, la modificaron añadiendo nuevos procesos y estados a la máquina de estados del módulo UART y lo depuraron ya que en un primer momento fallaba. Añadieron la funcionalidad de mostrar por consola del valor del período de refresco en formato de número de ciclos.

Por último, Beatriz y Raquel realizaron una optimización del diseño hardware, se redujeron notablemente los estados del controlador de la UART.

En cuanto al desarrollo de la memoria ha participado en la generación de la estructura de esta. Ha desarrollado en solitario los apartados 3.2 KiCad, 4.1.1.7 Debouncer, 4.2 Desarrollo en KiCad (con todos los subapartados) y ha participado en el desarrollo de los apartados 4.1.1.3 Controller, 4.1.1.4 UART, 4.1.1.5 UART Controller, 4.1.1.6 Conversor binario a BCD, 4.1.2 Desarrollo, 5 Conclusiones y trabajo futuro. Además de modificar la bibliografía y referencias y generar el índice de la memoria, el índice de las figuras y tablas y los diagramas de las máquinas de estados.

Raquel Ramos Corral

Investigó los documentos proporcionados por los tutores del Trabajo de Fin de Grado, sobre los *datasheets*, artículos de investigación, y otros documentos necesarios para el desarrollo del proyecto. También tuvo que investigar sobre el lenguaje VHDL cuando fue necesario.

Cuando todo el equipo realizó dicha investigación, se comenzó a desarrollar en conjunto el diseño de un módulo que simulaba el comportamiento de la memoria CY62167GE30 que se iba a usar, un contador que se encargaba de incrementar la dirección de memoria y una máquina de estados básica para realizar la lectura y escritura. Para depurar se creó un *testbench* con el cual poder comprobar con la simulación que ofrece Vivado los valores que tomaban las señales. Cuando aún no se había terminado de corregir esta parte comenzó el primer periodo de exámenes, debido al cual se decidió parar durante un mes el desarrollo del proyecto. Raquel fue la primera en quedarse libre, por lo que prosiguió con la depuración de lo que estaba implementado y lo terminó.

Una vez teniendo un controlador simple, se decidió empezar a usar CY62167GE30, para comprobar si realmente el controlador que se había implementado era correcto, así que Raquel buscó el fichero de pines que permitía establecer las conexiones del diseño con la placa.

Poco después comenzó la pandemia. Al solo disponer de una Cmod A7-35T y una CY62167GE30 pasamos a hacer 5 reuniones a la semana de 2 horas de duración aproximadamente, ya que era Beatriz quien las tenía.

Para comprobar que el diseño implementado anteriormente funcionaba con la CY62167GE30 se usaron los leds y los botones para la depuración.

Más tarde se implementó un contador de ciclos, pero para poder mostrar su valor se llegó al acuerdo de implementar una UART. Una vez añadida se modificó la máquina de estados que gestiona el envío de datos a la UART para adaptarla al proyecto.

Para que los datos se interpretaran por consola con el formato adecuado, se buscó un conversor a BCD para la cuenta de los ciclos. Todo el equipo participó

en la depuración necesaria para que se mostraran correctamente los datos de los ciclos.

Una vez ya se tuvo esto funcionando se repartieron las tareas para implementar la gestión de errores. Raquel se encargó de implementar el contador de los errores que llegaban de la salida ERR de la memoria y ayudó a Gonzalo con la máquina de estados que gestiona el envío de datos a la UART, a la cual había que añadir los estados necesarios para el mostrar el número de errores.

Al implementar esto último, se comprobó que el número de errores MCU era muy elevado, por lo que se comenzó un proceso de depuración que llevó casi un mes. En este proceso, con el uso de la UART se mostraron los datos que se creyeron convenientes para la detección del error.

Tras el fin del confinamiento Beatriz entregó la placa a Raquel.

Al acabar la implementación de la primera parte del diseño se comenzó con el diseño de la PCB. Para ello, Raquel tuvo que investigar el uso de KiCad. Cuando el equipo terminó con el estudio se diseñó la placa en conjunto.

Cuando se mandó la placa a imprimir, Raquel y Beatriz comenzaron a añadir una nueva funcionalidad que permitiese establecer un periodo de refresco en el cual se realizarían lecturas automáticas. Para ello implementaron dos contadores y un registro además de implementar una tabla con los periodos de refresco.

Se realizó un estudio para añadir la funcionalidad de recepción de datos por la UART. Una vez se hubo cambiado la UART, se comenzó con la depuración ya que no se interpretaba bien el dato que recibía de la consola. Debido al poco tiempo disponible para terminar esta parte, Raquel terminó la depuración de la UART e hizo la depuración de los tres componentes anteriormente implementados. Se decidió mostrar el periodo de refresco por consola para que el usuario supiera el periodo actual de refresco, por lo que Raquel y Beatriz hicieron los cambios necesarios en el controlador de la UART.

Raquel movió la máquina de estados que se encargaba de la gestión de los datos que se le envían a la UART a un componente nuevo para una mayor organización. Raquel y Beatriz revisaron el diseño en busca de posibles mejoras y optimizaron el módulo del controlador de la UART. Por último, Raquel se encargó de hacer los cambios sugeridos por los tutores para dejar la implementación del diseño completamente acabada.

Para el desarrollo de la memoria, Raquel se ha encargado de los puntos 1.3 Plan de trabajo, 2.1 Memoria SRAM CY62167GE30-45ZXIES, 3.1 Vivado y 4.1 Desarrollo en Vivado y 5 Conclusiones y trabajo futuro. Además, ha ayudado con el punto 1.2 Objetivos. Por último, se ha encargado de generar los diagramas de bloques.

Capítulo 2 - Elementos hardware utilizados

En este capítulo se va a hablar de los componentes hardware utilizados durante el proceso de desarrollo y ejecución del proyecto, y de ciertas características relevantes de cada uno. Estos componentes son:

- **SRAM CY62167GE30-45ZXIES** Se trata de una memoria de 2MB de capacidad, que se puede configurar, a través de la entrada BYTE, para funcionar en modo 1 M palabras x 16 bits (BYTE a 1), o 2 M palabras x 8 bits (BYTE a 0). También se pueden habilitar los bytes superior e inferior de las palabras estableciendo las señales BLE (*Byte Low Enable*), BHE (*Byte High Enable*) a 0. Para poder interactuar con dicha memoria se reutilizó una PCB ya diseñada utilizada en trabajos anteriores llevados a cabo por el grupo de investigación [19-21].

  Esta PCB consta de un conector de 36 pines mostrado en la Figura 2.1 y tiene modificadas las conexiones de tal forma que las señales BLE, BHE y BYTE estén fijadas a tierra y por lo tanto los bytes superior e inferior de las palabras están habilitados y la configuración es de 2 M palabras x 8 bits, como se puede apreciar en la Figura 2.2. Esto permite tener más pines disponibles en caso de ser necesarios para otra implementación y dejar fijada ya la configuración deseada de la memoria.

  Esta PCB se conecta a placa Cmod A7-35T a través de un cable plano de varios metros de longitud. Esto es necesario ya que la PCB va a ser irradiada y la FPGA debe estar lo suficientemente lejos de la fuente de irradiación para que los errores observados no sean atribuibles a la FPGA, sino únicamente a la memoria.

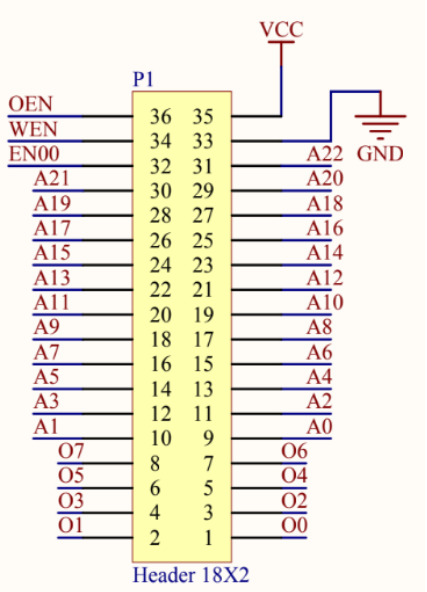

Figura 2.1 *Conector macho de 36 pines que permite la conexión de una FPGA con la memoria CY62167GE30*

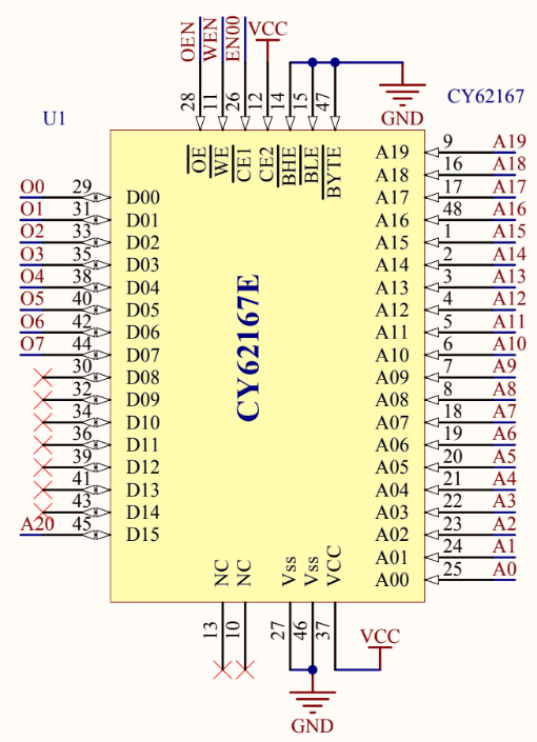

Figura 2.2 *Diseño esquemático de la memoria CY62167GE30 y sus conexiones con el conector de la PCB*

En la Figura 2.3 se puede ver el diseño completo de la PCB.

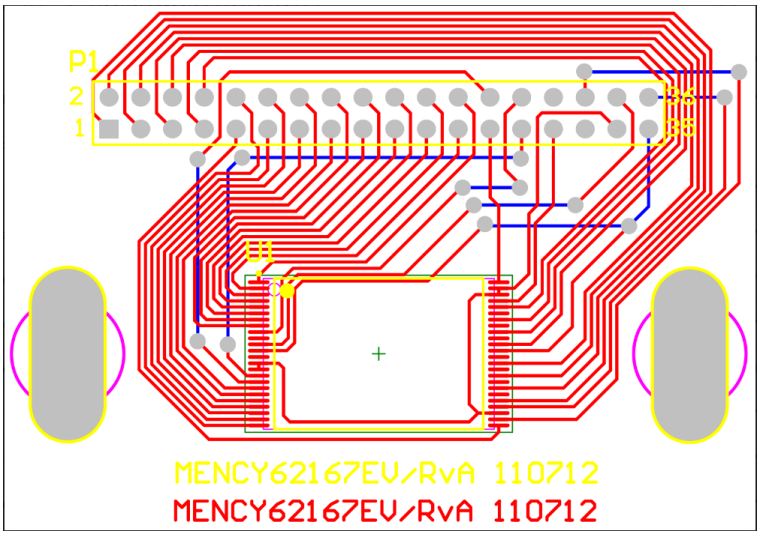

Figura 2.3 *Diseño de la PCB que contiene a la memoria CY62167GE30*

- **FPGA Artix-7 XC7A35T-1CPG236C** de Xilinx, la cual está integrada en una pequeña tarjeta de desarrollo DIP, apta para su inserción en las placas de prototipado y fabricada por Digilent, Cmod A7-35T.

La Figura 2.4 muestra la forma en la que se han realizado las conexiones entre la placa XC7A35T y la memoria CY62167GE30.

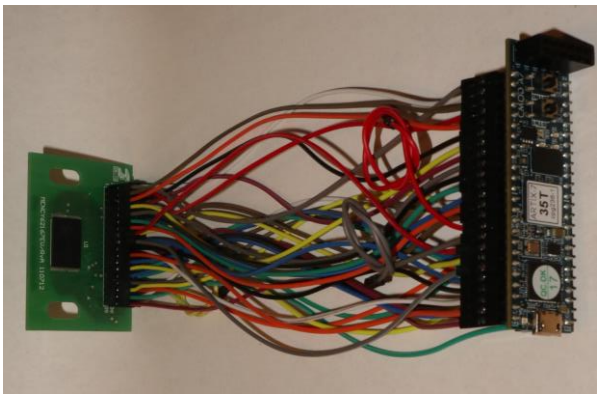

Figura 2.4 *Conexiones entre XC7A35T y CY62167GE30*

2.1 Memoria SRAM CY62167GE30-45ZXIES

La CY62167GE30 es una SRAM CMOS (*Complementary Metal Oxide Semiconductor*) de alto rendimiento y baja potencia con ECC incorporado. Dispone de un pin ERR que advierte de un error de un bit que ha sido corregido durante el proceso de lectura de una determinada palabra. En la Figura 2.5 se muestra cómo están organizados los pines de esta memoria.

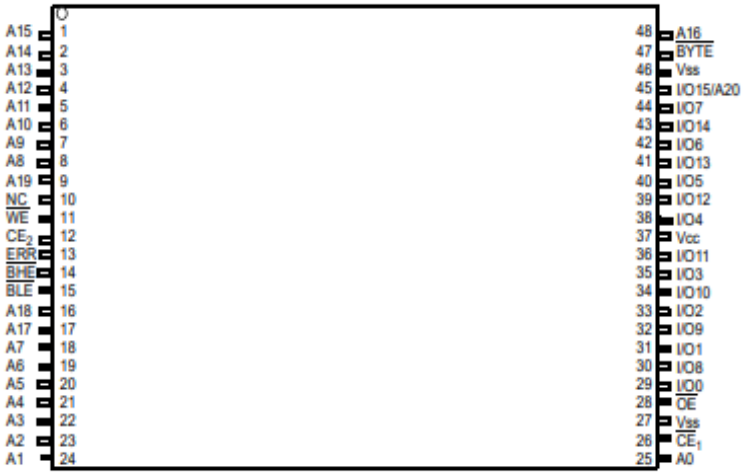

Figura 2.5 *Chip que contiene la memoria SRAM CY62167GE30, https://www.cypress.com/file/139521/download*

El dispositivo se puede configurar para funcionar en modo 1 M palabras x 16 bits (poniendo la entrada BYTE a 1) o 2 M palabras x 8 bits (poniendo la entrada BYTE a 0). En este caso, la configuración utilizada es esta última para así reducir el número de pines del bus de datos para acceder a la memoria. Para utilizar el dispositivo como chip único, se habilita la entrada $\overline{CE}$ y para utilizarlo como doble, se habilitan $\overline{CE_1}$ y CE_2.

Las entradas BHE y BLE controlan si se puede escribir en las líneas I/O correspondientes. $\overline{BHE}$ controla I/O8 – I/O15 y $\overline{BLE}$ controla I/O0 – I/O7. En modo 2 M palabras x 8 bits, los pines I/O8 – I/O15 no se utilizan.

Los pines I/O0 – I/O15 se establecen en un estado de alta impedancia cuando el dispositivo no está seleccionado o las señales $\overline{OE}$, $\overline{BHE}$ y $\overline{BLE}$ están desactivadas. Si las señales $\overline{BHE}$y $\overline{BLE}$ se deshabilitan, el dispositivo cambia a un estado de espera independientemente del estado del chip, lo que le permite un ahorro de energía.

Para poder realizar una escritura, se debe activar la señal $\overline{WE}$ y se tienen que proporcionar los datos a almacenar en los pines de datos del dispositivo (I/O0 – I/O7 o I/O0 – I/O15, dependiendo en qué modo se encuentre). Asimismo, se debe de disponer de la dirección en la cual se va a almacenar en los pines de dirección (A0 – A19 o A0 – A20, según su modo de funcionamiento).

Para poder realizar un proceso de lectura se establece la habilitación de salida $\overline{OE}$, y se proporciona la dirección de la cual se va a leer. En la Figura 2.6 se muestra la tabla de verdad del dispositivo.

Truth Table – CY62167G/CY62167GE

BYTE [46]	$\overline{CE_1}$	CE_2	WE	OE	BHE	BLE	Inputs/Outputs	Mode	Power	Configuration
X[47]	H	X[47]	X	X	X	X	High-Z	Deselect/Power-down	Standby (I_{SB})	2M × 8/1M × 16
X	X[47]	L	X	X	X	X	High-Z	Deselect/Power-down	Standby (I_{SB})	2M × 8/1M × 16
X	X[47]	X[47]	X	X	H	H	High-Z	Deselect/Power-down	Standby (I_{SB})	1M × 16
H	L	H	H	L	L	L	Data Out (I/O_0–I/O_{15})	Read	Active (I_{CC})	1M × 16
H	L	H	H	L	H	L	Data Out (I/O_0–I/O_7); High-Z (I/O_8–I/O_{15})	Read	Active (I_{CC})	1M × 16
H	L	H	H	L	L	H	High Z (I/O_0–I/O_7); Data Out (I/O_8–I/O_{15})	Read	Active (I_{CC})	1M × 16
H	L	H	H	H	L	H	High-Z	Output disabled	Active (I_{CC})	1M × 16
H	L	H	H	H	H	L	High-Z	Output disabled	Active (I_{CC})	1M × 16
H	L	H	H	H	L	L	High-Z	Output disabled	Active (I_{CC})	1M × 16
H	L	H	L	X	L	L	Data In (I/O_0–I/O_{15})	Write	Active (I_{CC})	1M × 16
H	L	H	L	X	H	L	Data In (I/O_0–I/O_7); High-Z (I/O_8–I/O_{15})	Write	Active (I_{CC})	1M × 16
H	L	H	L	X	L	H	High-Z (I/O_0–I/O_7); Data In (I/O_8–I/O_{15})	Write	Active (I_{CC})	1M × 16
L	L	H	H	L	X	X	Data Out (I/O_0–I/O_7)	Read	Active (I_{CC})	2M × 8
L	L	H	H	H	X	X	High-Z	Output disabled	Active (I_{CC})	2M × 8
L	L	H	L	X	X	X	Data In (I/O_0–I/O_7)	Write	Active (I_{CC})	2M × 8

ERR Output – CY62167GE

Output[48]	Mode
0	Read operation, no single-bit error in the stored data.
1	Read operation, single-bit error detected and corrected.
High-Z	Device deselected / outputs disabled / Write operation

Figura 2.6 *Tabla de verdad de la memoria SRAM CY62167GE30 y del pin de ERR,* https://www.cypress.com/file/139521/download

Cuando, al realizar la lectura de una palabra (8 o 16 bits, según la configuración), la memoria detecta un error de un solo bit en dicha palabra, lo corrige automáticamente. Además, lo indica poniendo a 1 la salida ERR (por tanto, si ERR se mantiene a 0, no hubo ningún error de lectura en esa palabra).

Las características de la memoria son:

- Corriente media en *standby* de 5.5µA
- Corriente máxima en *standby* de 16µA
- Alta velocidad, ciclo de reloj de 45ns para operaciones de lectura y escritura
- *Error-Correcting Code* (ECC) para la detección de errores de un solo bit

- Rango de voltaje: 4.5V a 5.5V
- Retención de datos garantizada con una tensión de alimentación de 1.0V
- Pin para indicar la detección y corrección de errores (ERR)
- 48 pines TSOP (*Thin Small Outline Package*) configurables como 1 M x 16 o 2 M x 8 bits

El diagrama de bloques de la memoria CY62167GE30 se muestra en la Figura 2.7.

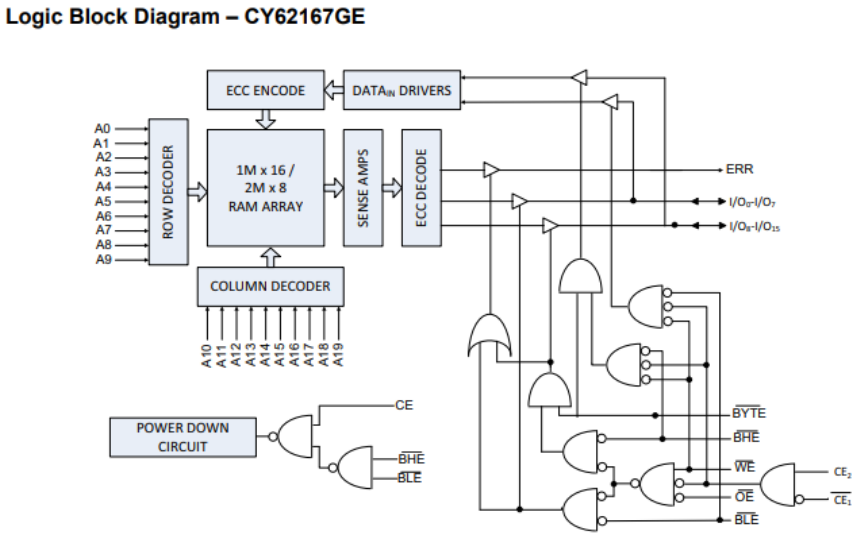

Figura 2.7 *Diagrama de bloques lógicos de la memoria SRAM CY62167GE30,* https://www.cypress.com/file/139521/download

2.2 FPGA Artix-7 XC7A35T-1CPG236C

Una FPGA es un dispositivo semiconductor formado a partir de una matriz de bloques lógicos y bloques RAM conectados a través de interconexiones programables. Su configuración puede ser programada y reprogramada después de su fabricación usando un lenguaje de descripción de *hardware* o HDL (*Hardware Description Language*) [22], en nuestro caso VHDL [23], lo que lo convierte en un dispositivo ideal para muchos proyectos e industrias.

Se diferencian de dispositivos similares como los ASIC (*Application-specific Integrated Circuit*), en su capacidad de reconfiguración, o los CPLD (*Complex Programmable Logic Device*) [24], en que, aunque también son reprogramables, estos últimos tienen menos potencia de cálculo y menor densidad de elementos lógicos programables. Además, las FPGA incluyen hardware dedicado como bloques de RAM, bloques DSP (*Digital Signal Processors*) [25] para el procesamiento

a alta velocidad de señales digitales, PLL (*Phase Lock Loop*) [26] para el control de señales, DCM (*Digital Clock Manager*) [27] para manipular las señales de reloj internas de la FPGA, controladores de memoria, transceptores de alta velocidad e interfaces de entrada/salida. Aun así, los CPLD tienen ciertas ventajas como su coste reducido o que son no volátiles[3] como las FPGA.

En este proyecto se va a usar la FPGA XC7A35T-1CPG236C de la familia Artix-7 [28] de Xilinx. Algunas de sus características [29, 30] destacables son:

- Número de *Logic Cells*: 32.280
- Número de *Logic Slices*: 5.200
- Número de *Flip-flops*: 29.200
- *Block* RAM (Kbits): 1.800
- *Clock Management Tiles*: 5
- DSPs *Slices*: 90
- Transceptores de 6,6Gb/s que permiten un pico de ancho de banda de 211 Gb/s (*full-duplex*)
- Estándares de E/S simples y dobles con velocidades de hasta 1,25 Gb/s
- Memoria DDR3 con velocidades de 1.066Mb/s y soporte para SODIMMs

2.3 Cmod A7-35T

En el proyecto se ha utilizado la FPGA que se encuentra montada sobre una placa Cmod A7-35T. Se trata de una pequeña placa de 48 pines DIP (*Dual In-line Package*) con dos líneas paralelas de 24 pines cada una, como se puede ver en la Figura 2.8.

Figura 2.8 *Vista inferior de la placa Cmod A7-35T,*
https://reference.digilentinc.com/reference/programmable-logic/cmod-a7/start

[3] No pierden su configuración cuando se desconecta la energía

También incluye un circuito de programación USB-JTAG, un adaptador USB-UART, una fuente de reloj de 12 MHZ, un conector Pmod (*Peripheral Module*), una memoria SRAM de 512 KB, una memoria no volátil Quad-SPI (*Serial Peripheral Interface*) Flash, y otros componentes básicos de E/S: un led RGB, 2 leds individuales y 2 botones, mostrados en la Figura 2.9. En la Figura 2.10 se puede observar un diagrama de las conexiones de dichos componentes con la placa Artix-7.

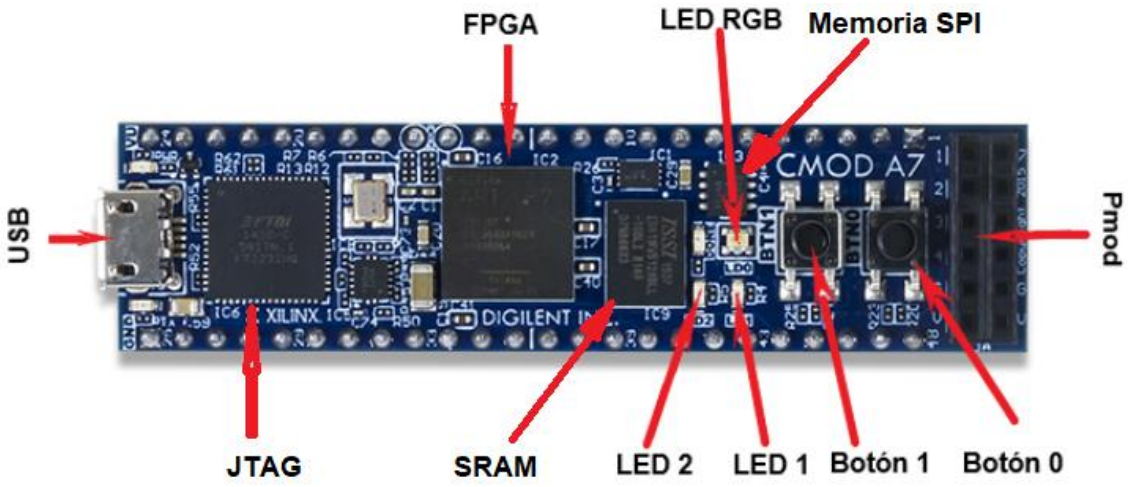

Figura 2.9 *Vista superior de la placa Cmod A7-35T,*
https://reference.digilentinc.com/reference/programmable-logic/cmod-a7/start

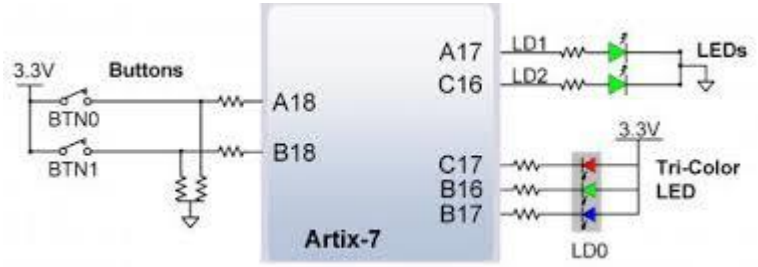

Figura 2.10 *Diagrama de los componentes de los botones y leds que contiene la FPGA,*
https://reference.digilentinc.com/reference/programmable-logic/cmod-a7/reference-manual

La FPGA contenida en la placa se puede configurar de dos maneras: a través de un circuito USB-JTAG o con un archivo de configuración contenido en la memoria no volátil Quad-SPI Flash que puede ser transferido a la FPGA a través del puerto SPI. Si no se encuentra ningún archivo de configuración en la memoria Flash, la FPGA se mantiene desconfigurada hasta que sea programada a través de la interfaz JTAG.

Los datos de la configuración de la FPGA se guardan en archivos llamados *bitstreams* [29], los cuales tienen extensión .bit. Estos *bitstreams* se almacenan en la memoria volátil de configuración de la FPGA y definen sus funciones lógicas y conexiones del circuito. Estos datos siguen siendo válidos hasta que se borran apagando el dispositivo o escribiendo un nuevo archivo de configuración.

Capítulo 3 - Tecnologías utilizadas

En este capítulo se va a hablar de la parte técnica de las herramientas[4] software que se han utilizado para el desarrollo del proyecto.

3.1 Vivado

Vivado [31] es una herramienta cuya principal funcionalidad es el desarrollo de hardware. Permite diseñar circuitos en hardware reconfigurable utilizando lenguajes de programación como VHDL o Verilog [32].

3.1.1 Project manager

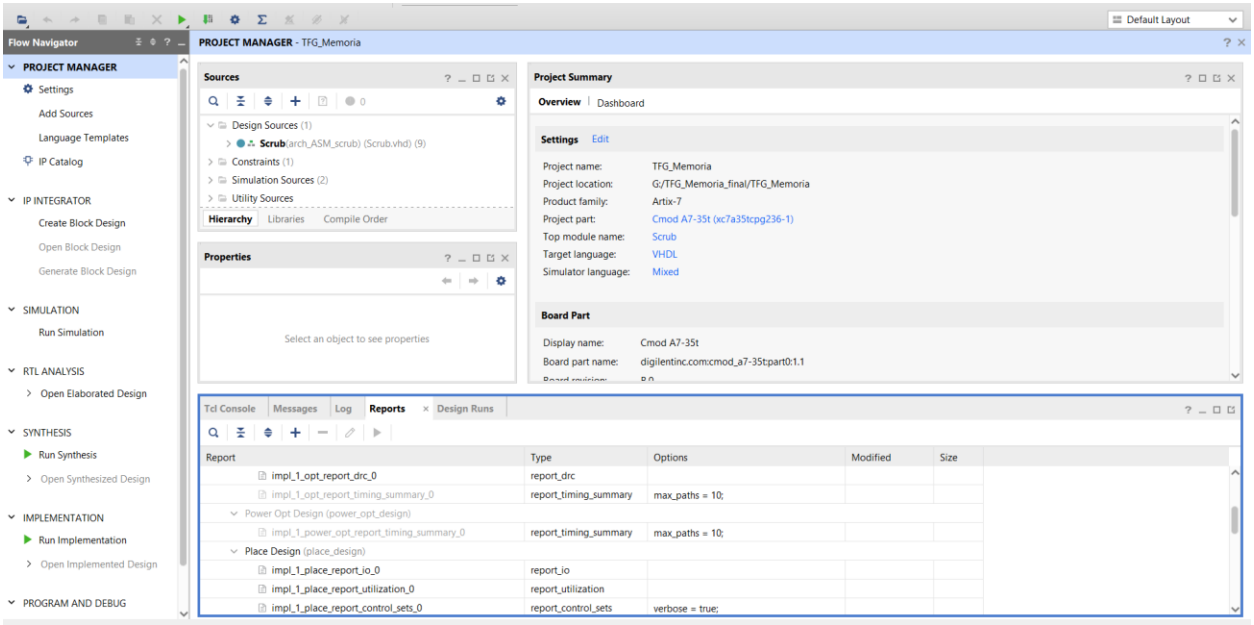

Figura 3.1 *Pantalla principal de Vivado*

En la Figura 3.1 se puede observar la organización de las ventanas por defecto de Vivado. Esta es la pantalla por defecto, donde se pueden observar los siguientes paneles:

- **Text Editor**

  Entorno para la edición de archivos. Reconoce la sintaxis del formato del archivo, resaltando las palabras clave e indicando errores de sintaxis.

[4] La información sobre estas herramientas se ha obtenido de las ventanas de ayuda del programa Vivado.

- **Flow Navigator**

Proporciona herramientas para el diseño del proyecto, y está dividido en siete secciones:

  - *Project manager*: en esta sección se puede cambiar la configuración del proyecto, agregar archivos, cambiar el idioma y ver el catálogo de IP Cores de Vivado.

  - *IP integrator*: crea, abre o genera un diseño de bloque.

  - *Simulation*: lanza la simulación del diseño.

  - RTL *analysis* (*Register Transfer Level*) [33]: genera un esquema RTL mediante DRC (*Design Rule Checks*). El esquema RTL es una vista gráfica de los componentes del diseño que muestra cómo están conectados entre sí. Estos pueden ser: registros, sumadores, multiplicadores, etc. Un esquema generado con esta herramienta es la Figura 4.1

  - *Synthesis*: permite sintetizar el diseño o abrir el diseño sintetizado, si ya se ha hecho una síntesis previamente.

  - *Implementation*: permite implementar el diseño o abrir el diseño implementado, si es que anteriormente esto ya hizo.

  - *Program and debug*: genera un archivo *bitstream* o abre el *hardware manager* si ya se ha generado este archivo.

- **Sources**

Proporciona diferentes vistas de los archivos del proyecto. Las vistas son las siguientes:

  - *Hierarchy*: muestra los módulos ordenados según se contenga en el diseño.

  - *Libraries*: muestra los archivos ordenados por bibliotecas.

  - *Compile order*: muestra los módulos ordenados en el orden de compilación. Hay tres órdenes distintas: síntesis, implementación y simulación.

- **Results windows area**

Es un conjunto de ventanas agrupadas situadas en la parte inferior de la Figura 3.1, que se compone de:

- TCL *Console (Tool Command Language)*: permite escribir comandos y ver los resultados de su ejecución.

- *Messages*: muestra los mensajes generados por el proyecto, organizados por gravedad. También da la opción de solo mostrar los tipos de mensajes que se elijan.

- *Log*: muestra los archivos de registro creados por la simulación, la implementación o la síntesis.

- *Reports*: proporciona acceso rápido a los informes de síntesis e implementación en el orden del flujo de diseño.

- *Design runs*: gestiona las ejecuciones del proyecto.

3.1.2 Hardware manager

En la Figura 3.2 se pueden observar las ventanas que contiene el *hardware manager*:

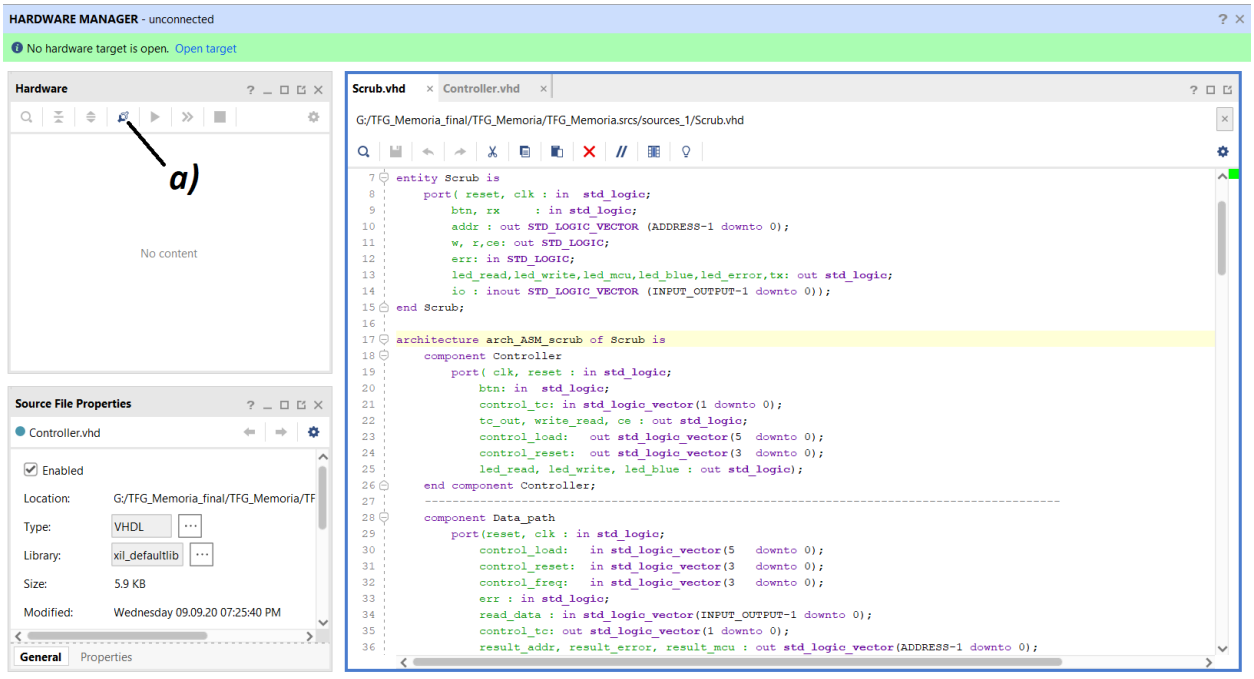

Figura 3.2 *Ventana Hardware manager*

- **Hardware**

  Muestra la vista del servidor *hardware* y el dispositivo conectado, en este caso, la FPGA XC7A35T. La barra de herramientas tiene comandos útiles como el mostrado en la Figura 3.2, etiqueta a, que sirve para conectarse automáticamente con un servidor *hardware* que se encuentre en el *host* local.

- **Hardware target properties**

  Muestra la información general del dispositivo seleccionado previamente en la ventana hardware.

3.1.3 Simulation

En la Figura 3.3 se pueden observar las ventanas que contiene el simulador que viene integrado con la herramienta Vivado:

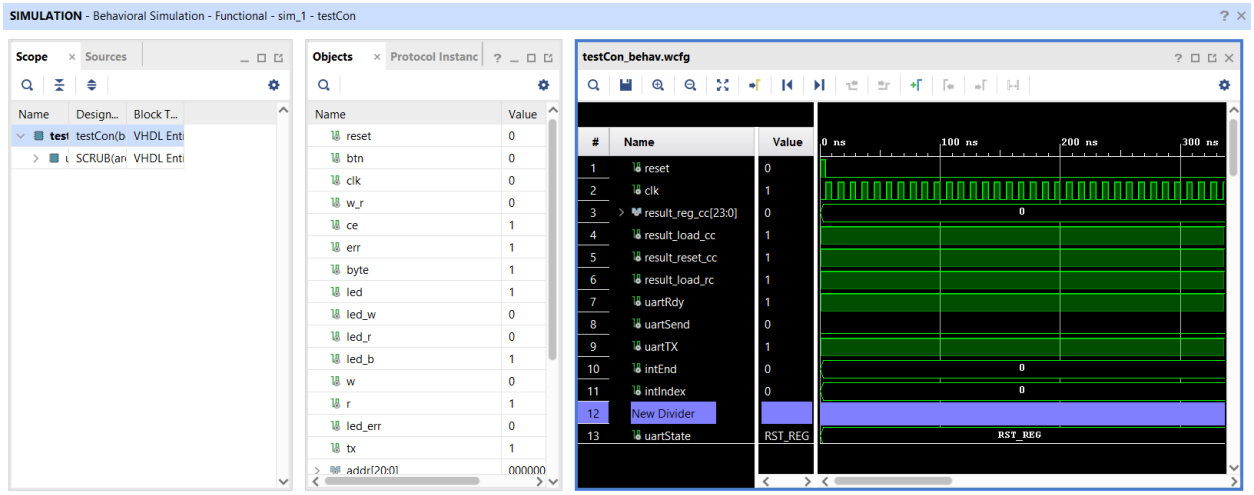

Figura 3.3 *Ventana Simulation*

- **Waveform window**

  Esta ventana muestra la simulación del *testbench*, test implementado para comprobar el funcionamiento del proyecto. En ella se ven las señales y registros como objetos HDL.

  En principio solo van a aparecer los objetos HDL que se encuentren definidos en el test, pero se pueden añadir también los objetos HDL que contengan el resto de componentes desde las ventanas *scope* u *objects*.

  Esta ventana se divide en dos regiones:

- La región izquierda contiene los nombres y los valores de los objetos HDL.

- La región derecha contiene la línea de tiempo de la simulación.

La barra de herramientas proporciona unos botones cuya funcionalidad permite expandir o contraer la línea de tiempo de la ejecución de la simulación. Cuenta con otra barra de herramientas, mostrada en la Figura 3.4, que ayuda a la depuración permitiendo ejecutar línea a línea el diseño, o avanzar un periodo de tiempo determinado la línea temporal.

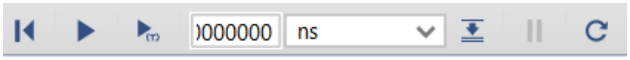

Figura 3.4 *Barra de herramientas para la depuración de la simulación*

- **Objects**

Esta ventana muestra los objetos HDL definidos en el componente que se seleccione desde la ventana *scope*. Los iconos que se muestran al lado del objeto HDL indican qué tipo de objeto es. Esta ventana permite gestionar los objetos que se muestran en la ventana *waveform window*.

- **Scope**

En esta venta se muestran los componentes ordenados de manera jerárquica, pudiendo seleccionar uno de ellos para ver sus objetos HDL en la ventana *objects*.

3.2 KiCad

KiCad [34, 35] es una herramienta de software libre y proyecto del profesor e investigador Jean-Pierre Charras que permite la creación de las PCB. Cuenta con un software para realizar el diseño esquemático de una PCB, *Schematic Layout Editor*, el cual permite incluir los símbolos de los componentes, añadir las conexiones entre ellos de diferentes formas y asociar sus huellas[5] a los símbolos con la funcionalidad *Asignación de huellas* (Figura 3.5, etiqueta b). Además, en caso de no existir el símbolo del componente y/o su huella, KiCad permite crearlas desde cero.

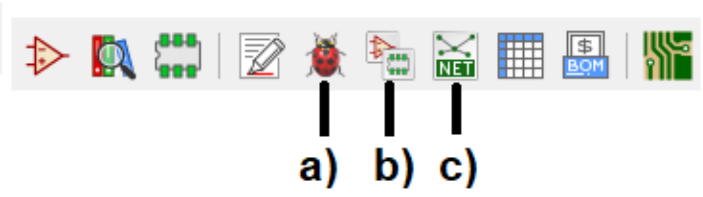

Figura 3.5 *Barra de herramientas de Schematic Layout Editor*

[5] Patrón del componente electrónico que va a ser soldado

En cuanto a la funcionalidad de asignación de huellas, cabe destacar que dispone de varios filtros que facilitan la búsqueda de las huellas de los componentes, como se puede observar en la Figura 3.6. En la sección izquierda de la ventana aparecen todas las librerías de huellas disponibles. La funcionalidad permite añadir librerías adicionales en caso de necesitar alguna específica; mientras que en la sección central se realizan las asignaciones de huellas a los símbolos y en la sección derecha aparecen todos los componentes filtrados disponibles de la librería seleccionada en la sección izquierda.

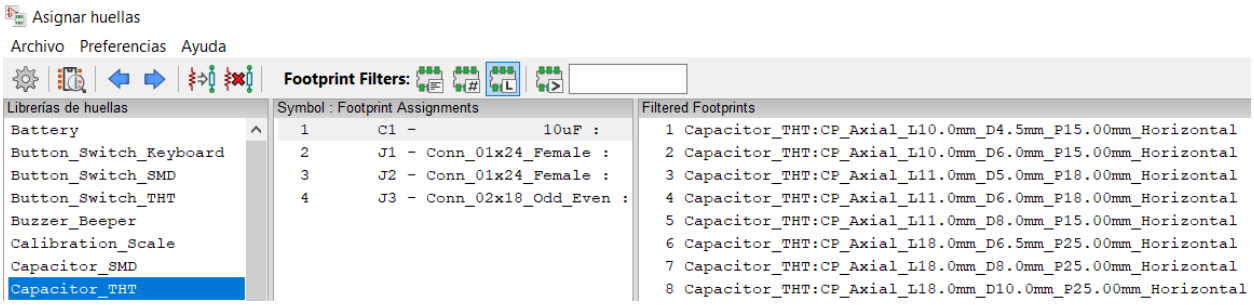

Figura 3.6 *Herramienta de asignación de huellas*

Durante todo el proceso de diseño tanto del esquemático como de la PCB, KiCad ofrece una funcionalidad, *Realiza un control de las reglas eléctricas* (Figura 3.5.a), que realiza la depuración del diseño. Esto muestra al usuario errores de conexiones, componentes mal definidos y en general errores no triviales.

Para el diseño del *footprint* o huella de la PCB, KiCad ofrece la herramienta *PCB Layout Editor* (Figura 3.7, etiqueta b), que permite generar la placa de circuito impreso a partir del esquemático anteriormente generado. Esto facilita el diseño de la PCB ya que, con la funcionalidad *Update PCB from esquematic*[6] (Figura 3.8, etiqueta a) sólo hay que añadir las pistas, colocar las huellas y añadir detalles a la placa. Esto es posible siempre que se haya generado la lista de redes y se hayan asignado las huellas a los componentes en el esquemático. El listado de redes anteriormente mencionado se genera con la funcionalidad *Generar listado de redes* (Figura 3.5, etiqueta c). Este listado es necesario para que al cargar los datos del esquemático en la herramienta *PCB Layout Editor* se muestren las huellas con sus conexiones.

[6] Permite cargar las huellas de los componentes asociados a los símbolos del esquemático con todas sus conexiones

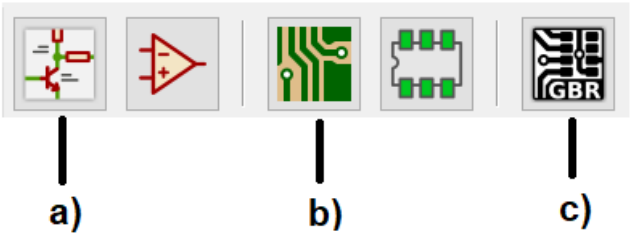

Figura 3.7 Barra de herramientas principal de KiCad

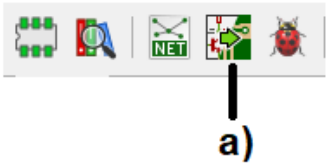

Figura 3.8 *Barra de herramientas de PCB Layout Editor*

Una vez completado el diseño (tanto el esquemático como del *footprint* de la PCB), se generan los ficheros Gerber desde la herramienta *PCB Layout Editor*. Dispone de una funcionalidad *Trazar* (Figura 3.9, etiqueta a), que permite generar los ficheros necesarios para la creación de la PCB en varios formatos como HPGL, PostScript o Gerber. En este proyecto se usará el formato Gerber, que es el estándar utilizado en la industria. Los ficheros Gerber son archivos generados a partir de las capas de la PCB que se envían al fabricante elegido para que construya la placa.

Figura 3.9 *Barra de herramientas de PCB Layout Editor*

KiCad dispone de otra funcionalidad que permite visualizar lo generado por los ficheros Gerber con la funcionalidad *GerberViewer* (Figura 3.7, etiqueta c).

3.3 Termite

Termite [36] es un terminal de transmisión de texto disponible en varios idiomas que usa el estándar RS-232 (*Recommended Standard 232*) [37], que es un protocolo de comunicación en serie (transmite datos bit a bit), creado en 1960 por la *Electronic Industries Alliance*, para conectar pares de dispositivos.

Está diseñado para tener un uso y configuración simples y, como se puede observar en la Figura 3.10, una interfaz sencilla con una barra de introducción de texto y una ventana en la que se muestran en color verde los datos transmitidos y en azul los recibidos (de hecho, en la Figura 3.10 se muestra un ejemplo de

funcionamiento del programa). Termite incluye un historial de los comandos utilizados y la opción de autocompletarlos, así como las opciones de copiar, guardar en un documento de texto, imprimir, limpiar o realizar una búsqueda de los datos del terminal y comandos de teclado para facilitar su uso.

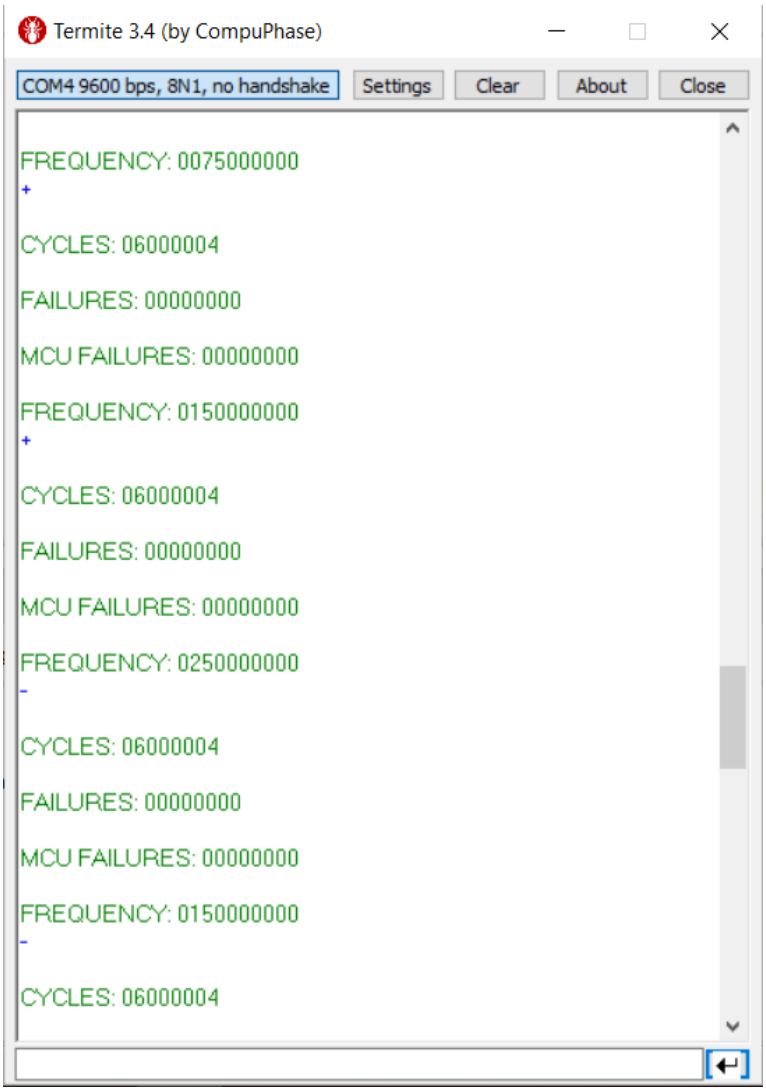

Figura 3.10 Ventana Termite con varias ejecuciones del programa

Termite permite establecer una amplia configuración, destacando la selección del puerto que incluye una búsqueda automática de los que se encuentran en uso, la tasa de baudios, el número de bits del dato transmitido, el número de bits de parada y la paridad además de múltiples opciones gráficas y funciones extra del terminal.

En la Figura 3.11 se puede observar la configuración usada en nuestro proyecto: una tasa de baudios de 9600, con un tamaño de datos de 8 bits, sin paridad, con 1 bit de parada.

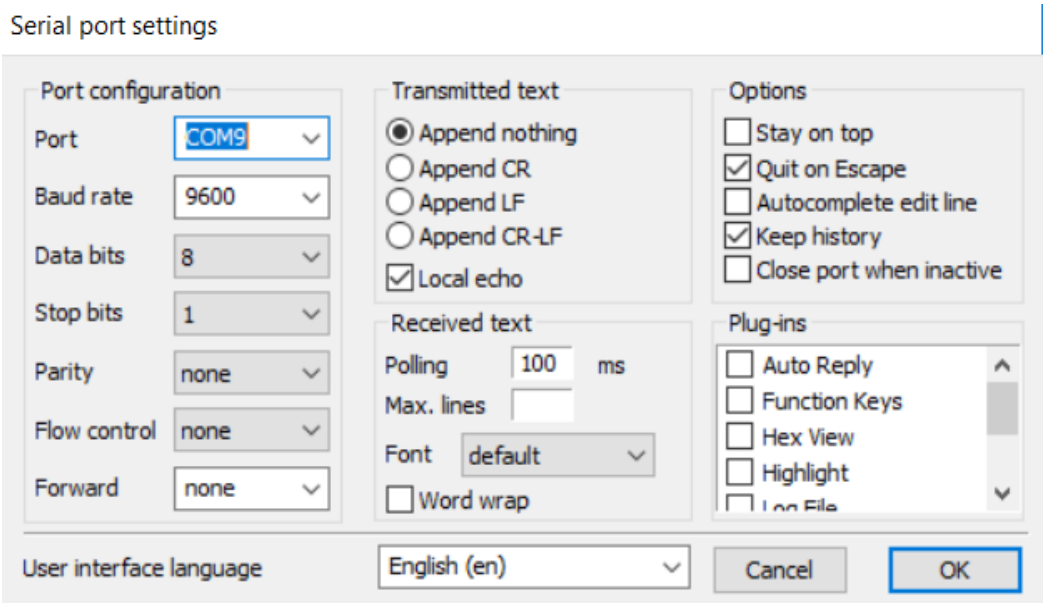

Figura 3.11 *Configuración de Termite*

Capítulo 4 - Desarrollo del proyecto

En este capítulo se va a explicar el proceso seguido para el desarrollo de las diferentes partes del proyecto.

En este proyecto se ha implementado un sistema de escritura y de ciclos de refresco en la memoria CY62167GE30. Dado que no se ha podido implementar un sistema que calcule una tasa de refresco automático a partir de los errores generados por la memoria, debido a que ésta no se pudo someter a radiación, se ha implementado una funcionalidad de refresco que consiste en que el usuario a través de la UART indique con un símbolo + o un símbolo -, si se debe aumentar o disminuir, respectivamente, el periodo de refresco. Todos estos periodos se encuentran predefinidos ("*hard-coded*") en una tabla.

Además, a través del puerto serie también se muestran los ciclos que tarda en realizarse una lectura, el número de errores corregidos automáticamente por el ECC y los errores que el ECC no pudo corregir. A éstos se les ha llamado "Errores MCU", donde MCU significa *Multiple Cell Upset* [38,39]. Estos son errores en los que fallan múltiples bits en una sola palabra de la memoria y que el ECC (que es de tipo *Single Error Correction – Double Error Detection* o SEC-DED) [40] no es capaz de corregir. La detección de estos ha servido para depurar, ya que permite comprobar fácilmente que se escribía y leía correctamente de la memoria.

A su vez, se ha diseñado una PCB que, junto con un cable de interconexión de 36 pines, funciona como enlace entre la placa XC7A35T y la PCB que contiene la memoria CY62167GE30.

4.1 Desarrollo en Vivado

El proceso de desarrollo en Vivado ha comprendido la mayor parte del proyecto. En ella se ha realizado el diseño e implementación de los componentes del módulo realiza el proceso de lectura y escritura en la memoria SRAM.

4.1.1 Componentes

Para que sea más eficiente el desarrollo de los componentes, se ha creado un fichero *Definitions*, que contiene todas las constantes utilizadas durante el desarrollo del código. Dentro de estas constantes se encuentran los tamaños de los vectores, los datos que se muestran por consola, y la tabla de periodos mostrada en la Tabla 4.1.

En todos los componentes que forman la ruta de datos (*Datapath*) y el controlador de la memoria (*Controller*) se ha introducido un *reset* asíncrono activado por el usuario desde el botón 1 (ver Figura 2.9) para poder reiniciar desde el exterior los valores de dichos componentes.

Índice de la tabla	*Ciclos entre refrescos*	*Período (Segundos)*
0	37.000.000	3.08
1	75.000.000	6.25
2	150.000.000	12.5
3	250.000.000	20.8
4	500.000.000	41.6
5	750.000.000	62.5
6	1.000.000.000	83.4
7	1.150.000.000	95.8

Tabla 4.1 *Periodos de refresco*

4.1.1.1 Scrub

Es el módulo principal del proyecto, donde se instancian los componentes *Datapath*, *Controller*, UART, *Controller* UART, *Debouncer* y BCD y éstos se conectan entre sí. Se puede observar el diagrama de bloques en la Figura 4.1:

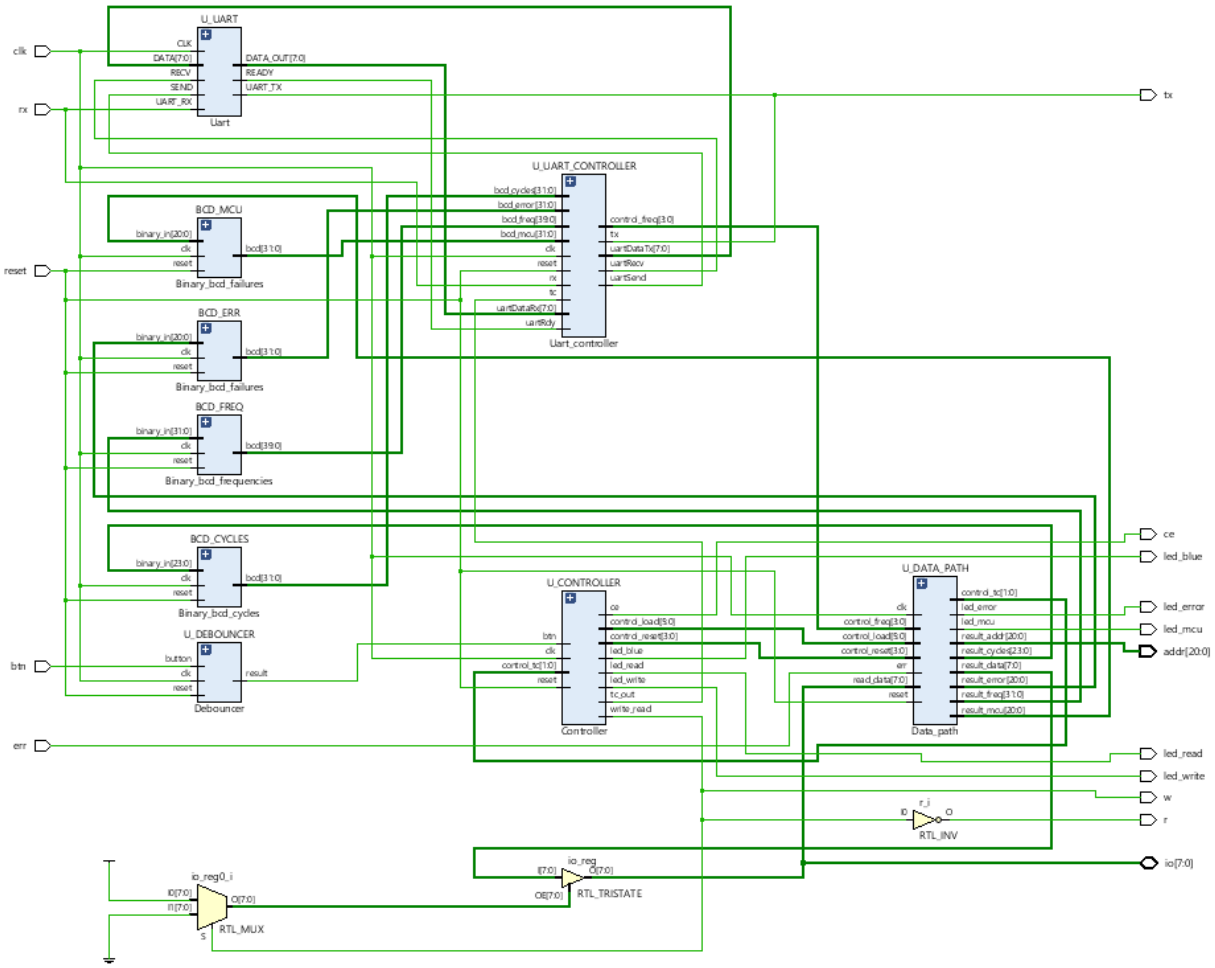

Figura 4.1 *Diagrama de bloques del módulo Scrub*

4.1.1.2 Datapath

En este módulo se instancian y conectan los módulos de los registros y contadores que constituyen la ruta de datos del diseño realizado en este proyecto. En él también se encuentra declarado el comparador del dato leído de memoria con el dato ("*hard-coded*") que se escribió anteriormente en ella, el resultado de esta comparación lo recibe MCU_COUNTER. También implementa el uso de los siguientes leds:

- **led_error**: si detecta que la señal ECC se ha activado permanece activo hasta que se realice una nueva lectura. Este led corresponde al color verde del led 0 de la placa, que es un led RGB.

- **led_mcu**: si detecta un error MCU permanece activo durante toda la lectura hasta que se realice una nueva. Este led corresponde al color rojo del led 0.

En la Figura 4.2 se muestra el diagrama de bloques del *Datapath*.

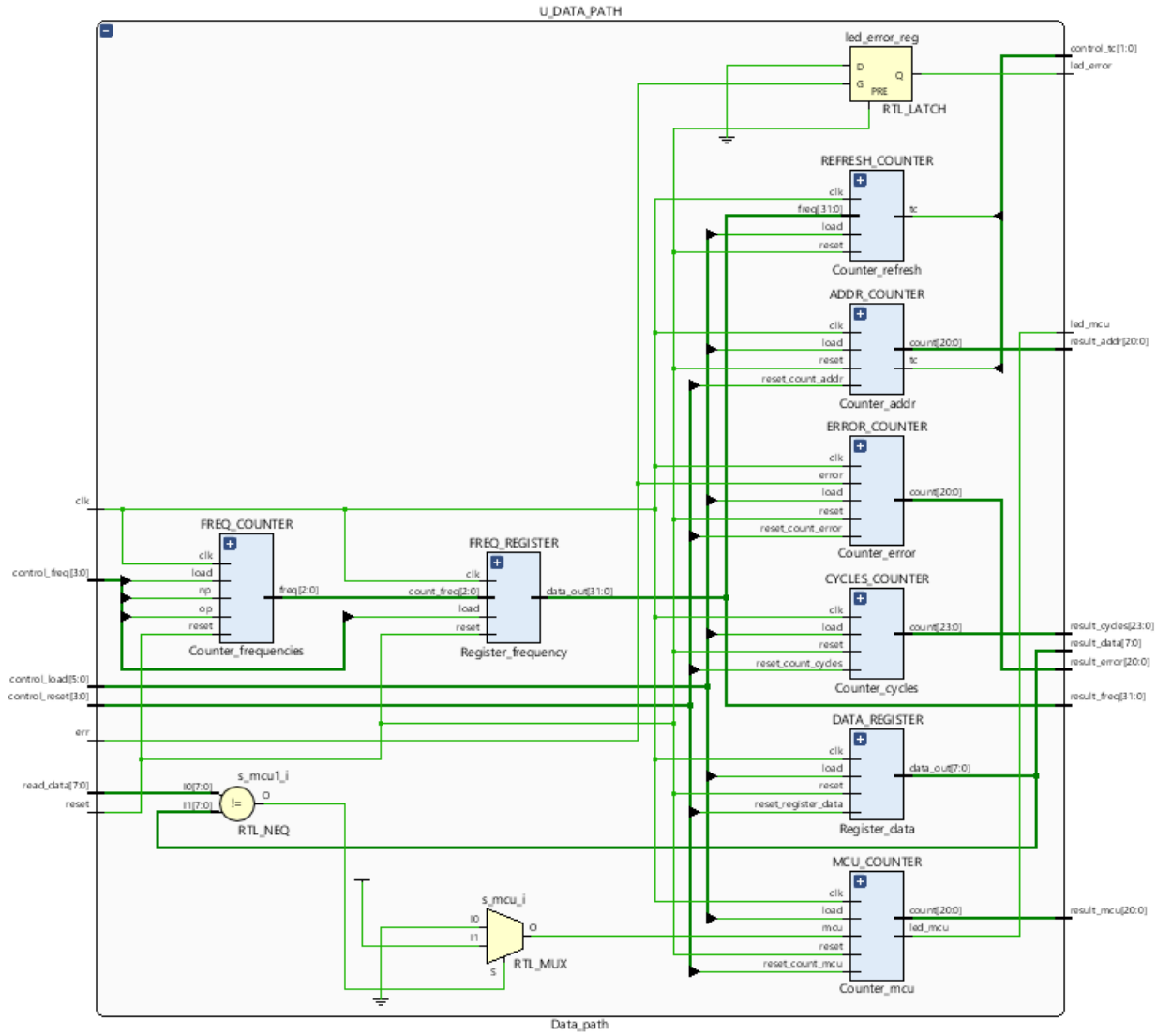

Figura 4.2 *Diagrama de bloques del Datapath*

Estos son:

- **ADDR_COUNTER:** se trata de un contador a través del cual la memoria se direcciona para así realizar las lecturas en todo su espacio de direccionamiento. Asimismo, se controla que no cuente fuera del rango de direcciones establecido (constante ADDRESS) y una vez alcanzado el máximo envía una señal al controlador indicando que ha llegado a su fin.

- **ERROR_COUNTER:** lleva la cuenta de los errores que se reciben de la salida ERR de la memoria.

- **MCU_COUNTER:** lleva la cuenta de los errores MCU que le llegan desde el comparador.

- **CYCLES_COUNTER:** cuenta de los ciclos desde que se inicia la lectura hasta que ésta llega a su fin.

- **FREQ_COUNTER:** aumenta o disminuye su valor en función de la señal op (un 0 suma y un 1 resta). Para aumentar o disminuir la posición en la que se encuentra de la tabla de frecuencias.

- **REFRESH_COUNTER:** cuenta que hayan pasado los ciclos necesarios, establecidos en FREQ_REGISTER, para poder realizar una nueva lectura. En caso de cambiar el tiempo de refresco a uno menor, si se ha superado la cuenta del nuevo máximo establecido, se acaba la cuenta actual y se indica al *Controller* que ya puede empezar un nuevo proceso de lectura, que se realizará teniendo en cuenta el nuevo tiempo de refresco establecido.

- **DATA_REGISTER:** almacena el dato, establecido por defecto, que se escribe en la memoria. Dado que, de momento, no es relevante que el usuario tenga que cambiar este dato manualmente, se ha decidido que su valor por defecto esté definido en el propio código (es decir, su valor está "*hard-coded*"). Aunque este dato podría haberse guardado con una constante en el código, se decidió tener un registro por si, en el futuro, se decidiese que el dato escrito en la memoria debiera ser variable.

- **FREQ_REGISTER:** almacena el periodo de refresco actual. Ésta se recupera de una tabla de periodos (FREQ_TABLE), la cual es indexada a través del contador *Counter_frequencies* (ver Figura 4.3).

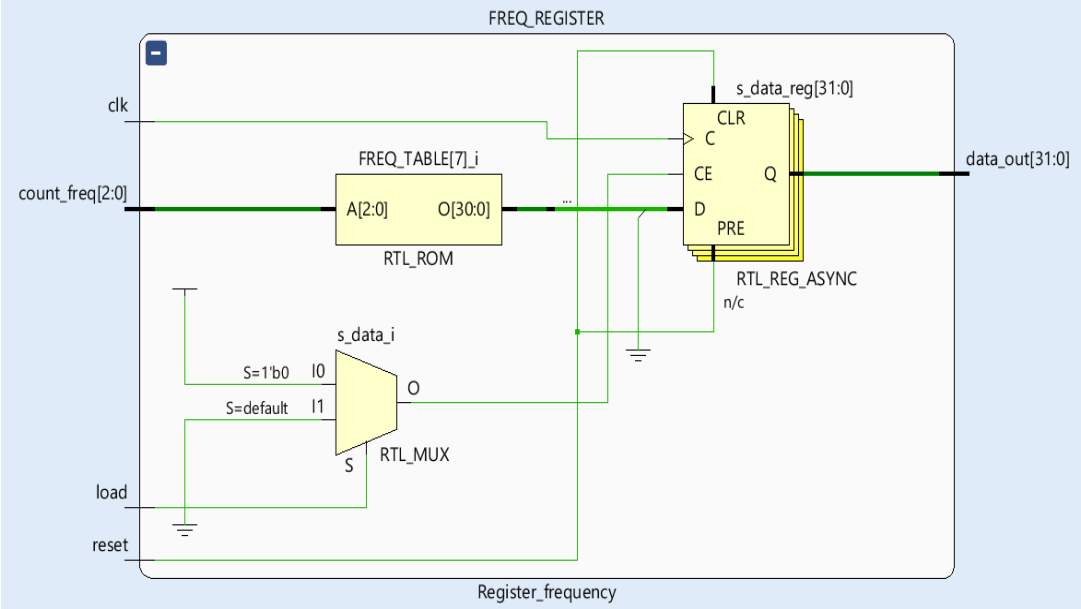

Figura 4.3 *Diagrama de bloques del FREQ_REGISTER. Nótese que la tabla de periodos (FREQ_TABLE) se encuentra dentro de este módulo*

4.1.1.3 Controller

Módulo que contiene la máquina de estados (FSM o *Finite State Machine*) principal, que se encarga de la gestión de la interacción con la memoria realizando una escritura inicial y posteriormente refrescos periódicos.

Para el proceso de escritura en la memoria CY62167GE30 se necesitan 2 estados. El dato escrito es el que está almacenado en el registro DATA_REGISTER (explicado anteriormente), cuyo valor es constante. Después del proceso de escritura se realizan refrescos periódicos, que consisten en leer todas las posiciones de la memoria, para lo que se necesitan otros 4 estados adicionales.

Dicha máquina de estados es síncrona para evitar incongruencias en los datos generados, pero contiene un *clear* asíncrono que establece los valores iniciales de todas las señales de salida y el estado inicial como IDLE.

Se diferencian un total de 5 estados, se pueden observar en la Figura 4.4.

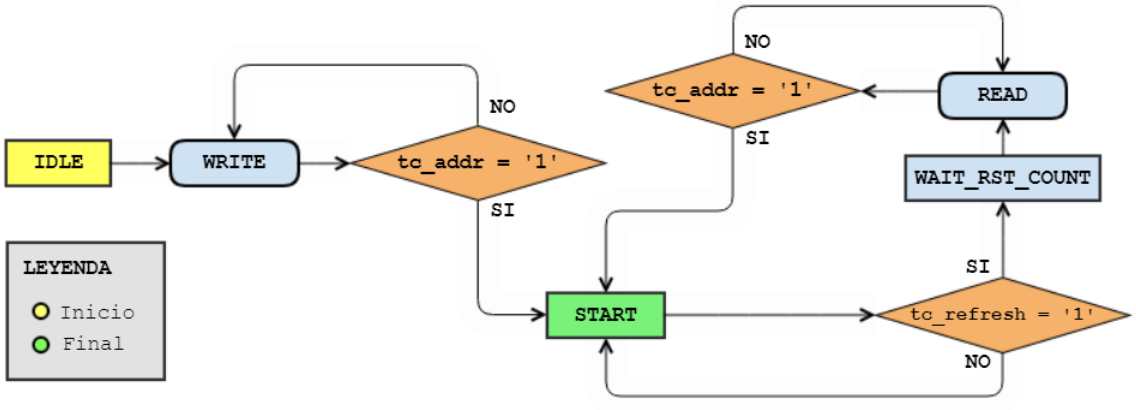

Figura 4.4 *Diagrama ASM[7] de la máquina de estados del componente Controller, https://www.gliffy.com/*

- **IDLE:** Es el estado inicial con el que se establecen los valores iniciales de todas las señales de salida. Su siguiente estado es WRITE.

- **WRITE:** En este estado se comprueba si se ha activado la señal proveniente de ADDR_COUNTER que indica que se ha llegado a la última posición de memoria (*tc_addr* = '1'). En caso afirmativo se desactiva el *chip enable*, se activa la señal *load* de REFRESH_COUNTER y el siguiente estado es START. En caso contrario, se prosigue con la escritura del valor almacenado en DATA_REGISTER en la posición indicada por el ADDR_COUNTER, y se mantiene en este estado.

- **START:** Estado en el cual se mantiene la FSM hasta que se activa la señal de refresco proveniente de REFRESH_COUNTER, que indica que se puede realizar una nueva lectura estableciendo como siguiente estado WAIT_RST_COUNT.

- **WAIT_RST_COUNT:** Estado en el cual se reinician los valores de los contadores CYCLES_COUNTER, ERROR_COUNTER y MCU_COUNTER. Siguiente estado READ.

- **READ:** En este estado se comprueba si se ha activado la señal proveniente de ADDR_COUNTER que indica que se ha llegado a la última posición de memoria (*tc_addr* = '1'). En caso afirmativo se desactiva el *chip enable*, se activa la señal *load* de REFRESH_COUNTER y el siguiente estado es START. En caso contrario se sigue con el proceso de lectura desde el valor almacenado en la posición indicada por el ADDR_COUNTER, en ese caso se mantiene en este estado.

[7] Todos los diagramas ASM se han generado desde https://www.gliffy.com/

En un principio se implementó que las lecturas se hicieran cuando se pulsaba el botón 0 de la XC7A35T. Sin embargo, ahora eso se hace con el tiempo entre refrescos. No obstante, por si fuera necesario en un futuro indicar manualmente cuándo se debe leer de la memoria desde la FPGA, se ha dejado el proceso que se encargaba de la gestión del botón.

También implementa el uso de varios leds, declarados como se muestra en la Figura 4.5. Su funcionalidad es la siguiente:

- **led_write**: activo cuando se está realizando la escritura en memoria. Este led corresponde al led 1 de la placa XC7A35T (ver figura 2.8).

- **led_read**: activo cuando se realiza la lectura de memoria. Este led corresponde al led 2 de la placa XC7A35T.

- **led_blue**: este led corresponde al color azul del led 0. Está desactivado para no interferir con los colores rojo y verde, ya que por defecto los tres colores del led RGB se mantienen activos.

```
## LEDs
set_property -dict { PACKAGE_PIN A17   IOSTANDARD LVCMOS33 } [get_ports { led_write }]; #IO_L12N_T1_MRCC_16 Sch=led[1]
set_property -dict { PACKAGE_PIN C16   IOSTANDARD LVCMOS33 } [get_ports { led_read }]; #IO_L13P_T2_MRCC_16 Sch=led[2]

set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports { led_blue}]; #IO_L14N_T2_SRCC_16 Sch=led0_b
```

Figura 4.5 *Declaración de los leds en el fichero de pines*

4.1.1.4 UART

La UART, también conocida como *Universal Asynchronous Receiver-Transmitter*, proporciona un DTE (*Data Terminal Equipment*) al dispositivo en el que se implemente. Al instanciar la UART se define una interfaz RS-232, y su función es transformar las señales que provienen del bus de datos en serie a paralelo y viceversa.

Para la implementación realizada en este proyecto, se utilizó un código base obtenido de una demo diseñada para la Artix-7 XC7A35T-1CPG236C de Digilentic [41]. Posteriormente fue modificado para que pudiera recibir datos de consola y se ajustase a la frecuencia correspondiente usando la siguiente fórmula:

$$round\left(\frac{12MHz}{9600baudios}\right) = 1250\ ciclos$$

Este es el valor por el que hay que dividir la frecuencia del oscilador de la placa (12 MHz) para así transmitir a una velocidad de 9600 baudios. Es decir, se necesita un divisor de frecuencia de 1250 ciclos para obtener el reloj de transmisión/recepción de la UART a partir de la frecuencia de reloj de la FPGA.

Esta UART tiene dos funciones: enviar datos por consola o recibirlos:

- Para enviar datos por consola es necesario recibir una palabra de 8 bits. Estos se transmiten uno a uno a la consola, añadiendo al principio un bit de *start* y al final un bit de *stop* dentro de estos 8 bits para que la consola los pueda ensamblar.

- En cambio, para recibirlos por consola es el proceso contrario, se reciben los bits uno a uno y se reensamblan para formar la palabra de 8 bits.

Este módulo tiene una máquina de estados interna que es síncrona y consta de 5 estados como se puede observar en la Figura 4.6.

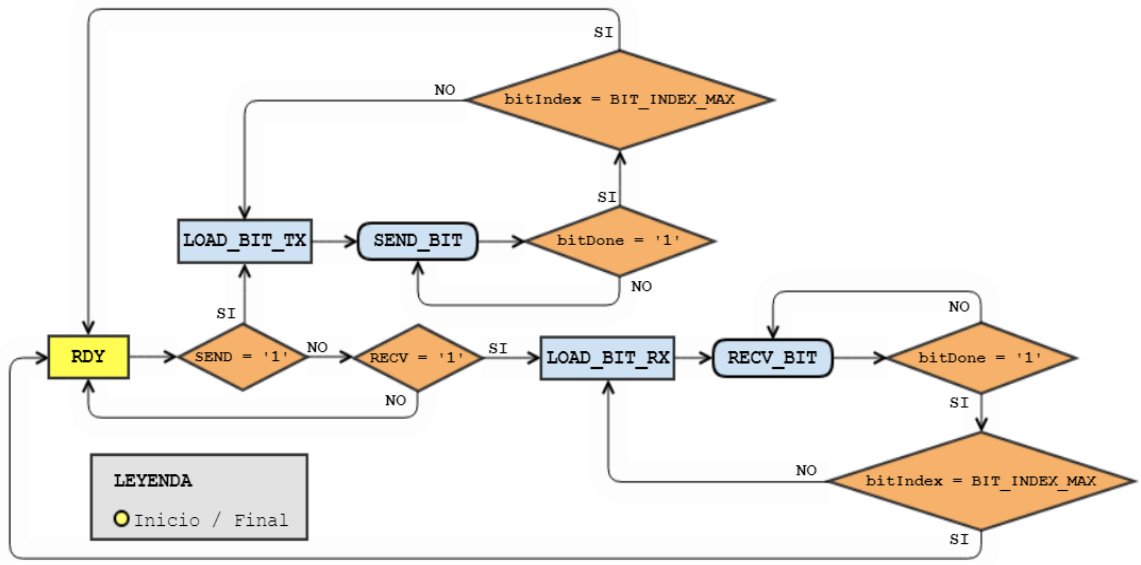

Figura 4.6 *Diagrama ASM de la máquina de estados del componente UART*

- **RDY:** estado inicial en el que se comprueba si se ha activado la señal de transmisión de datos SEND. Si es así, el siguiente estado es LOAD_BIT_TX, y si no, se comprueba si además se ha activado la señal de recepción de datos RECV, en cuyo caso el siguiente estado es LOAD_BIT_RX. De no haberse activado ninguna, se permanece en este estado.

- **LOAD_BIT_TX:** transmite a la consola el bit indicado por el índice[8] y aumenta dicho índice. El siguiente estado es SEND_BIT.

- **SEND_BIT:** comprueba si se ha terminado de transmitir el bit a la consola (*bitDone* = '1'). Si no, se mantiene en SEND_BIT, en caso afirmativo comprueba si ya se han transmitido todos y si es así el siguiente estado será RDY; en caso contrario, será LOAD_BIT_TX.

- **LOAD_BIT_RX:** almacena el bit recibido por consola en un vector *std_logic_vector* de 10 posiciones en la posición indicada por el índice y se aumenta dicho índice. El siguiente estado es RECV_BIT.

- **RECV_BIT:** comprueba si se ha terminado de recibir el bit desde la consola. Si no, se mantiene en RECV_BIT; en caso afirmativo comprueba si ya se han recibido todos y si es así, el siguiente estado será RDY. En caso contrario será LOAD_BIT_RX.

[8] Este índice recorre el array *DATA* que contiene 8 bits del dato a transmitir por la UART. Se usa para saber cuándo ha terminado de transmitir los 8 bits.

4.1.1.5 UART Controller

Este módulo gestiona los datos que se pasan al componente de la UART. Contiene la máquina de estados y los procesos necesarios para la gestión de la codificación y descodificación de los datos que se envían y reciben por la UART. Debido a que ésta sólo maneja datos de 8 bits, es necesario descomponer los datos que se desean enviar en ese tamaño.

Para ello se puede dividir en 4 funcionalidades diferentes:

- Cuando se desea enviar datos que ya están definidos como constantes, se pasan dichos datos al bus de datos.

- Cuando los datos que se desean enviar están en BCD hay que guardarlos en el bus anteriormente mencionado.

- Cuando ya se tienen cargados los datos en el bus se envían a la UART los bits de 8 en 8.

- Cuando se detecta que se están recibiendo datos de consola se le debe indicar que empiece el reensamblado y esperar a recibir el bit de confirmación para almacenar la palabra que devuelve en una señal.

Para implementar esta máquina de estados se tomó como base la que había en la demo diseñada por Digilentic [41].

Se trata de una máquina de estados síncrona (ver Figura 4.7), con una señal de *clear* asíncrona que inicializa las variables de salida y control de los contadores y registros y establece START como el estado inicial. Tiene 15 estados que se pueden agrupar en diferentes categorías:

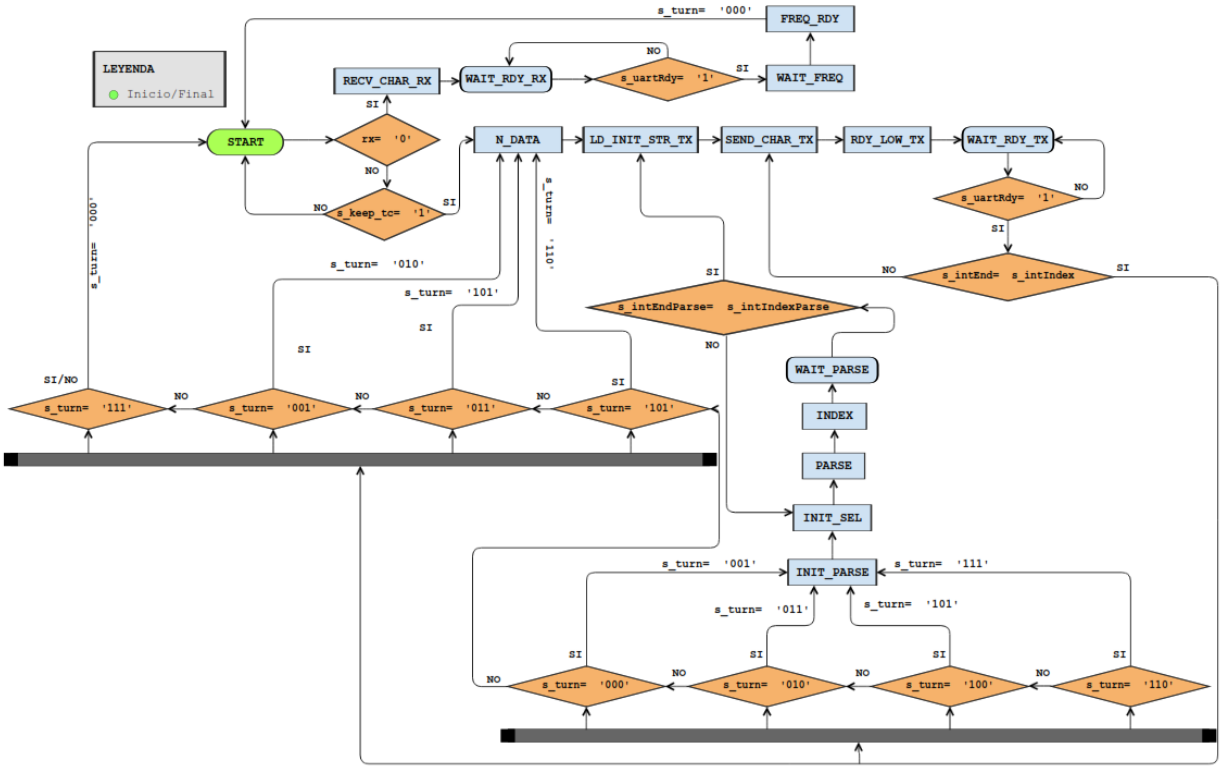

Figura 4.7 *Diagrama ASM del controlador de la UART*

- Estado de espera para transmitir o recibir datos:

  - **START:** estado de espera para transmitir o recibir datos. Comprueba si están recibiendo datos por consola, en cuyo caso establece RECV_CHAR_RX como el siguiente estado. Si se ha activado la señal *s_keep_tc*, que indica que se ha terminado con el proceso de lectura de la memoria, el siguiente estado es N_DATA y en caso contrario, se mantiene en START.

  - **INIT_PARSE, INIT_SEL, PARSE, INDEX, WAIT_PARSE:** este conjunto de estados (ver Figura 4.8) forman un bucle que gestiona la transformación de los datos a guardar en bus de datos. Primero se inicializan el índice inicial y final del bucle dependiendo del dato que se quiera mostrar, lo cual viene definido por la señal *s_turn* (ver Tabla 4.2). Después va transformando en cada iteración 4 bits de formato BCD a ASCII y una vez transformados los 4 bits se van guardando en el bus de datos hasta que el índice inicial alcance el valor del índice final. Si no llega al final, va aumentando el índice inicial.

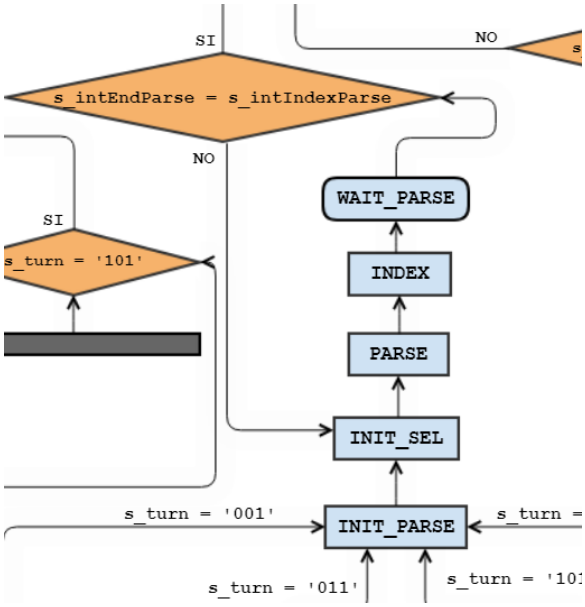

Figura 4.8 *Diagrama ASM del controlador de la UART, conversión de datos*

- **N_DATA:** este estado realiza la asignación de los datos constantes de tipo *string* (ver Figura 4.9) directamente al bus de datos dependiendo del valor de *s_turn*, en la Tabla 4.2. Establece el tamaño de los datos guardados en el bus.

```
    type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);
constant CYCLES_STR : CHAR_ARRAY(0 to N_CYCLES_STR-1) := (X"0A",   --\n
        X"0D",   --\r
        X"43",   --C
        X"59",   --Y
        X"43",   --C
        X"4C",   --L
        X"45",   --E
        X"53",   --S
        X"3A",   --:
        X"20"); --
```

Figura 4.9 *Ejemplo de valor constante de tipo string*

- **LD_INIT_STR_TX, SEND_CHAR_TX, RDY_LOW_TX, WAIT_RDY_TX:** (ver Figura 4.10) conjunto de estados que forman un bucle que gestiona el envío de datos a la UART. Primero se inicializa el índice inicial y toma como índice final el tamaño del dato que vaya a ser enviado. Se activan las señales del envío de datos de la UART y se va enviando a esta una porción de 8 bits del bus de datos en cada iteración. Finaliza cuando se hayan transmitido todos los bits de del bus de datos.

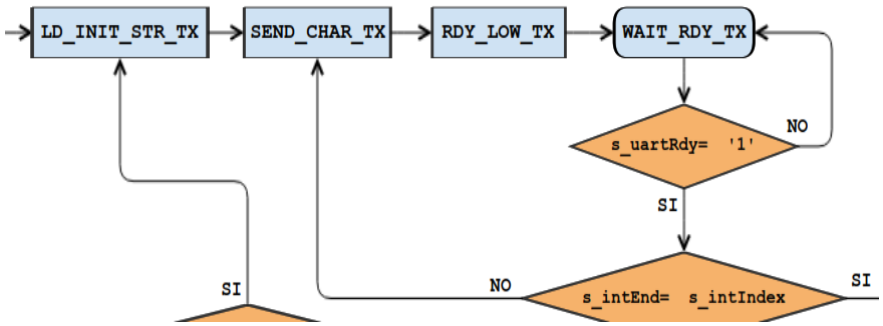

Figura 4.10 *Diagrama ASM del controlador de la UART, transmisión de dato tipo string constante*

s_turn	Estado siguiente	Tamaño de datos a enviar	Datos a guardar en DATA_STR
000	INIT_PARSE_CYCLES	N_CYCLES_STR	CYCLES_STR
001	N_ERROR	N_CYCLES_INT	X"3" & s_sel_cycles
010	INIT_PARSE_ERROR	N_ERROR_STR	ERROR_STR
011	N_MCU	N_FAILURES_INT	X"3" & s_sel_error
100	INIT_PARSE_MCU	N_MCU_STR	MCU_STR
101	N_FREQ	N_FAILURES_INT	X"3" & s_sel_mcu
110	INIT_PARSE_FREQ	N_FREQUENCY_STR	FREQUENCY_STR
111	START	N_FREQUENCY_INT	X"3" & s_sel_freq

Tabla 4.2 *Contiene los turnos de transmisión de la UART, el estado siguiente, los tamaños máximos y los datos a guardar en DATA_STR.*

- **RECV_CHAR_RX, WAIT_RDY_RX, WAIT_FREQ, FREQ_RDY:** (ver Figura 4.11) conjunto de estados en los que se avisa a la UART de que va a recibir datos por consola y luego esperan a que la UART termine de recibir. Si el valor recibido es un + o un - se indica al contador FREQ_COUNTER que esa es la operación a realizar y en función de esa operación aumenta o disminuye el periodo de refresco de la memoria (ver Tabla 4.1).

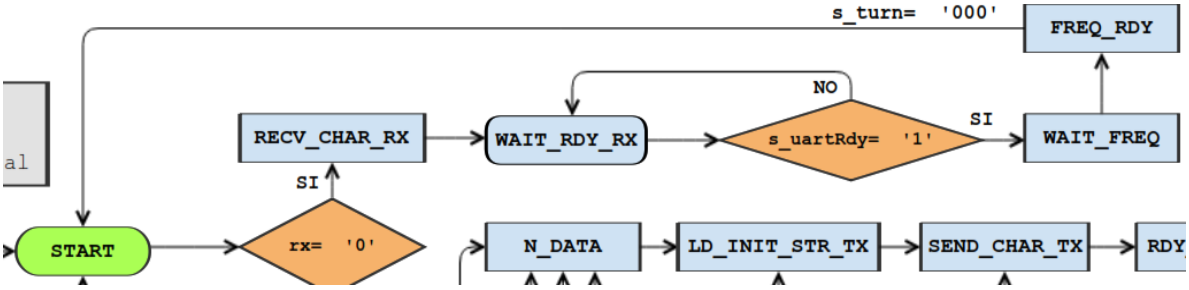

Figura 4.11 *Diagrama ASM del controlador de la UART, recepción de dato enviado por el usuario*

4.1.1.6 Conversor de binario a BCD

Para que el número en binario recibido por la UART sea fácilmente comprensible por un ser humano es necesario un componente que permita transformar los datos que se muestran por consola de binario a BCD.

En este proyecto se utilizó un componente obtenido en la web [42] que se ajusta a las necesidades del proyecto. Partiendo del mismo, se crearon tres componentes distintos, ya que los datos que se van a convertir a BCD tienen tamaños distintos:

- **BCD_CYCLES**: conversor cuya entrada es un vector de 24 bits (el tamaño máximo que puede tomar el valor de los ciclos de reloj que se tarda en realizar una lectura de memoria), y cuya salida es un vector de 32 bits (8 dígitos BCD).

- **BCD_ERR**: conversor que se usa tanto para los errores ECC como para los errores MCU. Su entrada es un vector de 21 bits (el tamaño máximo que puede tomar el valor de la cantidad de errores que se pueden observar en la memoria) y su salida es un vector de 28 bits (7 dígitos BCD).

- **BCD_FREQ**: conversor que transforma el valor en binario de los ciclos que debe esperar el programa para que se realice un refresco de memoria, con un vector de 32 bits de entrada y un vector de 40 bits de salida (10 dígitos BCD).

4.1.1.7 *Debouncer*

En las primeras versiones se usó un *debouncer* [43] (o eliminador de rebotes) para controlar rebotes del botón usado para realizar una lectura. En la última versión se añadió la funcionalidad de modificar el tiempo de refresco de la memoria y la lectura de memoria automática, por lo que ya no eran necesarios ni el botón ni el *debouncer*. Estos fueron inhabilitados por si se quisiera dar una funcionalidad distinta al botón en el futuro.

Un *debouncer* tiene como función eliminar los inevitables rebotes de un botón mecánico y devolver el valor generado por el mismo con transiciones (de 0 a 1 o de 1 a 0) totalmente limpias, sin pulsos espurios. A su vez, este *debouncer* tiene integrada una funcionalidad que permite establecer un tiempo de estabilidad que debe cumplir el botón. En este caso, ya que la mayoría de los botones/*switches* alcanzan esa estabilidad a los 10 ms se estableció ese tiempo. Así se evita que se active la lectura repetidas veces al presionar el botón, y aseguramos que el programa reconoce una única pulsación.

En el proyecto se utilizó el componente *debouncer* disponible en la web de Digikey. Se instanció este componente pasándole como reloj el *clk* de 12Mhz, configurando el *stable_time* a un valor de 10 ms (es el tiempo que debe permanecer estable la salida obtenida por los *flip-flops* observados en la Figura 4.12), y su entrada *button* sería la salida del botón 0 de la placa que es el botón sobre el que se querían eliminar los rebotes. Este componente genera de salida una señal limpia sin rebotes, que es la que se le pasa al controlador.

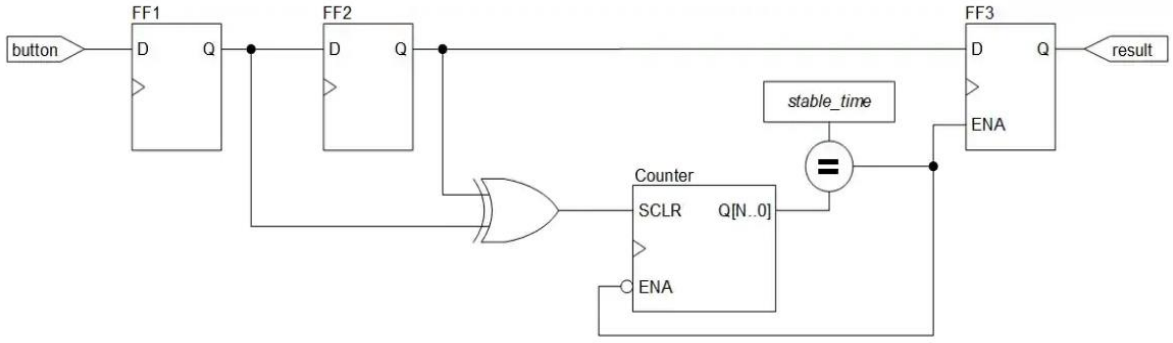

Figura 4.12 *Diagrama de bloques del debouncer*

Su funcionamiento es el siguiente: como se puede observar en la Figura 4.20, está formado por 3 *flip-flops*, un contador, un comparador y una puerta lógica XOR. El botón está conectado de forma directa con FF1, quien a su vez está conectado a FF2. Estos *flip-flops* guardan los dos estados lógicos más recientes del botón. En el momento que ambos estados son idénticos durante el tiempo estable de 10 ms, se habilita el FF3 (a través de su ENA) y por tanto, en la salida *result* se muestra la salida del botón limpia.

4.1.2 Desarrollo

Antes de realizar la implementación, se estudiaron los *datasheet* de los elementos *hardware* utilizados en el proyecto, la memoria SRAM CY62167GE y la FPGA Artix-7 XC7A35T-1CPG236C.

El desarrollo del proyecto se ha llevado a cabo por versiones para ir añadiendo funcionalidades paso a paso y comprobando su correcto funcionamiento:

- **V1.0:** se desarrolló una primera versión de la máquina de estados del controlador que solamente realizaba los procesos de lectura y escritura usando una memoria simulada como un componente de la FPGA en vez de usar la memoria SRAM física. Se hizo así para poder simular con un *testbench* su funcionamiento y poder comprobar que fuera el correcto. También se crearon el ADDR_COUNTER y DATA_REGISTER.

- **V2.0:** en esta versión se sustituyó la memoria simulada por la memoria real, se añadió el U_DEBOUNCER para poder utilizar correctamente los botones de la placa, el botón 0 para realizar una lectura de memoria y el botón 1 (ver Figura 2.9) para reiniciar el *hardware*. También se añadió un comparador para comprobar si los datos leídos eran correctos, que más tarde se eliminó y cuya función se añadió al MCU_COUNTER. Por último, se modificó la máquina de estados para adaptarla al funcionamiento con botones y el nuevo comparador.

- **V3.0:** en esta versión se implementó el control de errores para comprobar que se escribía y leía correctamente de la memoria. Para ello se añadieron los ERROR_COUNTER, MCU_COUNTER y el y un registro de errores[9] con sus dos instancias. Se controlan dos tipos de errores: los errores que son informados y corregidos por el ECC de la memoria y los errores múltiples (*Multiple Cell Upsets* o MCUs), que son aquellos en los que falla más de un bit en una sola palabra de la memoria, que el ECC no es capaz de corregir. En caso de producirse alguno de estos errores se enciende un led avisando del error. Se modificó la máquina de estados para funcionar con los nuevos componentes.

- **V4.0:** en esta versión se introdujo la UART para poder mostrar los ciclos que tardaba el proceso de lectura de la memoria. Para ello fue necesario implementar CYCLES_COUNTER para contar el número de ciclos y un registro[10] para almacenar esta cuenta además de introducir 3 conversores de binario a BCD para enviar los datos transformados posteriormente a ASCII por la UART. También se volvió a modificar la máquina de estados del *Controller*, para añadir las nuevas señales de control.

- **V4.1:** es una mejora de la versión anterior, que además de mostrar los ciclos, muestra la cuenta de los errores (tanto los mostrados por el ECC de la memoria como los MCUs). Nótese que en la versión V3.0 sólo se encendía un led cuando ocurría, al menos, uno de estos errores. En esta versión, dada la enorme cantidad de errores MCU producidos se intuyó que no se estaba realizando correctamente la lectura y escritura. Tras un nuevo estudio del *datasheet* de la memoria se llegó a la conclusión de que faltaba un estado de espera tanto en el proceso de lectura como de escritura, WAIT_READ y

[9] Registro posteriormente eliminado por optimización del diseño

[10] Registro posteriormente eliminado por optimización del diseño.

WAIT_WRITE[11] respectivamente, para que diera tiempo a realizar dichas operaciones.

- **V5.0:** en esta versión se incluyó la posibilidad de que el usuario decida aumentar o disminuir el periodo de refresco de manera manual, indicándolo con las órdenes '+' y '-', respectivamente. Para ello se tuvo que mejorar la UART para añadir la funcionalidad de recepción de datos y así poder indicar por consola si se desea aumentar o disminuir la misma. También se añadieron FREQ_COUNTER, REFRESH_COUNTER y FREQ_REGISTER; y se modificó la máquina de estados para añadir las señales de control de los nuevos componentes y eliminar la opción de lectura pulsando el botón de la placa (dicha lectura ya no se hace de forma manual, sino periódicamente de acuerdo con la frecuencia de refresco indicada).

- **V6.0:** en esta versión se realizó una optimización general del código. En concreto se redujeron los estados de la máquina de estados del controlador de la UART de 44 a 15, ya que se comprobó que muchos estados eran redundantes y se modificó el código para eliminarlos y mantener la funcionalidad. También se eliminaron varios registros que se usaban para guardar los valores de los contadores, que eran innecesarios. Se modificaron los módulos BCD que anteriormente recogían el valor transformado de binario a BCD en 8 o 10 vectores de 4 bits cambiándolos por un único vector del tamaño necesario. Se cambiaron las constantes que contienen los *strings* que se muestran por la UART del módulo *Scrub* al fichero *Definitions* y se movió toda la máquina de estados del controlador de la UART a un nuevo módulo aparte. Se eliminaron los estados de espera de la lectura y la escritura ya que se descubrió un error de cálculo y se vio que no eran necesarios.

[11] Estados posteriormente eliminados.

4.2 Desarrollo en KiCad

Antes de realizar la implementación de la placa PCB se investigó en distintos sitios web para obtener información acerca del *software* KiCad, de su funcionamiento y de su aplicación al proyecto [44]:

El diseño de una PCB en KiCad consta de los siguientes tres pasos:

4.2.1 Diseño del esquemático

Para empezar con el diseño se ejecutó el software *Schematic Layout Editor*, el cual permite generar el esquemático de la PCB. A continuación, se hizo la selección de los símbolos de los componentes para la creación de la PCB en función de los requisitos del proyecto, que en este caso fueron:

- 2 conectores genéricos hembra de 24 pines cada uno, mostrados en la Figura 4.13.a).
- 1 conector genérico hembra de 48 pines, mostrado en la Figura 4.13.b).
- 1 condensador, mostrado en la Figura 4.13.c), necesario para estabilizar la tensión y filtrar pulsos espurios.
- 1 power flag [45], es una etiqueta que KiCad necesita para saber que el componente al que se conecta va a recibir alimentación, mostrado en la Figura 4.13.d).

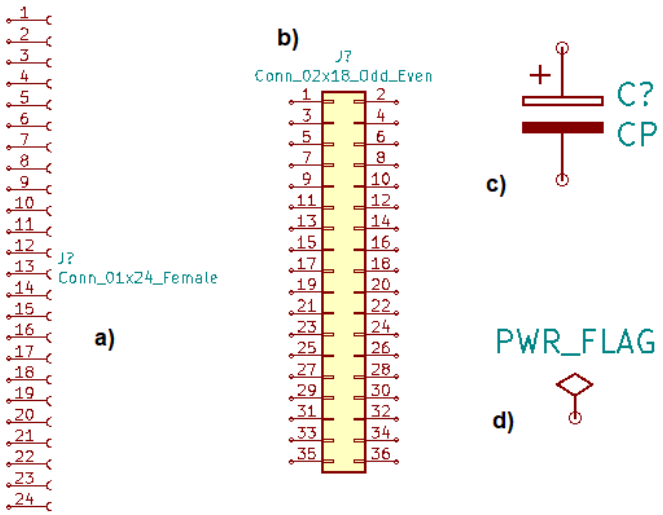

Figura 4.13 *Componentes de la PCB*

Una vez insertados los símbolos de los componentes en el esquemático se les referenció con un nombre. Este paso es obligatorio ya que, en caso de no asignar un nombre a los símbolos, el programa no puede hacerles referencia y se genera error al realizar el chequeo final:

- J1: Conector izquierdo de la Cmod A7-35T, contiene los pines 1-24, Figura 4.14.a).

- J2: Conector derecho de la Cmod A7-35T, contiene los pines 25-48, Figura 4.14.b).

- J3: Conector del bus con la memoria CY62167GE30, Figura 4.14.c).

- C1 (Condensador) y PWR_FLAG (Etiqueta de alimentación), Figura 4.14d).

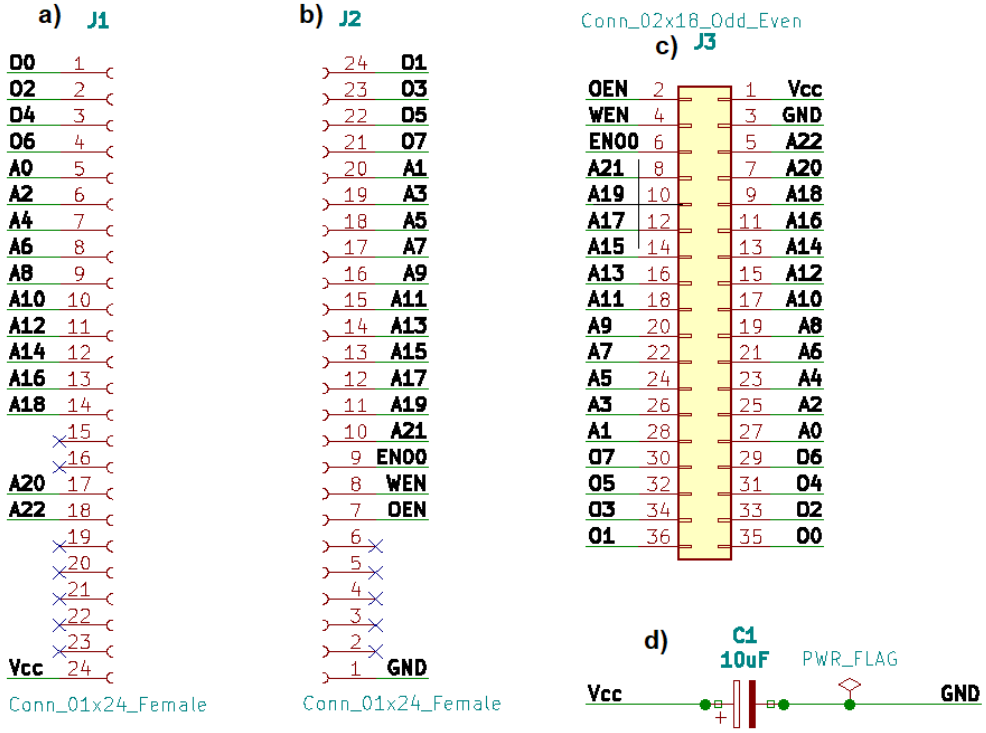

Figura 4.14 *Diseño del esquemático de la PCB*

Después de definir los símbolos de los componentes, se realizaron las conexiones entre sus pines. Se optó por usar una funcionalidad que proporciona KiCad, *Añadir etiqueta de red,* para evitar la aglomeración e intersección de las conexiones. Con esta funcionalidad se puede establecer la conexión de unos pines con otros referenciándolos con un nombre, quedando así el esquemático mucho más limpio. Para indicar los pines sin conexión, se utilizó la funcionalidad, *Añadir símbolo de no conexión.*

El siguiente paso consistió en revisar el diseño del esquemático con la funcionalidad *Realiza un control de las reglas eléctricas.* Este paso no es obligatorio,

pero permite comprobar que todas las conexiones son correctas y que no hay incongruencias como pueden ser pines sin conexión que no se hayan indicado, componentes sin referenciar o conexiones erróneas. En el caso de este proyecto, al ejecutar el control de reglas, solo dio un fallo: el condensador necesitaba un *power flag* anteriormente mencionado, que KiCad en concreto necesita para saber que dicho pin está conectado a la corriente.

Por último, con la funcionalidad *Generar listado de redes* se obtuvo el fichero con las conexiones entre componentes y se asociaron las huellas de éstos a los símbolos del esquema con la funcionalidad *Asignar huellas a símbolos de esquema*.

En la Figura 4.15 se pueden observar las huellas o *footprints* ya asignadas a los símbolos del proyecto. Para los conectores se seleccionó una huella de conector estándar con 24 pines y con un paso estándar de 2.54 mm. Ambos conectores se dispusieron en posición vertical. En cuanto al condensador, se seleccionó un condensador estándar circular con dos patillas de las dimensiones indicadas en la Figura 4.15. Se buscó un condensador electrolítico con una capacitancia de 10µF y que soportase el menor voltaje, 16V en este caso. Observando la tabla del *datasheet* [46] de los condensadores radiales electrolíticos de aluminio de largo 5 mm, se obtuvieron las medidas exactas para poder seleccionar la huella. En este caso fue la de un condensador estándar radial de 4 mm de diámetro, 5 mm de largo y pines de 1.5 mm de grosor. Como resultado final se obtuvo el esquemático mostrado en la Figura 4.14.

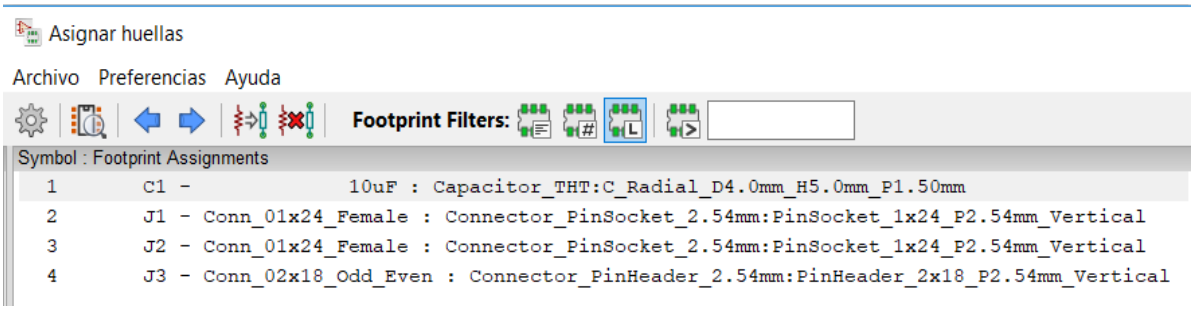

Figura 4.15 *Asignación de huellas a los símbolos del esquemático*

4.2.2 Diseño de la PCB

En los pasos anteriores se generó la lista de redes y se asignaron las huellas a los símbolos de los componentes. Se ejecutó el software *PCB Layout Editor* y seguidamente se usó la funcionalidad *Update PCB from schematic*, que permite cargar las huellas de los componentes ya conectados a partir del listado de redes, el cual se debe asociar previamente con la funcionalidad *Load netlist*.

Una vez cargadas las huellas, se colocaron en el plano de forma que la distancia entre ellas fuera la misma que la de los componentes físicos. En cuanto a las dimensiones de la PCB, se calculó el tamaño en milímetros de la placa que mide

de ancho 17.78 mm y de largo 69.85 mm y con la funcionalidad de medición *Añadir dimensión* se establecieron los límites. Primero se ajustaron en base a estos límites los conectores J1 y J2, que corresponden a la Cmod A7-35T y por último se colocaron el conector J3 y el condensador C1 entre los conectores J1 y J2 para aprovechar el espacio de la PCB. Podemos ver el resultado final en la Figura 4.16.

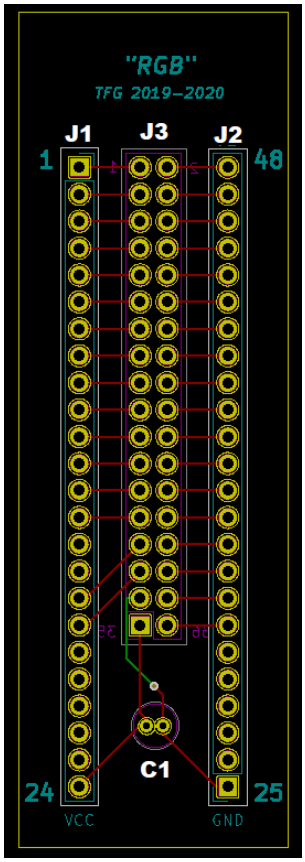

Figura 4.16 *Footprint final de la PCB*

Se diseñó la placa de tal forma que las conexiones entre pines no se cruzasen, y para ello se crearon dos capas. Se repartieron los componentes entre las dos capas existentes de tal forma que los dos conectores de 24 pines, J1 y J2, se encuentran en la capa frontal mostrada en la Figura 4.17.a) y el conector de 36 pines, J3, junto con el condensador, C1, se encuentran en la capa trasera mostrada en la Figura 4.17.b).

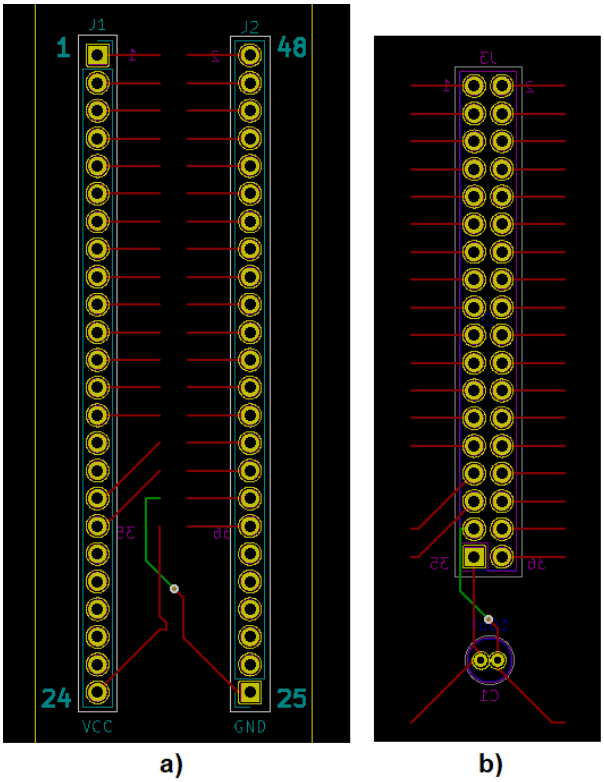

Figura 4.17 *Vista de las huellas frontal y trasera con las pistas de la PCB*

Con la funcionalidad *Enrutar pistas*, se dibujó el camino a seguir por las pistas de cobre. Así, las conexiones de J1, J2 y J3 se realizaron en la capa frontal de la PCB mientras que la conexión entre el GND (*ground*) situado en J2 con el GND de C1 se realizó en la capa trasera, como se puede observar en la Figura 4.18.a). En todas estas figuras, las pistas de la capa trasera son de color verde mientras que las de la capa frontal son rojas.

En este proyecto solo se cruzaba una conexión, la de GND, por lo que para poder enrutar la pista de la capa frontal a la trasera se usó una vía, es el círculo gris que se puede observar en la Figura 4.18.a), el cual une la pista verde con la roja. Esta vía permite conectar pistas de distintas capas entre sí, y en este caso, se evitó la intersección de la pista GND con la de VCC (*Voltage Common Collector*). En la Figura 4.18.b) se puede observar esto último, así como el trazado de las pistas generadas para conectar los pines 33 de J3, 2 de C1 y 25 de J2.

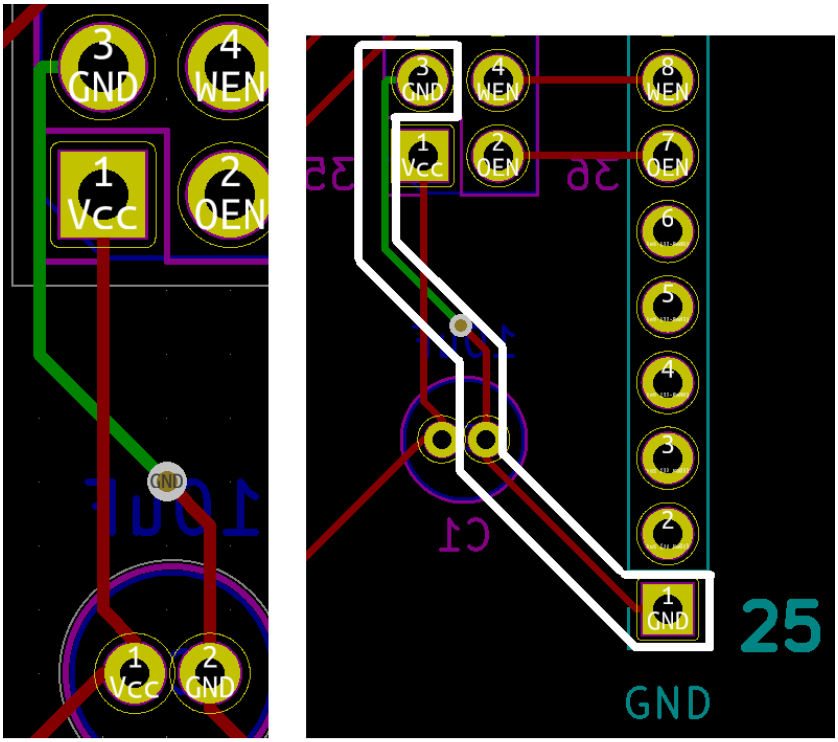

Figura 4.18 *Conexión de las pistas de la capa superior e inferior de los pines con señal GND*

Por último, se revisó el trazado de las pistas y, para facilitar el uso de la PCB, se añadieron algunas aclaraciones visuales a su diseño exterior. Se añadió la numeración de pines correspondientes a la Cmod A7-35T, se incluyeron solo los pines iniciales y finales de los conectores J1 y J2 en la capa frontal (Figura 4.19.a), para saber cuál es la posición de la placa. De manera similar, en la capa trasera también se aclaró la numeración de los pines del conector J3 (Figura 4.19.b).

Como detalle final se indicaron los pines correspondientes a la alimentación, Vcc, y a tierra, GND de los conectores J1 y J2, para situar bien la placa y se añadió un título y la fecha de realización del proyecto.

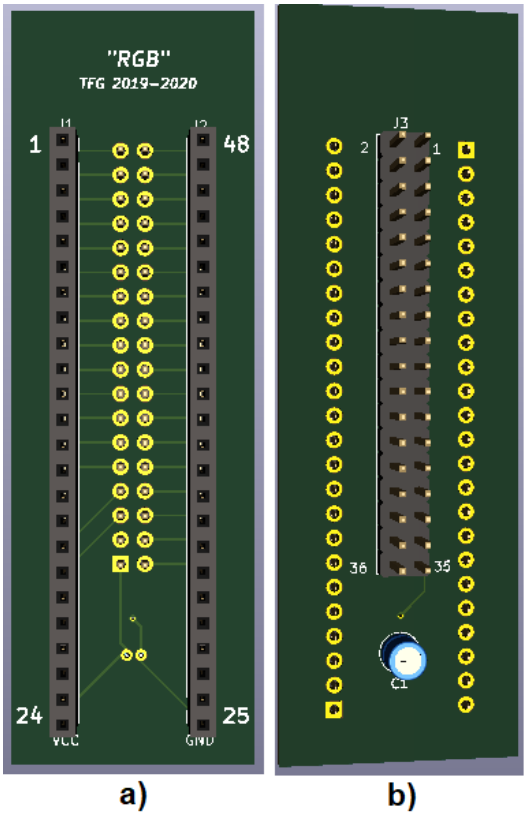

Figura 4.19 *Vista con la herramienta visor 3D del diseño de la PCB final*

4.2.3 Generación de ficheros Gerber

Para esta PCB se configuró la generación de los ficheros para las siguientes capas en formato Gerber con los ajustes que se muestran en la Figura 4.21. En este proyecto, al usar dos capas, se deben incluir como mínimo las siguientes:

- *F_Cu* / *B_Cu*: Capas de cobre superior e inferior
- *F_SilkS* / *B_SilkS*: Serigrafías superior e inferior
- *F_Mask* / *B_Mask*: Máscaras de soldadura superior e inferior
- *Edge_Cuts*: Bordes

En la Figura 4.20 se pueden observar todos los ficheros Gerber generados de las capas seleccionadas.

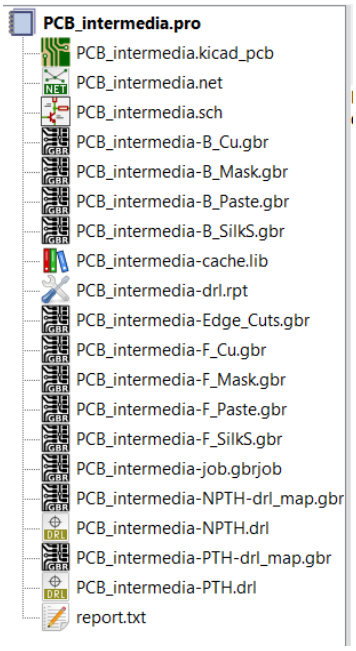

Figura 4.20 *Vista final de los ficheros del proyecto de la PCB*

KiCad ofrece unos ajustes avanzados [47], tales como eliminar las máscaras de soldadura que se encuentran sobre las vías, o imprimir los valores y referencias de las huellas en la capa de serigrafía. Estas opciones, junto con la posibilidad de forzar la impresión de valores y referencias invisibles, son bastante útiles en el caso de tener una PCB con componentes muy pequeños, ya que genera documentos que facilitan la conexión y reparación de los componentes y pistas gracias a la serigrafía de las capas.

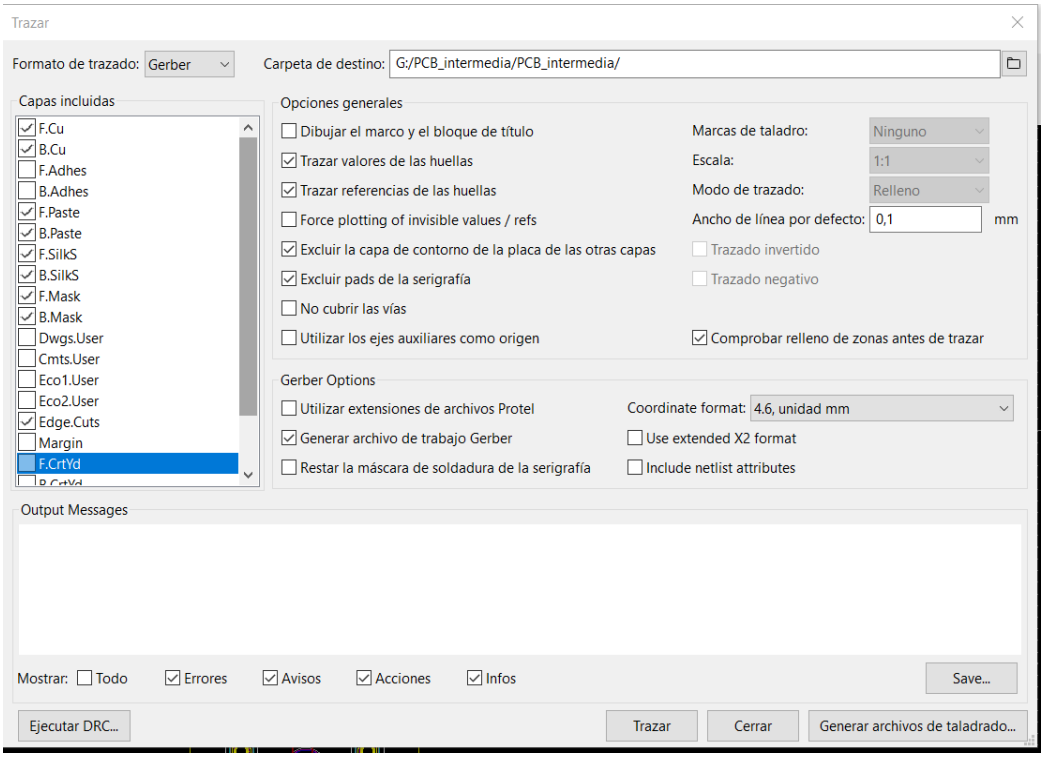

Figura 4.21 *Vista de la herramienta de generación de ficheros Gerber*

Para comprobar que la generación de los ficheros Gerber fue exitosa se usó la funcionalidad *GerberViewer* para observar todos los ficheros obtenidos. Así, en las Figuras 4.22.a) y 4.22.b) se pueden observar los ficheros generados de las capas frontal y trasera, respectivamente.

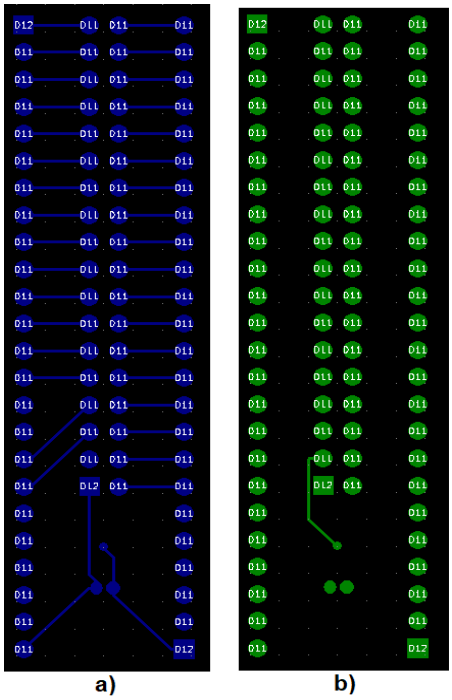

Figura 4.22 *Vista de la capa de cobre desde la herramienta GerberViewer*

Capítulo 5 - Conclusiones y trabajo futuro

Durante el periodo de desarrollo de este Trabajo de Fin de Grado, se han adquirido los siguientes conocimientos sobre:

- Los efectos de la radiación y la aplicación de DVS en memorias SRAM, tras la lectura de artículos de investigación.

- Los tipos de errores que se pueden generar al realizar la lectura de una posición de memoria y la funcionalidad ECC que presentan algunas SRAM.

- La investigación de *datasheets* para conocer el funcionamiento y las características de componentes, lo que ha permitido realizar el diseño *hardware*.

- El control de operaciones sobre una memoria mediante el diseño en VHDL.

- El diseño de una UART que permite la transmisión y recepción de datos, así como un controlador que la gestione según los requerimientos del proyecto.

- El uso de un *software* que permite la transmisión y recepción de información enviada y recibida, respectivamente, por la UART.

- El funcionamiento de un *software* para creación de PCB y el diseño de la misma según las especificaciones del proyecto.

Durante los años de grado se han adquirido conocimientos que han sido de ayuda a la hora de realizar el proyecto: por una parte, la asignatura de Tecnología de Computadores aportó la base de conocimiento sobre VHDL y el uso de FPGA; la asignatura de Estructura de Computadores, aunque no se impartió en VHDL, proporcionó conocimientos sobre el uso de FPGA y control de periféricos.

En particular, el Grado de Ingeniería de Computadores cuenta con asignaturas como Sistemas Empotrados y Programación de Sistemas y Dispositivos, que complementan lo aprendido por las dos asignaturas anteriormente mencionadas, se profundiza en el control de periféricos en lenguaje VHDL y se usan componentes como la UART para la transmisión y recepción de información con la interfaz Termite.

Las funcionalidades implementadas en este TFG, que se encuentran disponibles en la plataforma Github [48], van a incorporarse a la tesis doctoral del estudiante de doctorado Mohammadreza Rezaei, quien está haciendo su tesis doctoral en el contexto del proyecto de investigación TIN2017-87237, también asociado a la realización de este TFG. En cuanto los trabajos futuros que pueden realizarse sobre este proyecto se encuentran:

- El estudio de los efectos de la radiación durante la aplicación de DVS sobre la SRAM para mejorar la eficiencia.

- La posibilidad de añadir varias memorias que trabajen en paralelo.

- Estudiar experimentalmente, con un analizador lógico, el incremento de consumo que supone hacer un ciclo de refresco, sobre una o varias memorias.

- De igual manera, estudiar experimentalmente la reducción del consumo al aplicar DVS.

- Obtener 2 frecuencias consideradas como críticas: la frecuencia mínima de refresco que hace que la fiabilidad no varíe a pesar de bajar la tensión de alimentación (esto previsiblemente disminuye el consumo) y la frecuencia máxima de refresco que hace que el consumo no varíe a pesar de realizar dicho refresco (esto previsiblemente aumenta la fiabilidad).

Capítulo 6 - Introduction

6.1 Motivation

The interest in making a hardware project but without leaving aside the programming part, as well as the great versatility and flexibility of FPGAs (Field-Programmable Gate Arrays) [1], which can be used in many different areas, motivated us to choose these as the basis of our project.

Once we decided that we wanted to carry out the project with an FPGA, we decided to choose this specific topic for two reasons: the first one is the opportunity it offers us to collaborate and be part of a national research project, with reference TIN2017-87237, granted by the Spanish Ministry of Science, Innovation and Universities [2]. The second one is the possibility that our work can be used to mitigate the effects of radiation in the exploration of environments as hostile as space.

6.2 Goals

The aim of this project is to study the possible use of commercial off-the-shelf (COTS) memories in a radiation environment (such as the aeronautics or space sector) with the aim of saving costs since space certified components represent higher costs, as they are not manufactured in such large volumes as COTS components. Another reason for using COTS components in such environments is that they feature more processing power and storage capacity than space certified components (good examples are the European Space Agency's (ESA) Exomars and GAIA missions [3, 4]) and many of them already incorporate error detection and correction mechanisms (also known as Error Correcting Codes or ECC) that can be used to mitigate the effects of radiation [5-7].

In addition, it is desired to do this in an energy-efficient manner. To attain this objective, it is assumed that apply Dynamic Voltage Scaling (DVS) is applied, which consists of reducing the supply voltage of certain components of a system (such as volatile memories) while they are not being used in order to save energy. However, if the supply voltage is reduced, their sensitivity to radiation increases, as it has already been demonstrated in numerous studies involving various particle sources such as protons, neutrons or heavy ions [8-11].

To counteract this effect, a very effective method in memories implementing ECC is to make periodic readings of their entire contents (which we will call "refresh cycles" in the following). The period of these should be calculated according to the number of errors produced in the memory in the environment in which it is working. The ECC of the memory that is used in this project (as case study) automatically corrects simple errors when a memory word is read, and in addition the device includes an output pin that notifies that such action had to be taken, so it is possible to count the number of errors that were corrected during a refresh cycle. This allows

the frequency of the refresh cycles to be adjusted, increasing it if many errors were detected, or vice versa.

However, during such a refresh, an increase in power consumption occurs, so a study should be carried out to ensure that this is done effectively and that there is no excessive increase in power consumption due to too many refresh cycles or too many errors due to not enough ones.

To carry out this study, a SRAM memory (Static Random Access Memory [12]) has been used. In particular, a CY62167GE30, manufactured by Cypress Semiconductor [13], was used. Its operation was controlled by using as a system implemented in the Artix-7 FPGA that exists on a 48-pin Cmod A7-35T board [14]. The Artix-7 reads and writes to this memory and keeps track of the errors detected.

Finally, to facilitate the physical connection between the pins of the Cmod A7-35T board and the CY62167GE30 memory, a PCB (Printed Circuit Board) [15] has been designed using the free software KiCad [16]. This PCB allows to establish the connection between the two main components that are part of our project[12] in a simple and clean way.

6.3 Work plan

Most of the project development and implementation was done in group, otherwise there were usually at least two team members.

Gonzalo Fernández-Díez Ponte

At the beginning of the project, we made a separate study of the documentation provided by the advisors on the CY62167GE30 memory and the Cmod A7-35T board, and, since all the documentation was in English, we made an individual summary and translation of the most important parts in order to start with the implementation. In addition, we performed a VHDL review using, above all, the notes from the *Tecnología de Computadores* subject.

After finishing the documentation study, we started to meet at the Computer Science Faculty's library, approximately twice a week in two-hour meetings, to start with the development in VHDL of the first version of the project.

With the beginning of the COVID pandemic, we could not continue to hold face to face meetings, so we had to start holding them by video call, normally 5 two-hour meetings a week, in which one of the members carried out the development in Vivado, sharing the screen with the rest in order to collaborate. All the tests and debugging on the FPGA and the memory had to be carried out by Beatriz, as she was the only one who had these physical components.

[12] FPGA and SRAM memory

During this period we finished the development of the reading, writing, memory error control and the counting of the cycles that takes a reading, as well as the implementation of the UART to be able to communicate with the board and show the obtained data.

Once this part was done, we started with the design of the PCB in KiCad to establish the connections between the FPGA and the memory in a simple and orderly way. For this part of the project we continued using the same working methodology. We were able to finish and print the PCB already in the post-confinement period.

For the development of the BsC thesis dissertation, since we had done the project in common for the most part, we decided to divide the different parts among ourselves by drawing lots, corresponding to me the following points: 1.1 Motivation, 2.2 FPGA Artix-7 XC7A35T-1CPG236C, 2.3 Cmod A7-35T, 3.3 Termite and 4.1 Development in Vivado, this last part I did together with Raquel. In addition, I was in charge of the realization of the bibliography and references using the Mendeley reference manager.

Beatriz Villegas Sánchez

She did the study and research of the datasheets from the components that were going to be used in the Bachelor thesis as well as additional documentation such as articles provided by the advisors, tutorials, etc.

During the development of the project, she read documentation about the VHDL language in order to do the project implementation correctly. On average, the time of participation in the project since January was of 2 hours 5 days per week.

She developed with the whole team a module that simulated the physical memory and a basic functional controller with the reading and writing states.

When the COVID pandemic started, Beatriz kept the physical components (FPGA, memory and connection cables). This made it difficult to make an equitable distribution of tasks, so the whole team was connected to continue with the implementation. They looked for component modules that they needed such as the constraints file of the Cmod A7-35T board for the synthesis tool and Beatriz found a suitable debouncer and added it to the project. She also checked which pins of the Cmod board were going to be used and implemented that part. In this case only Buttons 0 and 1 and the LEDs 1 and 2 were used.

In order to debug the implementation of the above mentioned memory module, Beatriz assigned the reading functionality to button 0 and the writing functionality to button 1. In addition, she made sure that LED 1 was activated while a reading was performed and LED 2 for writing, so that it could be checked from the outside that the implementation of the controller was working. She also did isolated tests to control the Buttons and LEDs of the Cmod A7-35T.

She did, with all the team, the implementation of the reading cycle counter, they did the searched for a suitable binary to BCD converter to be used in the project. The latter was added by Beatriz to show the number of cycles through the UART. The team modified this together to make it fit with the project specifications. Later Beatriz implemented the MCU error counter and implemented (with all the team) the binary to BCD converters to show the ECC and MCU errors. The appropriate modifications were also made to the CY62167GE30 memory controller.

As this implementation was almost finished, the operating errors were debugged with all the team and finally Beatriz detected that too many MCU errors appeared when the operations of reading and writing from the CY62167GE30 memory actually worked fine. This was caused by the bad contact of the cables, so she put them back and it worked. Before discovering this solution, many tests were performed showing data by the UART to verify that the functionalities were correct and to discard possible failures. During this process, as Vivado took a long time to synthesize, implement and generate the bitstream, Beatriz had to carry out several tests and debug on her own. This made possible to speed up the development of the project.

Once these modifications were completed, Beatriz gave the components to Raquel. Beatriz read in various tutorials how to design a PCB and, with all the team, the design was carried out and sent to print. Then, it was necessary to add a new functionality consisting in carrying out automatic refresh cycles according to a refresh period provided as input. In order to make this happen, Beatriz and Raquel implemented the refresh counters and registers as well as the frequencies, in addition to establishing consistent refresh periods in the refresh table.

Beatriz and Raquel investigated how to adapt the existing UART to add the data reception functionality, modified it by adding new processes and states to the UART module state machine and debugged it because it initially failed. They added the functionality of displaying the refresh period value by console in number of cycles format.

Finally, Beatriz and Raquel optimized the hardware design, significantly reducing the number of states of the UART controller.

For the development of the BsC thesis dissertation, Beatriz participated in the generation of its structure. She developed Sections 3.2 KiCad, 4.1.1.7 Debouncer, 4.2 KiCad development (with all the subsections) and participated in the development of Sections 4.1.1.3 Controller, 4.1.1.4 UART, 4.1.1.5 UART Controller, 4.1.1.6 Binary to BCD converter, 4.1.2 Development, 5 Conclusions and future work. In addition, she modified the bibliography and references and generated the index of the memory, the index of the figures and tables and the diagrams of the state machines.

Raquel Ramos Corral

She read the documentation provided by the advisors: datasheets, research articles, and other documents necessary for the development of the project. She also had to learn more about the VHDL language when it was necessary.

When the whole team carried out this research, they started to develop together the design of a module that simulated the behaviour of the CY62167GE30 memory to be used, a counter that was in charge of increasing the memory address and a basic state machine to perform the reading and writing. A testbench was created for debugging purposes. When this part was not yet finished, the first period of exams began, due to which the team decided to stop the development of the project for a month. Raquel was the first one to stay free, so she continued with the debugging of what was implemented and finished it.

Once a simple controller was up and running, the team decided to start using CY62167GE30, to check if the controller that had been implemented was really correct, so Raquel looked for the pin file that allowed to establish the connections of the design with the board.

Shortly after this, the COVID pandemic started. Having only one Cmod A7-35T and one CY62167GE30, we started to make 5 meetings a week of 2 hours duration approximately, because Beatriz was the person who had these components.

To check whether or not the previously implemented design worked with the CY62167GE30, we used the LEDs and buttons for debugging.

Later a cycle counter was implemented, but in order to show its value it was agreed to implement a UART. Once added, the state machine that manages the data sent to the UART was modified to adapt it to the project.

To ensure that the data was interpreted by the console in the right format, a BCD converter was sought for the cycle count. The whole team participated in the necessary debugging to ensure that everything was displayed correctly.

Once this had been done, the tasks were distributed to implement error management. Raquel was in charge of implementing the counter of the errors coming from the ERR output of the memory and helped Gonzalo with the state machine that manages the data sent to the UART, to which the necessary states had to be added to show the number of errors.

When implementing the latter, it was found that the number of MCU errors was very high, so a debugging process was started and it took almost a month. In this process, with the use of the UART, the data that were considered convenient for the error detection were shown.

After the end of the COVID lockdown, Beatriz gave the board to Raquel.

When the implementation of the first part of the design was finished, the design of the PCB was started. To do this, Raquel had to investigate the use of KiCad. When the team finished with the study, the board was designed as a whole.

When the board was sent to print, Raquel and Beatriz began to add a new functionality that would allow a refresh period to be established during which automatic readings would be taken. To do this, they implemented two counters and a register as well as a table with the refresh periods.

A study was also carried out to add the functionality of data reception by the UART. Once the UART had been changed, the debugging was started as the data received from the console was not well interpreted. Due to the short time available to finish this part, Raquel finished the debugging of the UART and did the debugging of the three components previously implemented.

It was decided to show the refresh period by console so that the user would know the current refresh period, so Raquel and Beatriz made the necessary changes in the UART controller. Raquel moved the state machine that was in charge of managing the data sent to the UART to a new component for greater organization. Raquel and Beatriz reviewed the design for possible improvements and optimized the UART controller module. Finally, Raquel was in charge of making the changes suggested by the advisors to complete the implementation of the design.

For the development of the dissertation, Raquel was in charge of points 1.3 Work plan, 2.1 SRAM CY62167GE30-45ZXIES report, 3.1 Vivado and 4.1 Development in Vivado and 5 Conclusions and future work. She has also helped with point 1.2 Objectives. She was also in charge of generating the block diagrams.

Capítulo 7 - Conclusions and future work

During the development period of this project, the following knowledge has been acquired:

- The effects of radiation on components and systems, and the DVS applications of DVS to save power in SRAMs, after reading research articles.

- The types of errors that can be generated when reading a memory position and the ECC functionality presented in some SRAMs.

- How to read and interpret datasheets to learn the operation and characteristics of a component, which has allowed us to carry out the hardware design.

- The operations control a memory through VHDL design.

- The design of a UART that allows the transmission and reception of data and a controller that manages it according to the project requirements.

- The usage of software that is able to transmit and receive information sent and received by the UART.

- The operation of software for creation and design of PCBs according to the project specifications.

During the degree years, helpful knowledge for the project fulfilment has been acquired. On the one hand the *Tecnología de Computadores* subject provided the VHDL and FPGA use base knowledge. On the other and, the *Estructura de Computadores* subject, although not taught in VHDL, provided knowledge about FPGA and peripherals control.

In addition, the *Ingeniería de Computadores* degree has subjects such as *Sistemas Empotrados* and *Programación de Sistemas y Dispositivos* which complement what has been learned in the two subjects mentioned above, deepening in the peripherals control in VHDL and using components such as UART for information transmission and reception with the Termite interface.

The functionalities implemented in this project, that are located in the Github [48] platform, are going to be incorporated into the doctoral thesis of the PhD student Mohammadreza Rezaei, who is doing his PhD in the context of the research project TIN2017-87237, also associated with the realization of this BsC thesis. Referring to the future work that can be done on this project, we can mention the following points:

- The study of the radiation effects during the application of DVS on SRAM to improve efficiency.

- The possibility of adding more memories to the existing project, working in parallel.

- To experimentally study, with a logic analyzer, the increase in power consumption that involves a refresh cycle, on one or several memories.

- To experimentally study the reduction in power consumption when applying DVS.

- To obtain 2 refresh frequencies that can be considered as critical or very relevant: the minimum one which ensures that the reliability does not vary in spite of lowering the supply voltage (presumably, this will reduce the power consumption) and the maximum one which ensures that the power consumption does not vary in spite of making said refresh cycles(presumably, this will increase the reliability).

BIBLIOGRAFÍA

1. Xilinx. What is an FPGA? Field Programmable Gate Array 2020 [Internet]. Disponible en: https://cutt.ly/OfAVzei

2. Mecha López H, Molina Prego MC, Clemente Barreira JA, Gonzalez Calvo C, Fabero Jimenez JC, Mendías Cuadros JM. Técnicas hardware y software para el análisis y detección de errores inducidos por la radiación en sistemas digitales embarcados en misiones espaciales II. 2020 [Internet]. Disponible en: https://cutt.ly/RfSmn0B

3. European Space Agency Exomars 2016 [Internet]. Disponible en: https://cutt.ly/ifAVTvX

4. European Space Agency Gaia 2018 [Internet]. Disponible en: https://cutt.ly/OfAVKls

5. Shanken S. Protecting COTS ICs for Space. :92121.

6. Bokil H. COTS Semiconductor Components for the New Space Industry. 2020;1-4.

7. Allums KK, O'Neill PM, Reddell BD, Bailey CR, Nguyen K V. Radiation test results on COTS and non-COTS electronic devices for NASA johnson space center spaceflight projects. IEEE Radiat Eff Data Workshop. 2012.

8. Hiramoto T. Ultra-Low-Voltage Operation: Device Perspective. En: Proceedings of the 17th IEEE/ACM International Symposium on Low-Power Electronics and Design. IEEE Press; 2011. p. 59–60. (ISLPED '11).

9. Clemente JA, Hubert G, Fraire J, Franco FJ, Villa F, Rey S, et al. SEU Characterization of Three Successive Generations of COTS SRAMs at Ultralow Bias Voltage to 14.2-MeV Neutrons. IEEE Trans Nucl Sci. 2018;65(8):1858-65.

10. Uemura T, Kato T, Matsuyama H, Hashimoto M. Soft-error in SRAM at ultra-low voltage and impact of secondary proton in terrestrial environment. IEEE Trans Nucl Sci. 2013;60(6):4232-7.

11. Wu Q, Li Y, Chen L, He A, Guo G, Baeg SH, et al. Supply Voltage Dependence of Heavy Ion Induced SEEs on 65 nm CMOS Bulk SRAMs. IEEE Trans Nucl Sci. 2015;62(4):1898-904.

12. Definición de SRAM - Diccionario informático 2020 [Internet]. Disponible en: https://cutt.ly/WfAVMkw

13. Cypress Semiconductor Corporation CY62167GE30-45ZXI 2020 [Internet]. Disponible en: https://cutt.ly/TfAV3Aa

14. Artix-7 FPGAs Data Sheet: DC and AC Switching - Xilinx. 2018 [Internet]. Disponible en: https://cutt.ly/SfAV4Mt

15. Digilent Cmod A7 2020 [Internet]. Disponible en: https://cutt.ly/FfABwiE

16. KiCad Getting Started [Internet]. Disponible en: https://cutt.ly/ZfABexq

17. Laverde A. ¿PCB qué es y para qué sirve? 2017 [Internet]. Disponible en: https://cutt.ly/YfABt05

18. Mendeley 2020 [Internet]. Disponible en: https://www.mendeley.com/

19. Velazco R, Clemente JA, Hubert G, Mansour W, Palomar C, Franco FJ, et al. Evidence of the Robustness of a COTS Soft-Error Free SRAM to Neutron Radiation. IEEE Trans Nucl Sci. 2014;61(6):3103–3108.

20. Clemente JA, Franco FJ, Villa F, Baylac M, Ramos P, Vargas V, et al. Single events in a COTS soft-error free SRAM at low bias voltage induced by 15-MeV neutrons. IEEE Trans Nucl Sci. 2016;63(4):2072–2079.

21. Clemente JA, Hubert G, Franco FJ, Villa F, Baylac M, Mecha H, et al. Sensitivity Characterization of a COTS 90-nm SRAM at Ultralow Bias Voltage. IEEE Trans Nucl Sci. 2017;64(8):2188–2195.

22. Keim, R. What Is a Hardware Description Language (HDL)? 2020 [Internet]. Disponible en: https://cutt.ly/LfABaEd

23. Martínez, JI y Castillo J. ¿Qué es VHDL? Introducción a VHDL - VHDL.es. [Internet]. Disponible en: https://vhdl.es/tutorial-vhdl/

24. CPLD vs FPGA: Differences between them and which one to use? [Internet]. Disponible en: https://cutt.ly/UfABgva

25. Analog devices. A Beginner's Guide to Digital Signal Processing (DSP) 2020 [Internet]. Disponible en: https://cutt.ly/JfABjcw

26. Collins I. Phase-Locked Loop (PLL) Fundamentals 2018 [Internet]. Disponible en: https://cutt.ly/XfABlyT

27. Akthar S. Digital Clock manager DCM in Xilinx FPGA 2014 [Internet]. Disponible en: https://cutt.ly/gfABzmp

28. Xilinx. ARTIX-7 FPGAS 2018 [Internet]. Disponible en: https://cutt.ly/PfABxAE

29. Artix-7 FPGA Development Board - Digilent. 2020 [Internet]. Disponible en: https://cutt.ly/cfABnMh

30. Diligent. Arty Reference Manual 2020 [Internet]. Disponible en: https://cutt.ly/jfABEEQ

31. Xilinx. Vivado Design Suite HLx 2015 Editions [Internet]. Disponible en: https://cutt.ly/9fABROV

32. Ruiz de Clavijo P. Introducción a Verilog - Departamento de Tecnología Electrónica Universidad de Sevilla 2012 [Internet]. Disponible en: https://cutt.ly/PfABT4R

33. Xilinx. RTL and Technology Schematic Viewers Tutorial 2011 [Internet]. Disponible en: https://cutt.ly/lfABY1P

34. Kicad - EcuRed 2011[Internet]. Disponible en: https://www.ecured.cu/Kicad

35. Getting Started | KiCad EDA. [Internet]. Disponible en: https://cutt.ly/EfABIyx

36. Riemersma T. Termite: a simple RS232 terminal - CompuPhase. 2019 [Internet]. Disponible en: https://cutt.ly/1fABOrk

37. Dinesh T. RS-232C 2020 [Internet]. Disponible en: https://cutt.ly/gfABO7E

38. Baumann R. Soft errors in advanced computer systems. IEEE Des Test Comput. mayo de 2005;22(3):258–266.

39. Dixit A, Wood A. The impact of new technology on soft error rates. 2011 Int Reliab Phys Symp. 2011;5B.4.1–5B.4.7.

40. Darnell M. Error Control Coding: Fundamentals and Applications. Vol. 132, IEE Proceedings F Communications, Radar and Signal Processing. 1985. 1-68.

41. Digilent. Cmod-A7-35T-GPIO-GitHub. 2018 [Internet]. Disponible en: https://cutt.ly/EfABSRn

42. Akthar S. Conversor binario a BCD 2014 [Internet]. Disponible en: https://cutt.ly/sfABDJN

43. Larson S. Debouncer 2019 [Internet]. Disponible en: https://cutt.ly/5fABGfC

44. Rincón Ingenieril. KiCad - Introducción - YouTube 2015 [Internet]. Disponible en: https://cutt.ly/yfABHoL

45. S7.1 - Power flag symbols - | KiCad EDA [Internet]. Disponible en: https://cutt.ly/TfABH4K

46. Vishay Intertechnology, Inc. Condensador 097-RLP7 2020 [Internet]. Disponible en: https://cutt.ly/6fABJVN

47. Documentación Pcbnew [Internet]. Disponible en: https://cutt.ly/hfABKVp

48. GitHub Trabajo de Fin de Grado 2020 [Internet]. Disponible en: https://cutt.ly/yfGGJc4