



UNIVERSIDAD
DE MÁLAGA



ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento: Tecnología Electrónica.

Área de Conocimiento: Tecnología Electrónica.

TRABAJO FIN DE GRADO

**IMPLEMENTACIÓN EN VHDL SOBRE UNA PLACA FPGA
ARTIX-7 DEL MICROPROCESADOR DE 8 BITS , HTDI20.**

Grado en

Ingeniería Electrónica Industrial

Autor: DANIEL GARCÍA ESCOBAR

Tutor: JULIAN CALDERÓN ALMENDROS

Cotutor

MÁLAGA, Junio de 2.024

Implementación en VHDL sobre una placa FPGA ARTIX-7
del microprocesador de 8 bits, HTDI20.

Daniel García Escobar.

Agradecimientos

Este trabajo no habría sido posible sin todo el apoyo que me ha dado mi familia en este tiempo, que siempre me han apoyado en mis decisiones y me han animado a estudiar lo que a mí más me gustaba. En concreto, mi padre me despertó la vena de ingeniero con sus enseñanzas y mi madre animaba a desarrollar todas las inquietudes que tenía. Es por ello por lo que este proyecto se lo dedico a ella, que estaría muy orgullosa de verme acabar mi carrera y poder empezar a trabajar en lo que a mí me apasiona. Aunque no me puedo olvidar de mis hermanos mayores, que me han servido de inspiración, ya que me demostraron que era posible sacarse una carrera sin quedarse en el intento.

También le quiero agradecer a mi pareja todos los ánimos que me ha dado en este tiempo, además de pedirle disculpas por todas las noches que ha tenido que dormir con el ruido del teclado de fondo.

Y, por último, agradecer a todos mis amigos el tiempo que han pasado escuchando mis explicaciones de los avances que iba haciendo de este proyecto, lo que a ellos le sonaba a chino, pero en todo momento se han interesado sobre los avances que hacía.



**DECLARACIÓN DE ORIGINALIDAD DEL
PROYECTO/TRABAJO FIN DE GRADO**

D./ Dña.: Daniel García Escobar. DNI/Pasaporte: 25349704-R.

Correo electrónico: danielgarciaescobar@hotmail.com

Titulación: Grado en Ingeniería Electrónica Industrial.

Título del Proyecto/Trabajo: Implementación en VHDL sobre una placa FPGA ARTIX-7 del microprocesador de 8 bits, HTDI20.

DECLARA BAJO SU RESPONSABILIDAD

Ser autor/a del texto entregado y que no ha sido presentado con anterioridad, ni total ni parcialmente, para superar materias previamente cursadas en esta u otras titulaciones de la Universidad de Málaga o cualquier otra institución de educación superior u otro tipo de fin.

Así mismo, declara no haber trasgredido ninguna norma universitaria con respecto al plagio ni a las leyes establecidas que protegen la propiedad intelectual, así como que las fuentes utilizadas han sido citadas adecuadamente.

En Málaga, a 24 de junio de 2024.

Fdo.:

Resumen

En este Trabajo de Fin de Grado se implementa un microcontrolador teórico llamado HTDI20, en una FPGA de la familia Artix 7 del fabricante Xilinx. Este dispositivo fue diseñado para servir de apoyo en la docencia de la asignatura Sistemas Electrónicos Digitales. En la cual se estudia su funcionamiento y su estructura, que está completamente formada por componentes electrónicos comerciales, de los cuales los alumnos ya conocen su funcionamiento de asignaturas anteriores, para que de este modo cualquier alumno pueda entender cómo funciona. Es por ello por lo que esta implementación se ha realizado con la misma estructura que el impartido en la asignatura. Además, se ha diseñado de manera independiente cada bloque que componen el dispositivo, para que los alumnos no solo puedan probar el funcionamiento de todo el conjunto, sino que también puedan hacerlo de cada uno de los bloques, y que sirva de apoyo en las explicaciones impartidas en clase.

También se ha diseñado un compilador de lenguaje ensamblador, para que los alumnos puedan programar el dispositivo, y así verificar el funcionamiento de programas diseñados en clase. Para facilitar esta tarea, al microprocesador se le ha añadido un controlador de puerto serie. Que permite recibir los datos que el compilador envía por USB y los almacena en la memoria EEPROM del dispositivo.

Además, para hacer más sencillo el uso del microprocesador, se han diseñado dos PCBs. Una de ellas conecta la FPGA a la memoria EEPROM, y añade la posibilidad de alimentar el dispositivo con un rango de tensión de 5 V a 12 V, sin necesidad de que la FPGA esté conectada por USB. La otra placa, es un puerto paralelo formado por 4 entradas y 4 salidas de 8 bits, que sirve de ayuda a la hora de depurar programas.

Palabras Clave: Microcontrolador, FPGA, EEPROM, Compilador, PCB.

Abstract

In this Final Degree Project, a theoretical microcontroller called HTDI20 is implemented in an Artix 7 FPGA from Xilinx. This device was designed to support the teaching of the Digital Electronic Systems subject. In which its operation and structure is studied, which is completely made up of commercial electronic components, of which the students already know how they work from previous subjects, so that in this way any student can understand how it works. That is why this implementation has been carried out with the same structure as the one taught in the course. In addition, each block that makes up the device has been designed independently, so that students can not only test the operation of the whole, but also of each of the blocks, and so that it can serve as a support for the explanations given in class.

An assembly language compiler has also been designed so that students can program the device and thus verify the operation of programs designed in class. To facilitate this task, a serial port controller has been added to the microprocessor. This allows the data sent by the compiler via USB to be received and stored in the EEPROM memory of the device.

In addition, to make the use of the microprocessor easier, two PCBs have been designed. One of them connects the FPGA to the EEPROM memory and adds the possibility of powering the device with a voltage range of 5 V to 12 V, without the need for the FPGA to be connected via USB. The other board is a parallel port consisting of four 8-bit inputs and four 8-bit outputs, which is useful for debugging programs.

Key Words: Microcontroller, FPGA, EEPROM, Compiler, PCB.

Índice General

Capítulo 1. Introducción	21
1.1. Antecedentes	21
1.2. Objetivos	21
Capítulo 2. El microprocesador.....	23
2.1. Historia.....	23
2.2. Funcionamiento.....	25
2.3. Características	27
2.4. Tipos.....	28
Capítulo 3. FPGA.....	31
3.1. Introducción.	31
3.2. Estructura	31
3.3. VHDL.....	33
3.3.1. Constrain.	36
3.3.2. IP Cores.	36
3.3.3. Archivo COE.....	37
3.4. Placa de desarrollo CMOD A7	38
3.4.1. Interfaz USB a UART.....	40
Capítulo 4. Implementación del HTDI20.	41
4.1. Estructura del microprocesador HTDI20 teórico.	41
4.1.1. Arquitectura física.	42
4.1.2. Unidad de control.	43
4.2. Diferencias entre el HTDI20 teórico y el implementado.	51
4.2.1. Multiplicador de frecuencia.	51
4.2.2. Registros contadores de 16 bits.....	51
4.2.3. Controlador de interrupciones.....	53
4.2.4. Memoria de la unidad de control (UCROM).	53
4.2.5. Bus interno de señales de control (ICB).	55
4.2.6. Registro de interrupción vectorizada (IVECT).	56
4.2.7. Registro de interrupción autovectorizada (IAUTO).	56
4.2.8. Selector de zona.	57
4.2.9. Contador de zona.....	57
4.2.10. Reinicio del contador de zona.	58

4.2.11. Señal de reloj del SUC (CKSUC).....	58
4.2.12. Secuencial de la unidad de control (SUC).	59
4.2.13. Controlador UART.....	62
4.3. Códigos de instrucción.....	64
4.4. Cronogramas de funcionamiento.	66
Capítulo 5. Compilador.....	69
5.1. Introducción.	69
5.2. Implementación del compilador.....	69
5.2.1. Compilación de un archivo Flex.	72
5.3. Manual del compilador del HTDI20.	73
5.3.1. Estructura de los archivos de programación.	73
5.3.2. Tipos de errores.....	78
5.3.3. Instrucciones de uso del compilador.	78
Capítulo 6. PCBs del HTDI20.....	81
6.1. Placa de memorias.....	81
6.1.1. Características.	81
6.1.2. Selección de las memorias.	81
6.1.3. Conexión de la etapa de alimentación.....	83
6.1.4. Conexión de las memorias.	84
6.1.5. Conexión de las entradas del procesador.	84
6.1.6. Planos.	84
6.2. Placa de puerto paralelo.	86
6.2.1. Características.	86
6.2.2. Esquema de los puertos de entrada.	87
6.2.3. Esquema de los puertos de salida.....	88
6.2.4. Decodificador del puerto paralelo.	88
6.2.5. Planos.	89
Capítulo 7. Conclusiones.....	91
7.1. Introducción.	91
7.2. Conclusiones.	91
7.3. Líneas de trabajo futuras.	91
Bibliografía	93
ANEXOS	95
Anexo A: Juego de instrucciones del HTDI20.....	96

A.1.	Juego instrucciones y características.....	96
Anexo B:	Archivos de fabricación de las PCBs.....	103
B.1.	PCB de memorias.....	103
B.2.	PCB de puerto paralelo.....	105
Anexo C:	Bill of materials (BOM).....	107

Índice de figuras

Figura 1. Zuse Z4. (PFEIFFER, s.f.).....	23
Figura 2. Circuito del Intel 4004 con la firma de Federico Faggin. (Intel 4004 , s.f.).....	24
Figura 3. Evolución de la arquitectura de las FPGAs. (Maxfield, s.f.)	31
Figura 4. Estructura de los LUTs. (Uni)	32
Figura 5. Esquema DSP48. (AMD, s.f.).	33
Figura 6. Esquema de un PLL.....	33
Figura 7. Ejemplo de código en VHDL.	34
Figura 8. Ejemplo conexión de componentes.	35
Figura 9. Ejemplo de archivo XDC.....	36
Figura 10. Estructura básica de un archivo COE.	37
Figura 11. Placas de desarrollo BASYS 3 y CMOD A7 respectivamente.....	38
Figura 12. Estructura de un "Slice" de Artix 7.....	39
Figura 13. Interfaz USB a UART. (Digilent, 2019).....	40
Figura 14. Cronograma de funcionamiento del protocolo UART.	40
Figura 15. Esquema HTDI20 (bloques básicos).	41
Figura 16. Esquema de la Unidad de Microprograma.	45
Figura 17. Estructura de la arquitectura física.....	47
Figura 18. Estructura unidad de control.....	49
Figura 19. Multiplicador de frecuencias.	51
Figura 20. Cronograma de funcionamiento de los nuevos registros contadores.....	52
Figura 21. Circuito de control de la señal /INT.	53
Figura 22. Cronograma de funcionamiento del controlador de interrupciones.	53
Figura 23. Esquema del registro IVEC.	56
Figura 24. Esquema del registro IAUTO.	57
Figura 25. Conexión del bloque de vector inicio de interrupción.....	57
Figura 26. Esquema del reinicio del contador de zona.	58
Figura 27. Cronograma de funcionamiento del bloque de reinicio del contador de zona....	58
Figura 28. Esquema del generador de CKSUC.....	58
Figura 29. Cronograma de funcionamiento del generador de la señal CKSUC.	59
Figura 30. Diagrama de estados del SUC.	60
Figura 31. Cronograma funcionamiento del controlador UART.....	63
Figura 32. Esquema de conexión del controlador UART.....	63

Figura 33. Ejemplo del archivo Excel para diseñar los códigos de instrucción.....	64
Figura 34. Codificación de las instrucciones en el archivo Excel.....	65
Figura 35. Programa para el cronograma.....	66
Figura 36. Cronograma de funcionamiento	67
Figura 37. Estructura de archivo Flex.....	69
Figura 38. Esquema de compilación de un archivo Flex.	72
Figura 39. Aspecto del archivo de salida de codificación del programa.	76
Figura 40. Ejemplo de código para el compilador del HTDI20.....	77
Figura 41. Ejemplo de archivo de errores.	78
Figura 42. Ejecución del compilador (Solicitud de archivo).	79
Figura 43. Ejecución del compilador (El código no tiene errores).	79
Figura 44. Ejecución del compilador (Error al leer el archivo).	79
Figura 45. Ejecución del compilador (El código tienen errores).	79
Figura 46. Ejecución del compilador (Solicitud del número del puerto serie).	80
Figura 47. Ejecución del compilador (Puerto serie correcto).	80
Figura 48. Ejecución del compilador (Puerto serie incorrecto).	80
Figura 49. Regulador de tensión de 5 V.....	83
Figura 50 . Regulador de tensión de 3,3 V.....	83
Figura 51. Selector de memorias.....	84
Figura 52. Modelo 3D de la PCB de memorias.	85
Figura 53. PCB de memorias ensamblada.	86
Figura 54. Estructura de los puertos de entrada.	87
Figura 55. Estructura de los puertos de salida.....	88
Figura 56. Esquema del decodificador del puerto paralelo.....	89
Figura 57. Modelo 3D de la PCB de puerto paralelo.....	90
Figura 58. PCB de puerto paralelo ensamblada.	90
Figura 59. Top layer PCB de memorias.....	104
Figura 60. Bottom layer PCB de memorias.	104
Figura 61. Top layer PCB de puerto paralelo.	105
Figura 62. Bottom layer PCB de puerto paralelo.	106

Índice de tablas

Tabla 1. Operaciones de la Unidad Lógico Aritmética.....	43
Tabla 2. Lista señales de salida de la UCROM.....	54
Tabla 3. Señales del ICB.....	55
Tabla 4. Evolución de los estados de la SUC.....	61
Tabla 5. Codificación de las instrucciones básicas.....	65
Tabla 6. Suma de caracteres de los distintos grupos de instrucciones.....	66
Tabla 7. Patrones básicos en Lex.....	70
Tabla 8. Direccionamiento del espacio de memorias.....	84
Tabla 9. Juego de instrucciones con sus características.....	102
Tabla 10. Bill of Material del trabajo.....	107

Capítulo 1. Introducción

1.1. Antecedentes

La creciente demanda de circuitos integrados y la inminente construcción de una fábrica de semiconductores en España, va a incrementar la necesidad de especialista en el campo de la Microelectrónica. Esto ha llevado a España a promover másteres de formación en este sector, que cuenta con pocos profesionales en nuestro país.

Por tanto, asignaturas que formen a los alumnos en este campo, empiezan a cobrar un papel importante para cubrir esta demanda de profesionales. En concreto, el grado de Ingeniería Electrónica Industrial, cuanta con dos asignaturas, una de ellas es la asignatura Microelectrónica, en la que se enseña como implementar estos tipos de dispositivos complejos en una FPGA. Lo que abre un abanico de posibilidades a la hora de diseñar circuitos electrónicos. Y la otra asignatura es Sistemas Electrónicos Digitales, que muestra el funcionamiento interno de los microcontroladores con la ayuda de uno teórico, que está diseñado con componentes comerciales, de los cuales todos los alumnos conocen su funcionamiento de asignaturas anteriores. Siendo esto una buena introducción para este complejo y creciente sector.

El problema con el que cuenta este microcontrolador es que es teórico, por lo que los alumnos no pueden probar su funcionamiento y ver cómo se comporta en cada uno de los casos. Es por ello por lo que se lleva a cabo su implementación para la docencia, y que de este modo los alumnos puedan comprobar su funcionamiento.

Para llevar a cabo esta tarea se hace uso de una FPGA, en la cual se implementa el microcontrolador mediante bloques, de igual modo que en el libro de teoría de la asignatura (Sotorriño, 2021). Para que así los alumnos no solo puedan probar cómo funciona el dispositivo completo, sino que también puedan aislar estos bloques, y ver cómo se comportan de manera independiente. Lo que es de gran ayuda en las explicaciones dadas en clase.

1.2. Objetivos

El objetivo del presente Trabajo de Fin de Grado es modelar el microcontrolador teórico, HTDI20, en un lenguaje de descripción de circuitos electrónicos, en este caso VHDL. Y así poder implementar el dispositivo en una FPGA.

Para facilitar su uso se diseñará un compilador con su propio lenguaje ensamblador y un gran abanico de instrucciones, además de fabricar dos placas de desarrollo que le otorga una mayor versatilidad.

Capítulo 2. El microprocesador

2.1. Historia

Desde los inicios de la humanidad, el ser humano ha buscado la forma de automatizar procesos para hacer su vida más fácil, lo que ha dado lugar a que en la actualidad contemos con gran cantidad de dispositivos que nos facilitan muchas tareas. Uno de los elementos claves para este avance tecnológico ha sido la creación del microprocesador.

El primer artilugio de computación del que se tiene constancia es el Mecanismo de Anticitera, creado por los griegos a finales del siglo II a.C. Este se usaba para predecir eventos astrológicos y eventos importantes, facilitándole a los antiguos esta tarea. Esto demuestra la inquietud de nuestros antecesores en automatizar tareas y facilitar su desarrollo.

Pero sin duda uno de los avances que supuso un punto de inflexión para la posterior creación del microprocesador, fue el desarrollo de la máquina de Turing de mano del matemático británico Alan Turing, que sentó las bases de las ciencias de computación moderna, y dio lugar a investigaciones e invenciones posteriores, que empezaron a sentar la bases para la creación de esta tecnología.

Aunque todo esto no habría sido posible sin la posterior creación del transistor en los años 50, el que es considerado como el mayor avance del siglo XX. Ya que permitió a los investigadores la miniaturización de las computadoras digitales, que hasta la fecha se fabricaban con tubos de vacío, que eran lentos, voluminosos y frágiles.



Figura 1. Zuse Z4. (PFEIFFER, s.f.)

En la Figura 1, se muestra la computadora Zuse Z4 diseñada por el ingeniero alemán Konrad Zuse en 1945. Esta máquina es considera la primera computadora comercial de la historia y fue adquirida por la escuela politécnica de Zúrich. Estaba fabricada con tubos de vacío y relés, lo que hacía que fuera lenta comparada con las computadoras a transistores actuales. Aun así, este dispositivo era bastante útil, ya que era capaz de realizar cálculos complejos, que fueron de gran ayuda a la hora de diseñar el caza suizo FFA P-16.

Todo esto llevó a que en 1971 Intel creará el primer microprocesador de la historia, el Intel 4004, diseñado por Federico Faggin para una calculadora, la cual pasó de estar formada por 7 chips a solo uno.

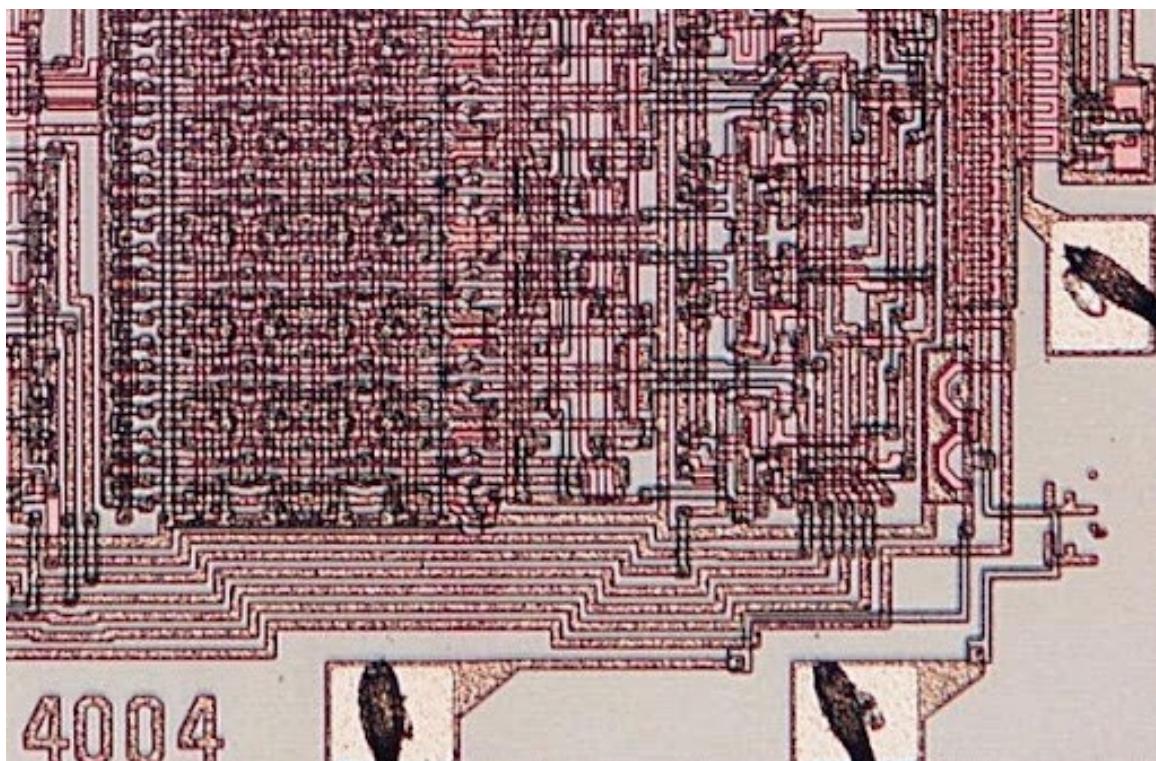


Figura 2. Circuito del Intel 4004 con la firma de Federico Faggin. (Intel 4004 , s.f.).

A pesar de la revolucionaria tecnología, Intel no apostó lo suficiente por esta tecnología, por lo que Faggin decidió marcharse y fundar su propia compañía, Zilog, en la que diseñó uno de los microprocesadores más emblemáticos de la historia el Zilog Z80, que supuso una revolución en la época, ya que era más barato y simple que los de la competencia. Este dispositivo contaba con nueve trampas en el diseño, que hicieron que no se pudiera copiar. Estos errores estaban tan bien implementados que en la actualidad solo se tiene constancia de 5 de ellos.

Aunque sin duda el chip que marcó un antes y un después fue el Intel 8086, el primer microprocesador de 16 bits de la historia, el cual empleó IBM para fabricar una computadora

doméstica. Además de establecer la arquitectura x86, que se usó en gran cantidad de ordenadores años después. Esto hizo que Intel se diera cuenta de los beneficios que podía generar esta industria, y empezara a apostar fuerte por esta nueva tecnología hasta convertirse en la compañía que conocemos en la actualidad.

Estos dispositivos sentaron las bases para la creación de los microporcesadores que se usan en la actualidad, pero también surgieron diseños que rompían con estos esquemas. Como es el caso de la arquitectura ARM, creada en 1985 por la empresa Acorn Computers, de la mano de Sophie Wilson y Steve Furber. Esta arquitectura es capaz de desconectar los módulos que no se están usando en ese momento para reducir el consumo del dispositivo. En la actualidad esta tecnología se usa en móviles inteligentes, ya que reduce el consumo e incrementa la duración de la batería.

Esta industria ha crecido a pasos agigantados, principalmente gracias a la mejora en los procesos de fabricación, que permiten crear transistores más pequeños. Siendo los actuales de tan solo 7 nm.

2.2. Funcionamiento.

Los microporcesadores están formados por distintos tipos de bloques básicos, que trabajan de manera sincronizada para realizar operación más o menos complejas. La mayoría de los microporcesadores cuentan con estas unidades:

- **ALU (Unidad Lógico Aritmética):** Es el “cerebro” del microporcesador, ya que se encarga de realizar las operaciones lógicas y aritméticas. Mediante una combinación de bit se selecciona la operación que se desea realizar, y este la ejecuta con los datos proporcionados. Los procesadores modernos cuentan con más de una de estas unidades, además tienen algunas únicamente dedicadas al procesamiento de números con coma flotante.
- **Buses:** Son las conexiones que enlazan cada uno de los módulos que componen el dispositivo.
- **Registros:** Pueden estar diseñados para un uso específico o para un uso general. En el caso de los registros de uso general, el usuario usarlos a su antojo para almacenar información que le va a servir en el futuro. Sin embargo, los de uso específico, cada uno tiene una función predeterminada por el fabricante, siendo los siguientes los más importantes:

- **PC (Contador de programa):** Este registro contiene la dirección de la memoria de programa, que se está ejecutando en ese instante. Esta dirección va incrementado a medida que se va ejecutando cada una de las instrucciones solicitadas por el usuario en el programa.
- **IR (Registro de instrucción):** En él se almacena el código de instrucción que se está ejecutando en ese momento. El código de instrucción es el dato, en binario, que indica la instrucción que debe realizar el dispositivo. Por ejemplo, una suma o guardar un dato en la memoria.
- **SP (Puntero de pila):** Indica la dirección de la memoria donde se encuentra la pila del procesador en ese instante. La pila del procesador es donde se almacenan el valor del contador de programa, que tenía el dispositivo cuando se ha producido una interrupción. De modo que, una vez atendida la interrupción, el procesador debe recuperar los datos con los que estaba trabajando. Por lo que los recupera de la pila del programa, donde los había guardado antes de ejecutar la interrupción y continua con la ejecución del programa.
- **PSW (Registro de estado):** Es un registro importante, que almacena datos que indican el estado de funcionamiento actual del procesador, como puede ser si las interrupciones son vectorizadas o autovectorizadas o si están enmascaradas.
- **Unidad de control:** Es la unidad más importante, ya que se encarga de sincronizar cada uno de los bloques que forman el dispositivo, para que realicen la operación deseada.

Una vez que se conocen las partes básicas de este tipo de dispositivo, se puede describir el funcionamiento normal de estos, siendo el siguiente:

- 1) Se lee el código de instrucción de la dirección de la memoria de programa que indica el registro PC, y se almacena en el registro de instrucción. Indicándole este código a la unidad de control, la instrucción que se desea ejecutar.
- 2) En caso de ser una instrucción que trabaje con datos facilitados por el usuario, se leen de las siguientes direcciones de la memoria que indica el registro PC y se almacenan en registros de usuario.
- 3) Una vez se han leído todos los datos, se comienza a trabajar con ellos para obtener el resultado final de la operación. En las operaciones lógicas y aritméticas, estos datos se le proporcionan a la ALU y realiza la operación que se le ha indicado.

- 4) Tras obtener el resultado final de la operación, se almacena donde desee el usuario, en la memoria o en los registros de uso general.
- 5) Se vuelve a realizar los pasos anteriores y se ejecuta una nueva instrucción.

Esta sería la ejecución normal del programa, en la mayoría de los microprocesadores se pueden llevar a cabo otros modos de funcionamiento que son los siguientes:

- **ADM:** En este modo, se desconectan los buses de la memoria del programa, deteniendo por tanto la ejecución, y dándole el control de la memoria a otro dispositivo para que haga uso de ella. Este modo, se puede usar para borrar un programa anterior y escribir uno nuevo programa en la memoria.
- **Inicio:** En él, se inicializan todos los registros de uso específico y se comienza de nuevo la ejecución del código del programa desde el principio. Es decir, es el proceso que se lleva a cabo cuando se reinicia el microprocesador.
- **Interrupción:** Se detiene la ejecución normal del código y se empieza a ejecutar una tarea que tiene más prioridad que el programa principal. Una vez termina esta tarea, continua con la ejecución del programa principal. Antes de comenzar a ejecutar la tarea, el dispositivo almacena el valor del registro PC en la pila del procesador, para recuperarlo una vez terminada la interrupción.

2.3. Características

Para poder comparar entre distintos microprocesadores y elegir el que mejor se adapte a una determinada situación, ha de tenerse en cuenta una serie de características que determinan su potencia y desempeño. Las principales características que hay que comparar para seleccionar el dispositivo que mejor se adapte a una determinada aplicación, son:

- **Frecuencia de reloj:** Los microprocesadores son dispositivos síncronos, es decir que trabajan bajo el mando de una señal que sincronizar cada uno de sus módulos. Esta señal es periódica y se genera por un oscilador o reloj, soliendo tener un periodo fijo. Esta característica es muy importante, ya que determina la velocidad del dispositivo, junto a otros parámetros que son más difíciles de conocer por el usuario. Por ejemplo, puede haber un dispositivo que trabaje a más velocidad de reloj que otro, pero sin embargo tarde más periodos de reloj en ejecutar sus instrucciones porque no están optimizadas. Esto haría que el dispositivo con más velocidad de reloj sea más lento en la práctica, aunque suele ser un caso excepcional, ya que los fabricantes se cercioran de optimizar lo máximo posible las instrucciones.

- **Juego de instrucciones:** Algunos dispositivos pueden contar con instrucciones específicas para una determinada aplicación. Es por ello por lo que se debe buscar un microprocesador que facilite la ejecución del programa deseado, sin la necesidad de concatenar un gran número de instrucciones para obtener el resultado deseado. En el caso de que se desee trabajar con señales digitales, existen dispositivos que cuentan con instrucciones específicas para tratar este tipo de señales. Son los conocidos como DSP o “Procesadores de señales digitales”.
- **Tamaño del bus de datos:** Determinará la cantidad de bits con los que es capaz de trabajar el dispositivo, sin necesidad de anidar instrucciones. Por lo que, si los datos que va a manejar el microprocesador son como máximo de 16 bits, sería suficiente con usar un dispositivo de 16 bits, ya que uno de mayor tamaño de bus será más costoso y no se le va a sacar su máximo partido.
- **Tamaño del bus de direcciones:** Establece la cantidad de datos que podrá tener la memoria del programa, limitando el tamaño máximo del programa que va a ejecutar.
- **Número de núcleos:** En la actualidad hay en el mercado gran cantidad de microprocesadores que cuentan con más de un núcleo de procesamiento. Lo que les otorga mayor capacidad de cómputo, ya que pueden realizar varias tareas en paralelo. Esto, junto con la velocidad de reloj son las características principales que determinarán la velocidad de ejecución del programa.

Si la aplicación para la que se va a utilizar no requiere ejecutar un gran número de instrucciones en un corto periodo de tiempo, sería conveniente elegir un microprocesador con un núcleo, ya que será más económico.

2.4. Tipos

Los microprocesadores se pueden catalogar de distintas formas, una de ellas es en función de su juego de instrucciones, siendo estos dos tipos los más comunes:

- **RISC (Ordenador con Conjunto Reducido de Instrucciones):** Este tipo de arquitectura se caracteriza por contar con un número reducido de instrucciones, que suelen ser de un solo byte. Por lo que, para poder realizar una operación compleja, es necesario concatenar varias instrucciones. Esto hace que se necesiten menor número de instrucciones en su fabricación, y por tanto se reduzcan los costes de fabricación y el consumo de energía que tiene.

- **CISC (Computación con conjunto de instrucciones complejas):** El dispositivo cuenta con un amplio número de instrucciones, que le permiten realizar operaciones complejas con solo una instrucción.

Otra forma para catalogar los microprocesadores es en función de la arquitectura usada, de las cuales las más conocidas son las siguientes:

- **x86:** Es la arquitectura CICS usada en la actualidad por la gran mayoría de ordenadores, esta fue establecida por Intel en el procesador 8086 del que se ha hablado anteriormente. Desde sus inicios ha ido evolucionando y aumentando el número de bits que es capaz de manejar, pasando de trabajar con datos de 16 bits a hacerlo con datos de 64 bits en los ordenadores actuales.
- **ARM (Maquinas Acorn RISC):** Es la arquitectura RISC que predomina en dispositivos móviles, aunque en los últimos años se está implementando en los ordenadores también, todo esto debido a su eficiencia.

También se pueden diferenciar por su manera de almacenar los datos en la memoria, en los que destacan dos tipos:

- **Arquitectura Harvard:** Se caracteriza por contar con dos memorias de programa, en una de ellas se almacenan las instrucciones que se van a ejecutar y en la otra los datos de los que hacen uso estas instrucciones. Es la menos común en la actualidad a pesar de su elevado rendimiento, debido a que es más compleja de implementar.
- **Arquitectura de Von Neumann:** A diferencia de la anterior, solo cuenta con una memoria, que contiene tanto los códigos de instrucción como los datos. Es la preferida por los fabricantes y la más usas en la actualidad, por su versatilidad y sencillez frente a la arquitectura de Harvard.

Estas son algunas de las diferencias en la construcción que se pueden encontrar entre microprocesadores.

Capítulo 3. FPGA

3.1. Introducción.

Fueron inventadas en 1987 como una evolución de los CPLD. La FPGA (Field Programmable Gate Array), denominada en castellano como Matriz de Puertas Lógicas Programables en Campo, es un circuito integrado que tiene la capacidad de variar las conexiones internas de cada uno de sus bloques lógicos, que se encargan de realizar operaciones lógicas simples. La ventaja que esto supone es que permite el procesamiento en paralelo, ya que de este modo se crea un circuito electrónico interno, que realiza la función deseada. Permitiendo así replicar circuitos electrónicos de gran complejidad. No como ocurre en el caso de los microcontroladores, que tienen una estructura predefinida y deben ejecutar el código secuencialmente para obtener el resultado deseado.

La forma de describir el circuito, que se desea implementar, es mediante un lenguaje de programación específico, que se conoce como HDL (Lenguaje de descripción de hardware) donde destacan VHDL y Verilog.

3.2. Estructura

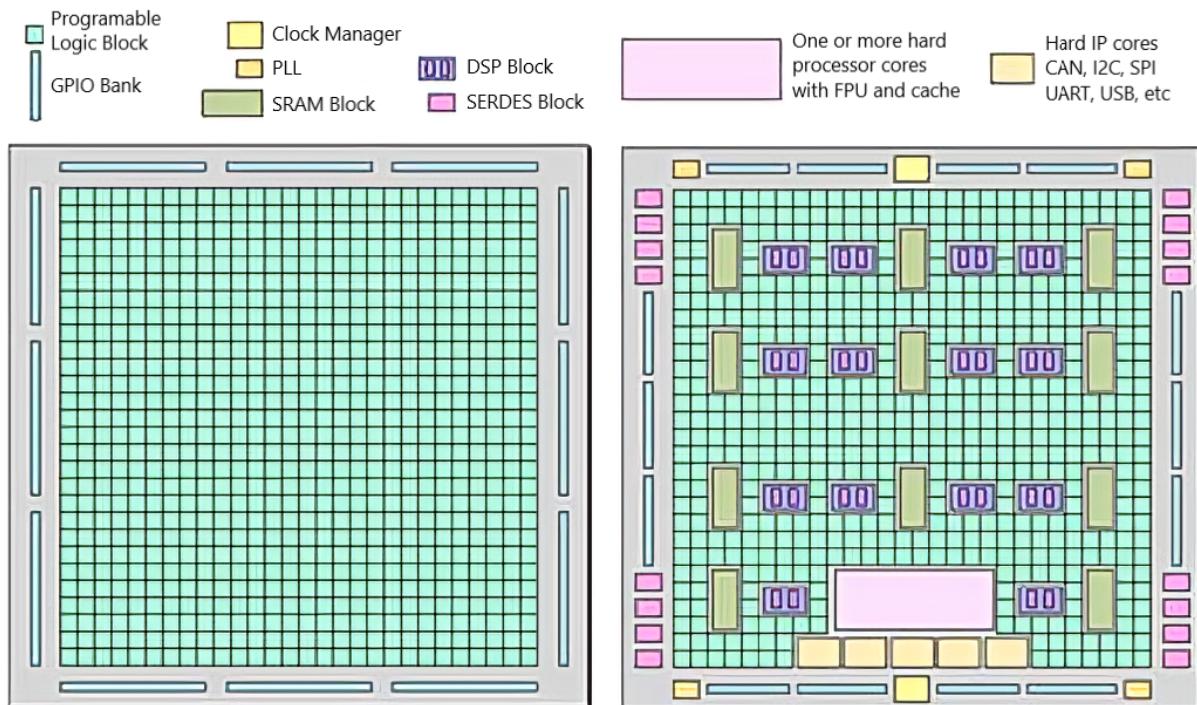


Figura 3. Evolución de la arquitectura de las FPGAs. (Maxfield, s.f.)

La estructura interna de las FPGAs ha cambiado desde sus inicios, como se aprecia en la Figura 3, contando actualmente con gran cantidad de unidades destinadas a usos específicos. Que le otorgan una mayor versatilidad y potencia. La gran mayoría de FPGAs del mercado cuentan con los siguientes bloques:

- **Look-Up Table (LUT):** Son las unidades básicas de una FPGA, ya que otorgan la posibilidad de implementar funciones lógicas de “n” variables Booleanas. El tamaño del LUT viene determinado por el número de entradas “n”, siendo el tamaño de 2^n . En la Figura 4 se puede observar la estructura de estas unidades, que están formadas por multiplexores que seleccionan, en función de las entradas, un valor preestablecido por el usuario para cada una de las combinaciones posibles.

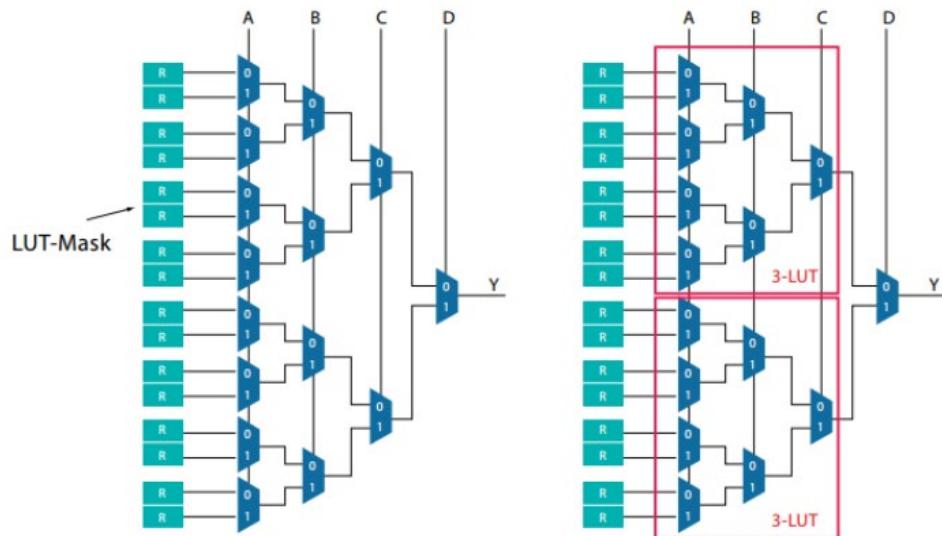


Figura 4. Estructura de los LUTs. (Diego)

- **Flip-Flops (FF):** Son biestables que se encargan de registrar datos, generalmente las salidas.
- **BRAMS:** Bloques de memoria RAM integrados en la arquitectura para reducir el uso de LUTs. Generalmente son de doble puerto, lo que le otorga la característica de poder leer y escribir un dato en el mismo ciclo.
- **Procesadores:** Son los bloques más complejos con los que puede contar, ya que forman un procesador completo con cache. Esto permite trabajar con un procesador sin la necesidad de diseñarlo, reduciendo así el uso de bloques básicos.
- **Bloques de procesamiento de señales digitales (DSP):** Es una unidad lógico-aritmética (ULA), diseñada para trabajar con señales digitales, embebida en el núcleo de la FPGA. En la marca Xilinx se conoce como DSP48, y tiene la estructura que se puede ver en la Figura 5.

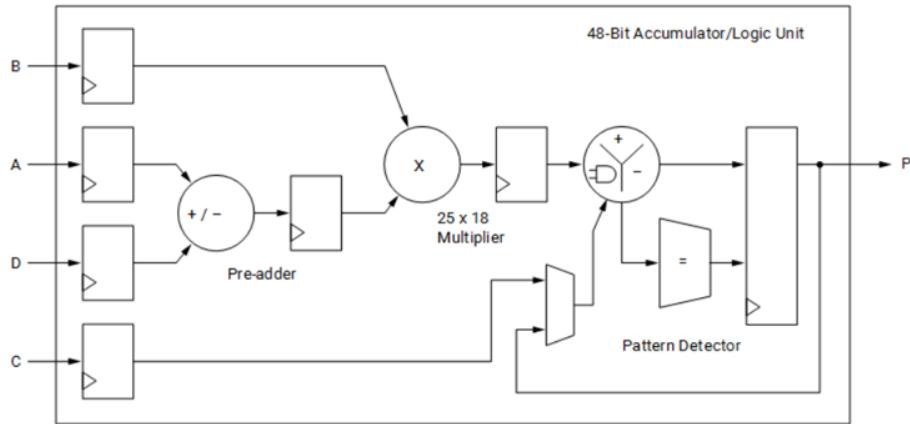


Figura 5. Esquema DSP48. (AMD, s.f.).

- **Buces de bloqueo de fase (PLL):** Es un circuito realimentado que es capaz de determinar el desfase existente entre dos señales, permitiendo así controlar la frecuencia y fase de una onda suministrada por un ‘Oscilador Controlado por Tensión’. Esto otorga la posibilidad de trabajar con distintas frecuencias de reloj en el núcleo de la FPGA, teniendo solo un oscilador de cuarzo en la placa. El esquema de este circuito es como el mostrado en la Figura 6.

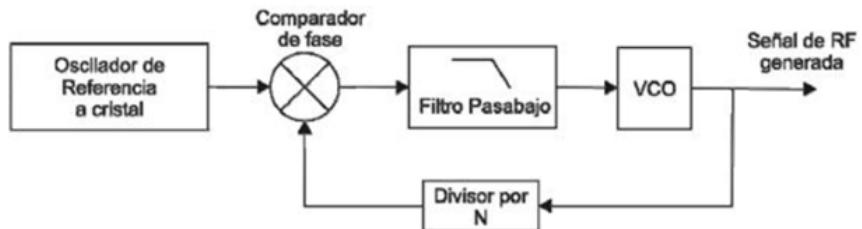


Figura 6. Esquema de un PLL.

- **Gestor de relojes de modo mixto (MMCM):** Genera señales de reloj con distinta frecuencia, como en el caso de los PLLs, pero en este caso un solo bloque es capaz de generar más de una señal. La desventaja es que son menos precisos que los PLLs.

3.3. VHDL

El VHDL es un lenguaje de programación de descripción de hardware, que se usa para describir circuitos electrónicos. Este se establecido por el IEEE en 1993, para programar FPGAs y CLPs principalmente. Junto al Verilog son los lenguajes más usados en la actualidad para programar FPGAs, aunque en este trabajo se ha optado por el VHDL, debido a su simplicidad y la cantidad de recursos que se pueden encontrar en la web.

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.NUMERIC_STD.ALL;
5  use WORK.ALL;
6
7
8  ENTITY Ejemplo_Implementacion IS
9      --Declaración de entrada y salidas del componente diseñado
10     PORT( Entrada : in STD_LOGIC;      --Entrada de una sola señal
11            BUS_Estrada : in STD_LOGIC_VECTOR (7 downto 0);  --Bus de entrada de 8 bits
12            Salida : out STD_LOGIC;      --Salida de una sola señal
13            BUS_Salida : out STD_LOGIC_VECTOR (7 downto 0)    --Bus de salida de 8 bits
14        );
15 END Ejemplo_Implementacion;
16
17 ARCHITECTURE Behavioral OF Ejemplo_Implementacion IS
18     --Declaración de señales que se usan internamente, no tiene un pad del IC asignado
19     SIGNAL SENAL_INTERNA: STD_LOGIC;      --Señales
20
21 BEGIN
22     --Aqui se declara el funcionamiento del circuito
23
24     Salida <= '0';                      --Asignación de un valor a una salida, 0 lógico.
25     SENAL_INTERNA <= '1';                --Asignación de un valor a una variable, 1 lógico.
26     BUS_Salida <= "11111111";          --Asignación de un valor a un BUS, 1 lógico a todas.
27
28     Triestado: PROCESS(Entrada, BUS_Estrada)
29         VARIABLE VAR1: STD_LOGIC_VECTOR (7 DOWNTON 0); --Variable solo para el proceso
30     BEGIN
31         VAR1 := "00000000";   --Asignación de un valor a una Variable.
32
33         --Si se produce un flanco ascendente en la señal ENTRADA
34         IF(rising_edge(Entrada))THEN
35             --La señal SENAL_INTERNA está a nivel bajo
36             IF(SENAL_INTERNA = '0')THEN
37
38                 BUS_Salida <= BUS_Estrada;  --BUS de salida igual al BUS de entrada
39
40                 --La señal SENAL_INTERNA está a nivel alto
41                 ELSIF(SENAL_INTERNA = '1')THEN
42
43                     BUS_Salida <= (OTHERS => 'Z');  --BUS de salida en alta impedancia
44
45             END IF;
46         END IF;
47     END PROCESS Triestado;
48
49 END Behavioral;

```

Figura 7. Ejemplo de código en VHDL.

Para hacer una pequeña introducción a la programación VHDL, se va a comentar el código que se ve en la Figura 7, que tiene la estructura básica de este tipo de programas. La estructura es la siguiente:

- **Líneas 2-5:** Al principio del código se incluyen las bibliotecas que se van a utilizar. En este caso se usan las bibliotecas del IEEE, para las declaraciones de las señales (STD_LOGIC_1164) y para trabajar con valores numéricos tipo integer o unsigned (NUMERIC_STD). Por último, se usa la librería “WORK” que contiene los componentes diseñados en otros archivos del proyecto.
- **Líneas 8-15:** Se declara el componente que contiene el archivo y se va a diseñar, en este caso se llama “Ejemplo_Implementación”. En el apartado “PORT” se enumeran las señales de entrada y salida que tiene este componente, pudiendo ser buses de varias señales.

- **Línea 17:** Se comienza a establecer la arquitectura “Behavioral” del componente “Ejemplo_Implementación”.
- **Líneas 18-20:** Entre “Architecture” y “Begin” se pueden enumerar señales que vaya a usar la arquitectura internamente, pero no sean de entrada o salida como en el caso de “PORT”.
- **Líneas 24-26:** Se asignan valores lógicos a las salidas y las señales, en caso de las variables se hace con “:=” y en el caso de las salidas y señales con “<=”.
- **Línea 28-47:** Comienza un proceso que tiene lugar en el componente. En este caso, se produce un triestado en la salida “BUS_Salida” cuando sucede un flaco ascendente en la entrada “Entrada” y la señal “SENAL_INTERNA” está a nivel alto, en caso contrario la salida sería igual a “BUS_Estadia”.

Estas serían las características principales de este tipo de programación, con estos conceptos básicos se pueden diseñar la mayoría de los componentes del proyecto. Ya que en este proyecto se diseñan todos los componentes, del microprocesador, en archivos separados y luego se unen para formar el microcontrolador.

```

1  ARCHITECTURE Behavioral OF Ejemplo_Implementacion IS
2
3  BEGIN
4      --Conexion de la Unidad de Control
5      UC: ENTITY Unidad_Control
6          PORT MAP(LCB_IN(0) => LCB_IN(0),
7                      LCB_IN(1) => CK,
8                      RST        => RST,
9                      BUS_OUT    => OUT_BUS
10                     );
11
12 END Behavioral;

```

Figura 8. Ejemplo conexión de componentes.

En el caso de querer añadir componentes diseñados en otros archivos y conectarlos, la estructura seria la que se muestra en la Figura 8. En ella se conecta el componente “Unidad_Control”, siendo este nombre el que se le ha dado en el archivo de origen, y se renombra como “UC” en el archivo actual. Después con la función “PORT MAP” se establece la conexión de las entradas y salidas del componente. A la derecha se coloca el nombre de la entrada o salida del componente, seguido de “=>” y a continuación se nombra la señal a la que se conecta.

3.3.1. Constrain.

El archivo constrain, también conocido como XDC, se encarga de establecer la conexión de los componentes con los pines de la FPGA, además de decretar otras normas para la implementación del código en el circuito integrado.

Este archivo no es necesario que lo redacte el usuario, ya que se puede obtener en internet buscando el nombre de la placa de desarrollo, que se utiliza en el proyecto, seguido de “Master”. Por ejemplo, en el caso de este trabajo, se usa la placa de desarrollo “CMOD A7”, por lo que buscando “CMOD A7 Master” se puede encontrar el contenido de este archivo, que solo habría que retocar un poco para adaptarlo al proyecto.

```

1 ## 12 MHz Clock Signal
2 #set_property -dict { PACKAGE_PIN L17 IOSTANDARD LVCMOS33 } [get_ports { sysclk }];
3 #create_clock -add -name sys_clk_pin -period 83.33 -waveform {0 41.66} [get_ports {sysclk}];
4
5 ## LEDs
6 #set_property -dict { PACKAGE_PIN A17 IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
7 #set_property -dict { PACKAGE_PIN C16 IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
8 #set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
9
10 ## RGB LED
11 #set_property -dict { PACKAGE_PIN B17 IOSTANDARD LVCMOS33 } [get_ports { led0_b }];
12 #set_property -dict { PACKAGE_PIN B16 IOSTANDARD LVCMOS33 } [get_ports { led0_g }];
13 #set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33 } [get_ports { led0_r }];
14
15 ## Buttons
16 #set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33 } [get_ports { btn[0] }];
17 #set_property -dict { PACKAGE_PIN B18 IOSTANDARD LVCMOS33 } [get_ports { btn[1] }];

```

Figura 9. Ejemplo de archivo XDC.

En la Figura 9 se puede apreciar un extracto de este código, en el que solo habría que modificar las entradas y salidas para indicar las que tiene nuestro proyecto. Esto se haría sustituyendo el texto que se encuentra entre corchetes al final de cada línea, por ejemplo “led[0]” en la línea 6. Además, sería necesario eliminar la almohadilla al principio de las líneas que se usan.

3.3.2. IP Cores.

Estos son componentes diseñados por el programa Vivado de Xilinx, que se pueden utilizar para implementar componentes complejos de diseñar. En el caso de este trabajo, se usan varios IP Cores, para simplificar el código de programa y también aligerar el uso de componentes de la FPGA.

Se ha usado un Core que se llama “Clocking Wizard”, que sirve para modificar la frecuencia y fase de una señal de reloj, para tener de este modo varias frecuencias de reloj en el Chip. Este componente hace uso de los PLLs que se han comentado anteriormente.

Otro de los bloques de los que se ha usado es el “Block Memory Generator”, que como su nombre indica se usa para generar memorias. Este tiene la posibilidad de establecer la capacidad, tipo, contenido, el control de la memoria, etc. Con este tipo de componente se puede

reducir el uso de bloques lógicos de la FPGA, ya que hace uso de las memorias que tiene el dispositivo integradas en el Chip. Por ejemplo, en el caso de este trabajo se usa una memoria ROM de bastante capacidad que, si se implementaría mediante código en VHDL, esta ocuparía la mayoría de LUTs con los que cuanta la FPGA. Además de invertir una gran cantidad de tiempo y recursos del ordenador en crear la conexión de estos bloques lógicos.

3.3.3. Archivo COE.

Este tipo de archivo se usar para declarar los datos que alberga una memoria en Vivado, y tiene la estructura que se observa en la Figura 10. Donde la primera línea define la base de los datos, en este caso se usa base 16 porque son datos en hexadecimal. Y en las siguientes líneas se colocan los datos que contiene la memoria en orden, desde la primera dirección hasta la última, separados por un espacio o coma. Como se aprecia, los datos que almacena esta memoria son de 96 bits.

Figura 10. Estructura básica de un archivo COE.

3.4. Placa de desarrollo CMOD A7

Para este proyecto es necesarios contar con al menos 42 pines de conexión en la placa desarrollo, por lo que se ha elegido el modelo CMOD A7 de Digilent, que tiene hasta 56 terminales de conexión, lo que es más que suficiente. Aunque para probar cada uno de los bloques que se han diseñado para el dispositivo, se ha usado el modelo BASYS 3 de la misma marca, ya que cuenta con interruptores y LEDs que facilita la tarea de probar los códigos.

A pesar de ser dos modelos distintos, las dos usan el mismo modelo de FPGA, el chip ARTIX 7 de Xilinx. La única diferencia, es que una de ellas usa las conexiones con la que cuenta este chip para conectar interruptores, LEDs u otros periféricos, para que sea más interactiva. Estos modelos se pueden ver en la Figura 11, siendo la de la izquierda la placa BASYS 3 y la de la derecha el modelo CMOD A7.

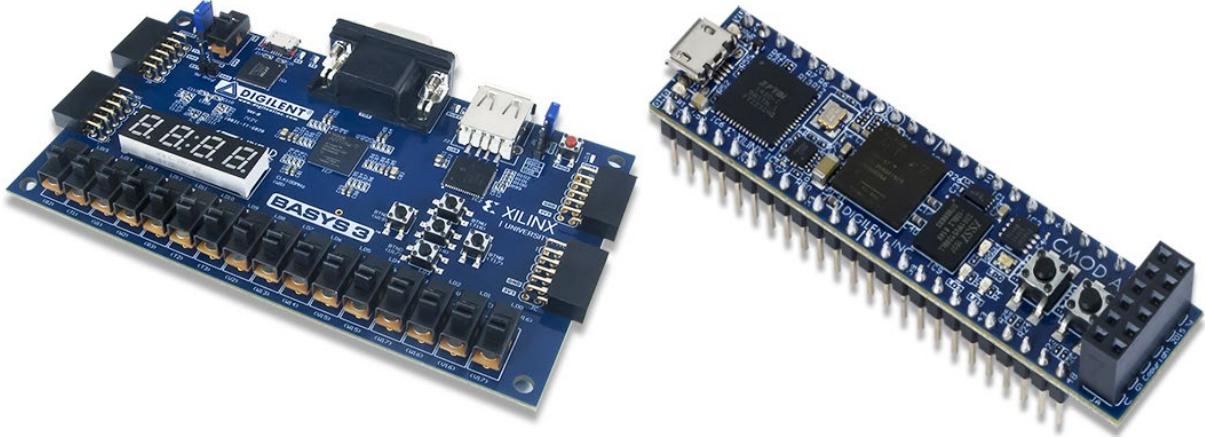


Figura 11. Placas de desarrollo BASYS 3 y CMOD A7 respectivamente.

En concreto el modelo de FPGA que usan estas placas es el “XC7A35T-1CPG236C” de la familia Artix-7 de Xilinx. Las características de este chip son:

- **Celdas lógicas:** Cuenta con 33280 celdas lógicas, esta es una unidad que se utiliza en el mundo de las FPGAs para establecer su potencia de computación, y así poder compararla con otros modelos que tienen estructuras distintas. Por ejemplo, el modelo de FPGA más potente en la actualidad, de la marca Intel, tiene 10,2 millones de celdas lógicas.
- **Bloques lógicos configurables (CDLBs):** Este dispositivo cuenta con 2600 de estos bloques, que están divididos en 2 partes, denominadas “slices”. Estos “slices” están compuestas por 4 Look-Up Tables, 8 biestables y multiplexores, la estructura se puede ver en la Figura 12.

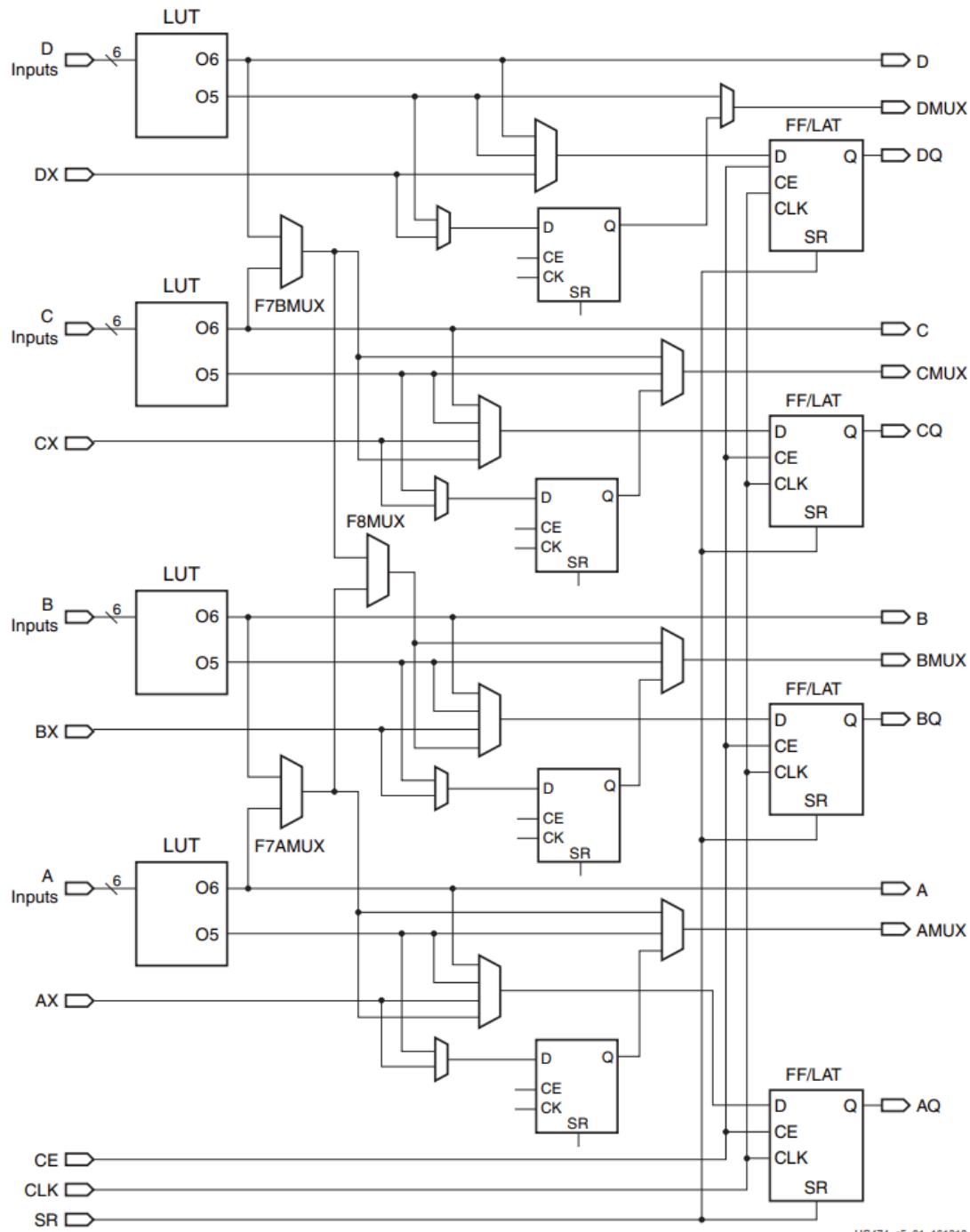


Figura 12. Estructura de un "Slice" de Artix 7

- **Bloques de RAM:** Tiene una capacidad de 1800 kb divididos en bloques de 36 kb de hasta 72 señales y pudiendo configurarse como FIFO.
- **Unidad de control de reloj:** Tiene 5 de estas unidades, que están formadas por 1 MMCM y 1 PLL.

3.4.1. Interfaz USB a UART.

Estas placas de desarrollo cuentan con un controlador que convierte datos enviados por el puerto USB a protocolo UART. Otorgándole así la posibilidad de enviar datos desde el ordenador hasta la FPGA mediante un puerto COM, y este CI convierte de protocolo USB a UART que es más fácil de interpretar y por ende de diseñar un controlador UART. El esquema de conexión de esta interfaz está presente en la Figura 13.

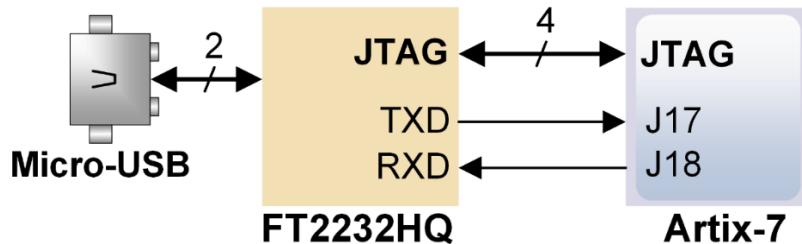


Figura 13. Interfaz USB a UART. (Digilent, 2019).

El protocolo UART envía los datos con la forma que se ve en la Figura 14 y muestrea, se lee, cada uno de los bits en los flancos de subida de la señal “Muestreo”. Es decir, los bits se leen a mitad del tiempo que dura el bit o lo que es lo mismo, a mitad de su periodo. Para evitar así posibles errores de lecturas debidos a la capacitancia de las entradas, que crea rampas de subida muy pronunciadas.

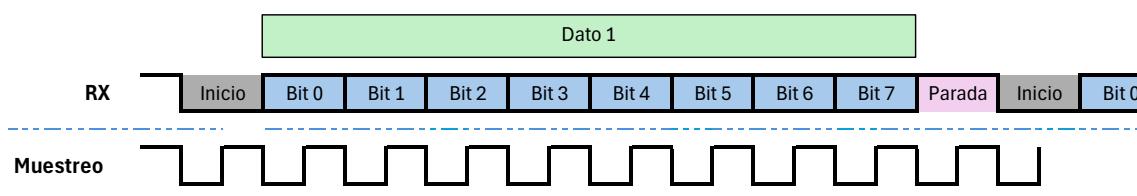


Figura 14. Cronograma de funcionamiento del protocolo UART.

Como se puede ver, la transmisión de un dato siempre comienza con un bit de inicio que genera un flanco descendente e indica el inicio de la transmisión, ya que las señales de envío y recepción están a nivel alto cuando están en reposo. Tras el bit de inicio se manda el dato en orden, empezando con el bit menos significado y terminando con el más significativo. Por último, una vez se ha transmitido todo el dato, se envía un bit de parada que siempre es a nivel alto, para generar un flanco descendente con el bit de inicio del siguiente dato que se mande, y así indicar que se está enviando un dato nuevo.

Capítulo 4. Implementación del HTDI20.

4.1. Estructura del microprocesador HTDI20 teórico.

Este dispositivo es un microprocesador teórico creado por un docente de la Universidad de Málaga, Pedro J. Sotorriño, que tienen la peculiaridad de que está diseñado a partir de componentes comerciales comunes. De modo, que el alumnado puede comprender su arquitectura al completo, ya que conocen el funcionamiento de cada uno de estos componentes de asignaturas anteriores. La estructura de este componente se expone en el libro “Desmitificando el microprocesador” (Sotorriño, 2021), en el cual se muestra como están formados cada uno de los bloques que lo componen.

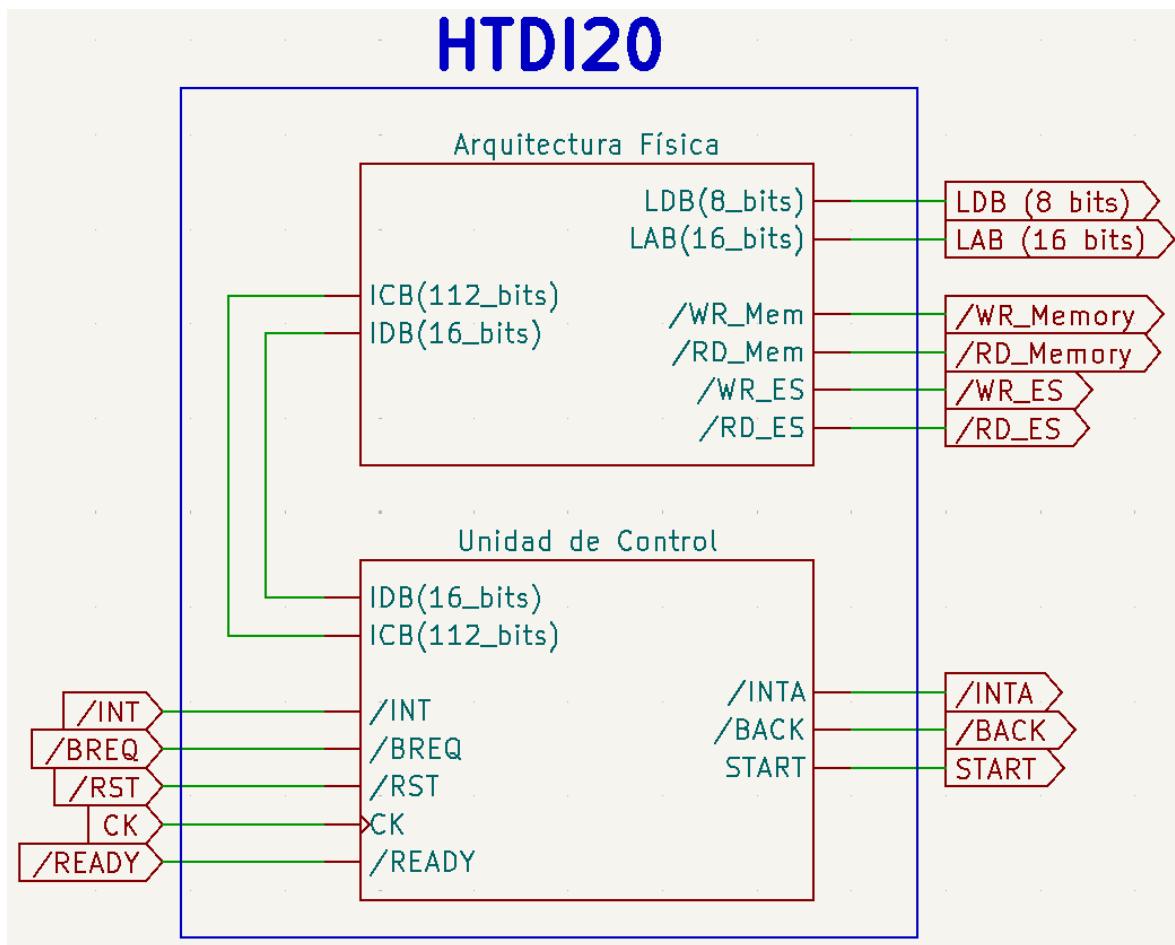


Figura 15. Esquema HTDI20 (bloques básicos).

El HTDI20 este compuesto por dos bloques principales que a su vez están formados por bloques más simples en funcionamiento, su estructura a grandes rasgos es la que se observa en la Figura 15.

4.1.1. Arquitectura física.

Esta entidad contiene todos los registros con los que cuenta el dispositivo además de contener la Unidad Lógico Aritmética (ALU). Su función principal es el manejo de todos los datos necesarios para que funcione el dispositivo y también los datos que necesita el usuario para hacer la función que quiere realizar. En la Figura 17 se muestra la estructura de esta unidad, donde los bloques principales que la componen son:

- **Registros de usuario:** Son 3 registros de 8 bits y 2 registros de 16 bits de carga asíncrona, y todos con salida triestado, en los que el usuario puede almacenar los datos que quiera. Los de 8 bits se llaman A, B y C y los de 16 bits D y E.
- **Registros de uso específico:** Estos son los ya mencionados PC (Contador de Programa) y SP (Puntero de Pila), además de incluirse dos registros más, el IX y el PSW (Registro de Estado). El registro IX está dedicado a contener una dirección de usuario y tiene el mismo funcionamiento que el registro contador de programa. Y el registro PSW contiene información importante para el funcionamiento del procesador, como puede ser los modos de interrupción o si se la operación que se ha realizado a dado como resultado cero o acarreo.
- **Registros auxiliares:** Suman un total de 8 registros triestado de carga asíncrona, donde 6 de ellos son de 8 bits y permiten seleccionar si se guarda el LSB o el MSB del bus de datos. Los otros 2 restantes son de 16 bits y tienen la capacidad de incrementar o decrementar al valor que tienen almacenado. Todos ellos solo pueden ser utilizados por las instrucciones y no por el usuario.
- **Registro de direcciones (AR):** Es el registro que contiene la dirección, del espacio de memoria o de entrada y salida, en la que se desea guardar o leer un dato. Es de 16 bits con carga asíncrona y salida triestado, para de este modo poder desconectar el HTDI20 del bus y permitir a otro dispositivo acceder a la memoria.
- **Registro de datos (DR):** Es bidireccional, permitiendo así leer o guardar datos en la memoria, además de ser triestado para desconectar el bus local de datos del HTDI20 cuando lo va a usar otro dispositivo.
- **Unidad lógico-aritmética (ULA):** Es el bloque que realiza todas las operaciones lógicas y aritméticas del dispositivo. Dos registros son los encargados de almacenar los datos con los que se desea operar y con 4 señales (ULA0-ULA3) se elige una de las 16 operaciones disponibles, siendo la combinación de estas señales la mostrada en la Tabla 1.

ULA3	ULA2	ULA1	ULA0	Operación	Tipo
0	0	0	0	Suma	Funciones Aritméticas
0	0	0	1	Resta	
0	0	1	0	Incremento	
0	0	1	1	Decremento	
0	1	0	0	OR	Funciones Lógicas
0	1	0	1	AND	
0	1	1	0	XOR	
0	1	1	1	NOT	
1	0	0	0	Desp. Lógico a izquierda	Desplazamientos y rotaciones
1	0	0	1	Desp. Lógico a derecha	
1	0	1	0	Desp. Aritmético a izquierda	
1	0	1	1	Desp. Aritmético a derecha	
1	1	0	0	Rot. Lógico a izquierda	
1	1	0	1	Rot. Lógico a derecha	
1	1	1	0	Rot. Aritmético a izquierda	
1	1	1	1	Rot. Aritmético a derecha	

Tabla 1. Operaciones de la Unidad Lógico Aritmética.

4.1.2. Unidad de control.

Es la encargada de monitorear el funcionamiento del dispositivo manejando cada una de las señales de control de los bloques que componen el microprocesador, de modo que no se produzca ningún error en la ejecución del programa. El esquema se encuentra en la Figura 18, como se puede apreciar cuenta con un gran número de bloques distintos el uno del otro, no como en el caso de la arquitectura física. Como el funcionamiento de todos ellos se commenta en el libro (Sotorriño, 2021), aquí solo se va a realizar una pequeña introducción a cada uno de ellos.

- **Sincronización de Reset (RST_SINC):** Este elemento se encarga de sincronizar la señal de Reset, que es una señal asíncrona, con el reloj del dispositivo para que trabaje en los mismos intervalos de tiempo que el resto de las señales.
- **Secuencial de la unidad de control (SUC):** Es uno de los bloques más importantes, ya que es el encargado de establecer en cuál de los 4 estados de ejecución (Inicio,

Búsqueda, Ejecución o Interrupción) se encuentra el microprocesador en cada instante.

- **Condicionales:** Es el responsable del correcto funcionamiento de las instrucciones condicionales, saltos y llamadas, determinando si se deja de ejecutar esa instrucción y no se produce el salto o llamada. O en caso de ser cierta la afirmación, se lleva a cabo el salto a otra línea del código o se llama a una función. Es un bloque muy necesario en cualquier microprocesador, ya que de este modo se pueden comparar datos y en función del resultado, realizar una acción u otra. Dicho de otro modo, con estas instrucciones se forman los comandos “If” o “else” de la programación en C.
- **Control de buses (CONT_BUSES):** Su función es la de poner los buses locales, es decir los externos al dispositivo, en alta impedancia para entrar en estado de ADM. Para que, de este modo otro dispositivo pueda hacer uso de los periféricos que tiene conectados, como puede ser la misma memoria del programa.
- **Control de la señal de interrupciones (Control_INT):** Lleva a cabo el procesamiento de la señal de interrupción. Primero sincroniza esta señal con la señal de reloj y en función de si se ha elegido las interrupciones por flanco descendente o nivel bajo, produce una señal que es interpretada por el SUC.
- **Selector de interrupciones (Selector_INT):** Como su nombre bien indica es el encargado de seleccionar entre los dos tipos de interrupciones posibles, las vectorizadas o las autovectorizadas. A este bloque le entra una señal que produce la SUC, cuando le llega la señal de interrupción y ha terminado la ejecución de la instrucción que estaba llevando a cabo. Cuando esta señal está activa, el selector elige entre las interrupciones vectorizadas o autovectorizadas, en función de lo que haya elegido el usuario al principio del programa.
- **Sincronización de la señal BREQ (Sinc_BREQ):** Cumple varias funciones, entre ellas la de sincronizar la señal BREQ, que es la señal de solicitud de los buses locales. También genera la respuesta a esta señal de entrada, indicando así al dispositivo que solicita la entrada en ADM, que ya tiene disponible los buses. Estas señales de salida son interpretadas por el bloque “CONT_BUSES” y se encarga de poner en alta impedancia los buses locales.
- **Registros de direcciones de rutinas de interrupción(IVEC y AUTO):** Estos dos registros determinan la dirección, de memoria del programa, donde se encuentra la rutina de interrupción en cada uno de los posibles casos. Siendo el “Registro_IVEC”

para las rutinas de interrupciones vectorizadas y el “Registro_AUTO” para las de las interrupciones autovectorizadas.

- **Vector de inicio (Circuito_Vect_Ini):** Tiene la misma función de los dos anteriores, pero en este caso otorga la dirección de memoria donde empieza el programa, que siempre es la dirección 0xFFFF0.
- **Controlador del reloj de la SUC (Deten_SUC):** Como se aprecia por su nombre, es el encargado de controlar la señal de reloj del Secuencial de la Unidad de Control”, generando flancos ascendentes cuando se termina de ejecutar una instrucción, para de este modo la SUC establezca el siguiente estado del microcontrolador. En el caso de que un dispositivo externo solicite la entrada en ADM, este bloque se encarga de detener el reloj del SUC para que no avance el programa mientras el dispositivo está en estado de ADM.
- **Unidad de microprograma:** Es el bloque más complejo de todo el dispositivo, ya que es el encargado de controlar 96 señales de control, jugando con ellas en cada instrucción para que realice la función deseada por el usuario. Los bloques que la componen se pueden ver en la Figura 16, el funcionamiento completo de este dispositivo se encuentra en el libro (Sotorriño, 2021), aunque más adelante se expondrá como se ha generado los códigos de programa.

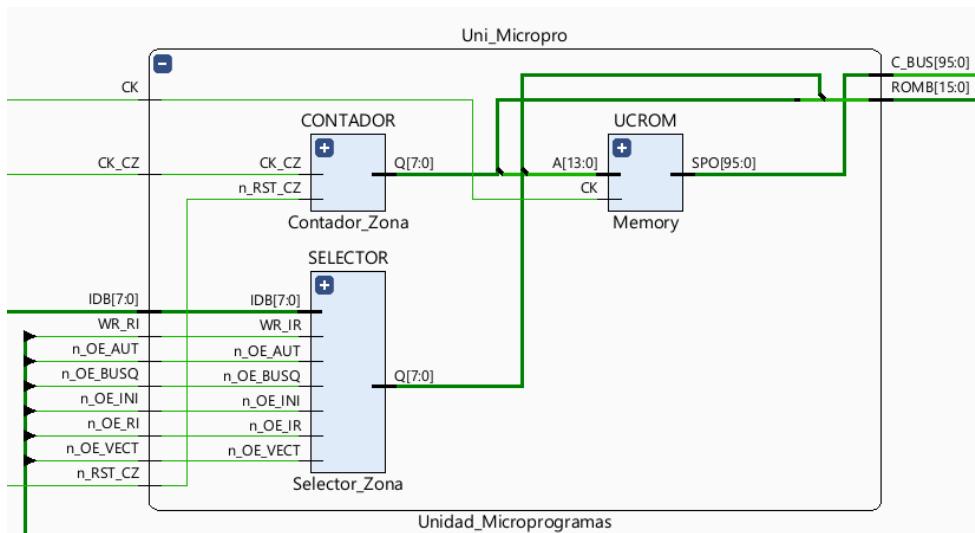


Figura 16. Esquema de la Unidad de Microprograma.

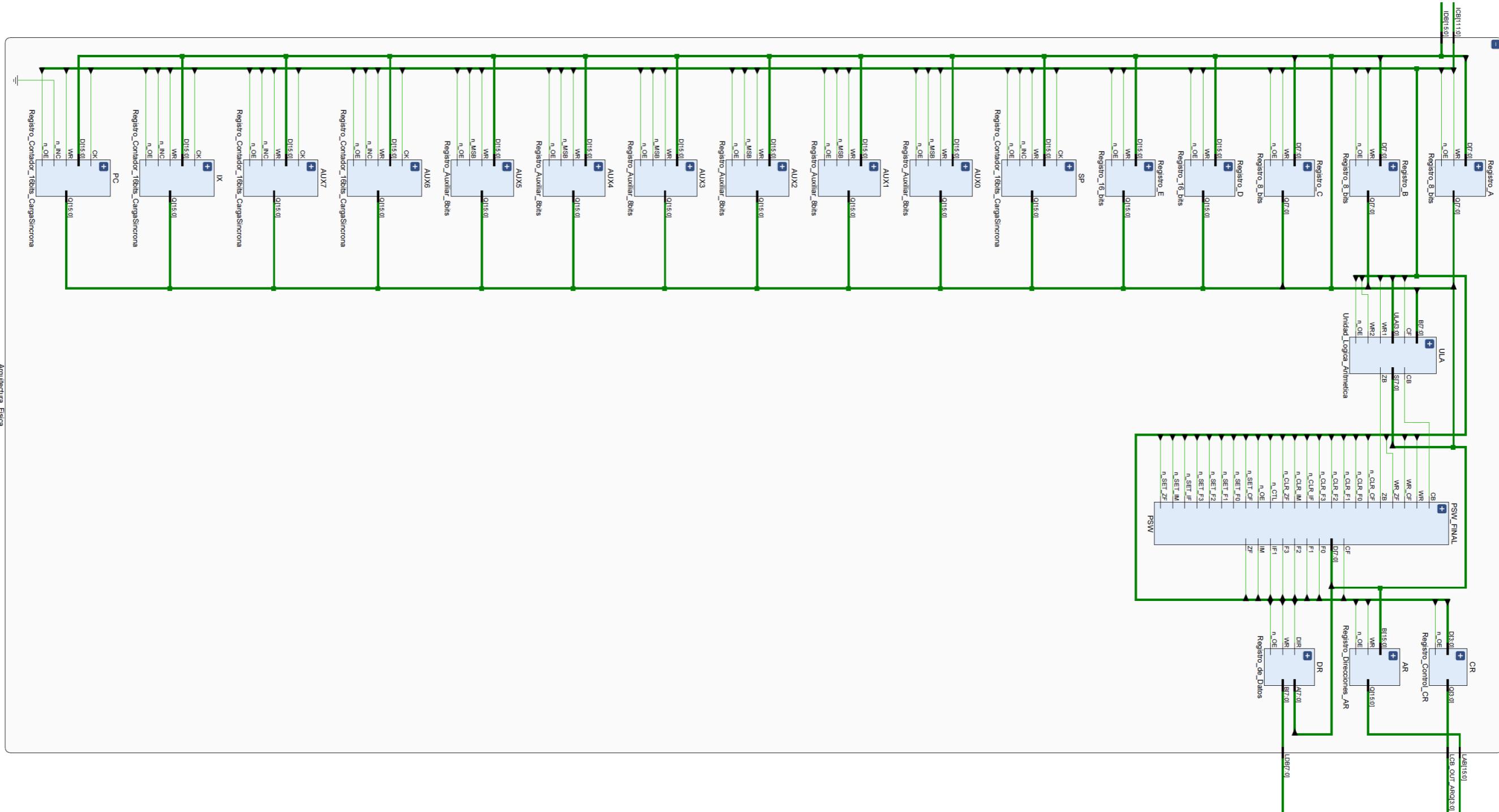


Figura 17. Estructura de la arquitectura física.

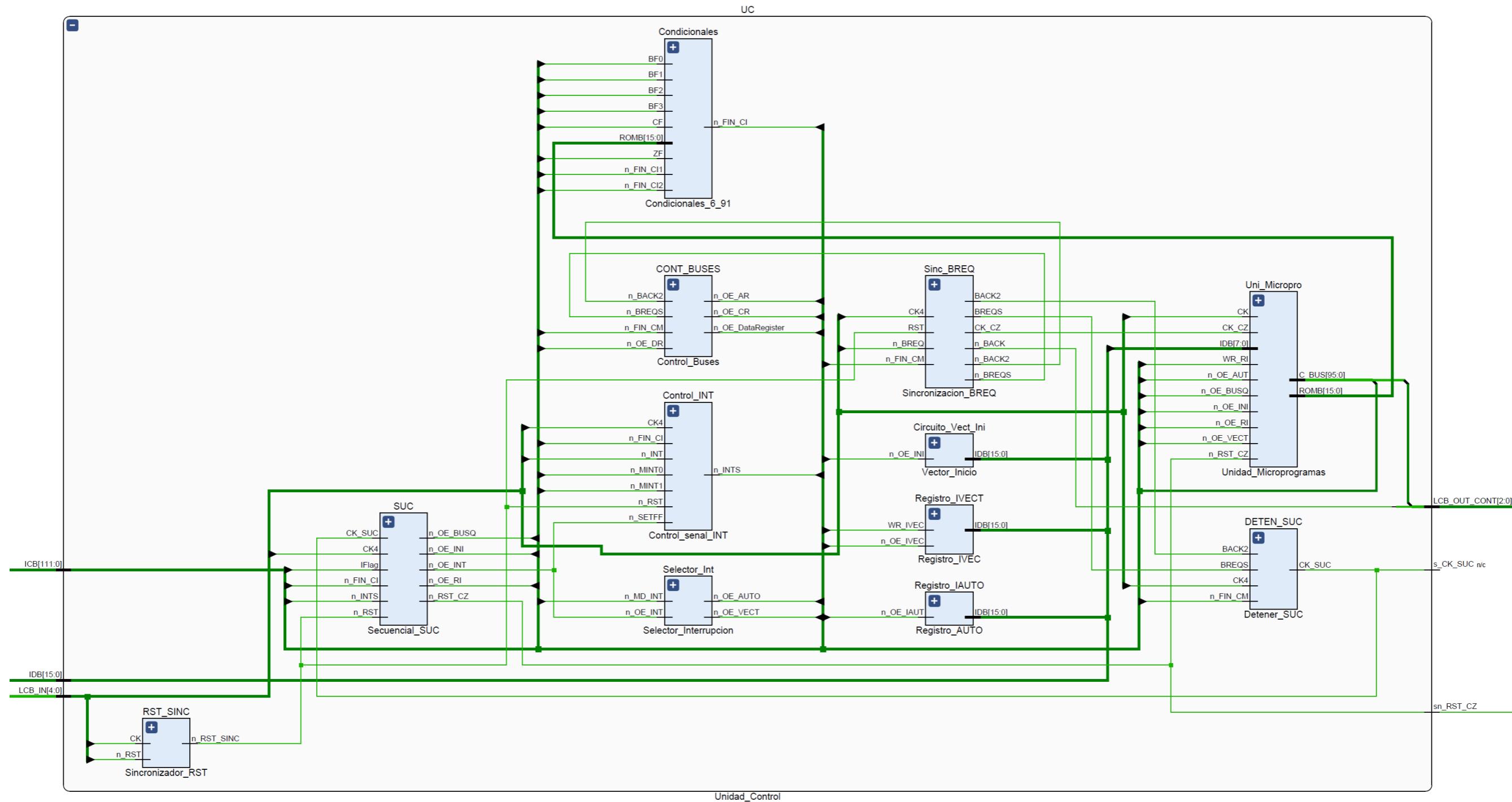


Figura 18. Estructura unidad de control.

4.2. Diferencias entre el HTDI20 teórico y el implementado.

A la hora de implementar el microprocesador teórico tal y como se muestra en el libro (Sotorriño, 2021), se ha tenido que cambiar el funcionamiento de algunos de sus bloques o eliminarlos para que sea posible su funcionamiento en una FPGA.

4.2.1. Multiplicador de frecuencia.

Se ha cambiado el bloque multiplicador de frecuencia, que es el encargado de generar la señal CK4 en el dispositivo, que es una señal de reloj con la frecuencia 4 veces la del reloj de entrada. Ya que, no es necesario tener una señal de reloj con la frecuencia multiplicada.

Pero este bloque no se elimina porque se usa con otra finalidad, y es la de tener varias velocidades de reloj para que el usuario puede elegir la frecuencia a la que trabaja el microprocesador, mediante un jumper instalado en la PCB. Se puede elegir entre cinco frecuencias de funcionamiento: 32 MHz, 16 MHz, 8 MHz, 4 MHz o una frecuencia de reloj que suministre el usuario por uno de los terminales del dispositivo, aunque esta señal no la procesa este bloque. Todas estas frecuencias se generan usando los PLLs y los MMCMs, con una señal base de 12 MHz que suministra la placa CMOD A7. Por lo que, el bloque pasaría a tener el aspecto de la Figura 19.

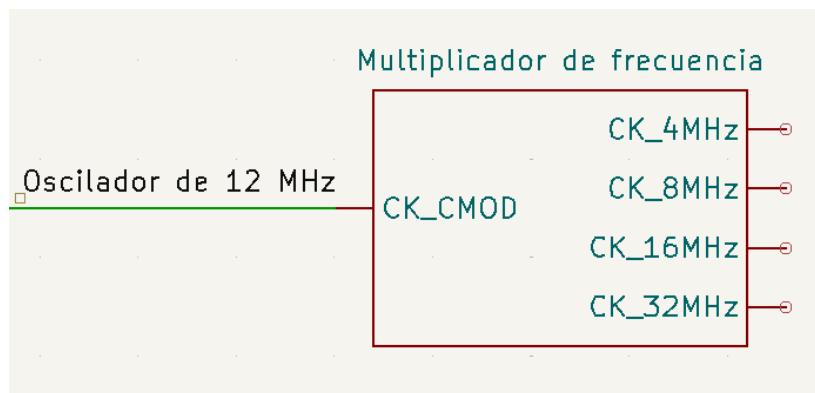


Figura 19. Multiplicador de frecuencias.

4.2.2. Registros contadores de 16 bits.

Los siguientes registros del dispositivo son de este tipo:

- Registro IX.
- Contador de programa (PC).
- Puntero de pila (SP).
- Registros Auxiliares de 16 bits (AUX6 y AUX7).

Todos estos elementos, como se comentaba anteriormente, son registros contadores de 16 bits con carga asincrónica. Que la carga sea asincrónica, ha sido una tarea imposible de realizar, por lo que se ha optado por cambiar a carga síncrona con la señal de reloj de entrada. De modo, que el bloque tiene el mismo aspecto, pero varía en funcionamiento. Siendo el funcionamiento el mostrado en la Figura 20, donde cada una de las etiquetas indica lo siguiente:

- Se carga el dato del bus (IDB) en el registro.** Esto se produce tras el flanco ascendente de la señal CK mientras la señal WR estaba a nivel alto.
- Se incrementa el valor del dato que contiene el registro.** Cuando se produce un flanco ascendente en el reloj con la señal WR a nivel bajo, se incrementa o decrementa el valor del registro en función del valor de /INC. En este caso, como la señal está a nivel bajo, se incrementa el contenido del registro.
- Se decrementa el dato guardado.** Como ocurre en el caso anterior, la señal de escritura está a nivel bajo cuando se produce un flanco ascendente en CK, por lo que se incrementa o decrementa. Esta vez se decrementa el valor del registro, porque la señal /INC está a nivel alto.
- Se vuelca el contenido del registro en el bus de datos (IDB).** Al ponerse la señal /OE a nivel bajo, se elimina el triestado de la salida del registro y el dato que almacena se ve reflejado en bus de datos. Se podría incrementar o decrementar el valor del registro mientras se encuentra en este estado.

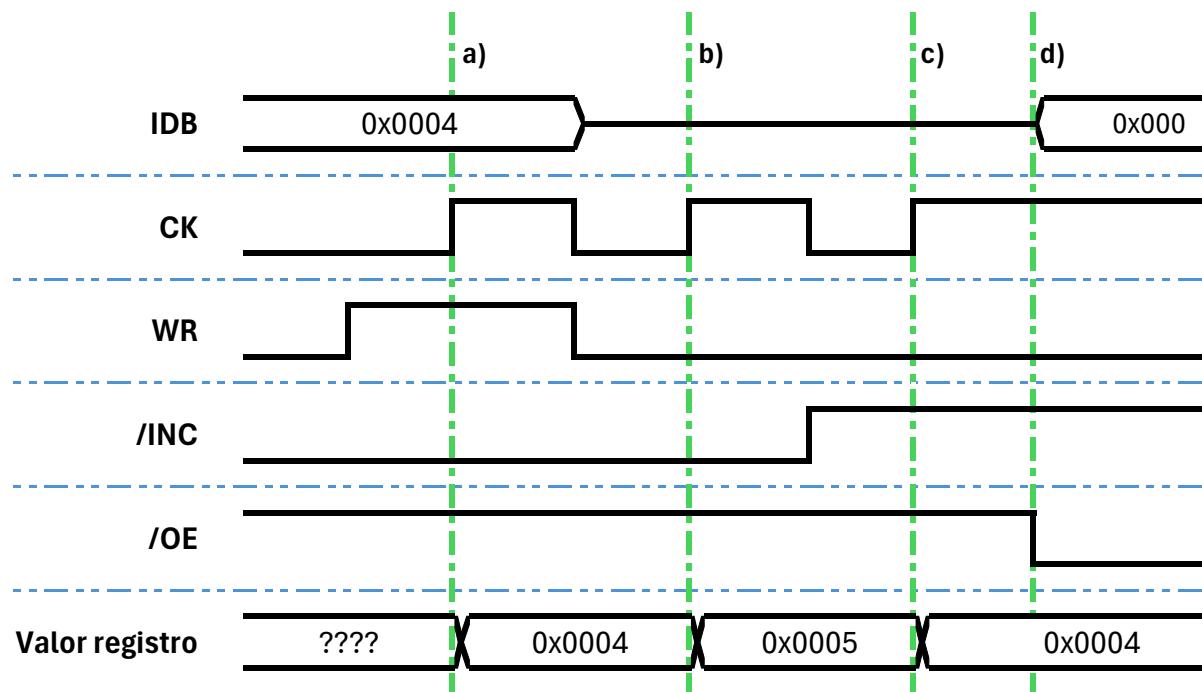


Figura 20. Cronograma de funcionamiento de los nuevos registros contadores.

4.2.3. Controlador de interrupciones.

Sería el circuito que se muestra en la Figura 5.2 del libro (Sotorrío, 2021). En este caso, se modificaría el biestable encargado de la señal /NINT, el denotado como B5, haciendo que ahora se sincronizo con la señal de reloj CK4. Por lo que, este biestable se sustituiría por un biestable tipo SR, manteniendo las otras partes del circuito igual. El circuito quedaría como en la Figura 21 y el cronograma de funcionamiento sería el presente en la Figura 22.

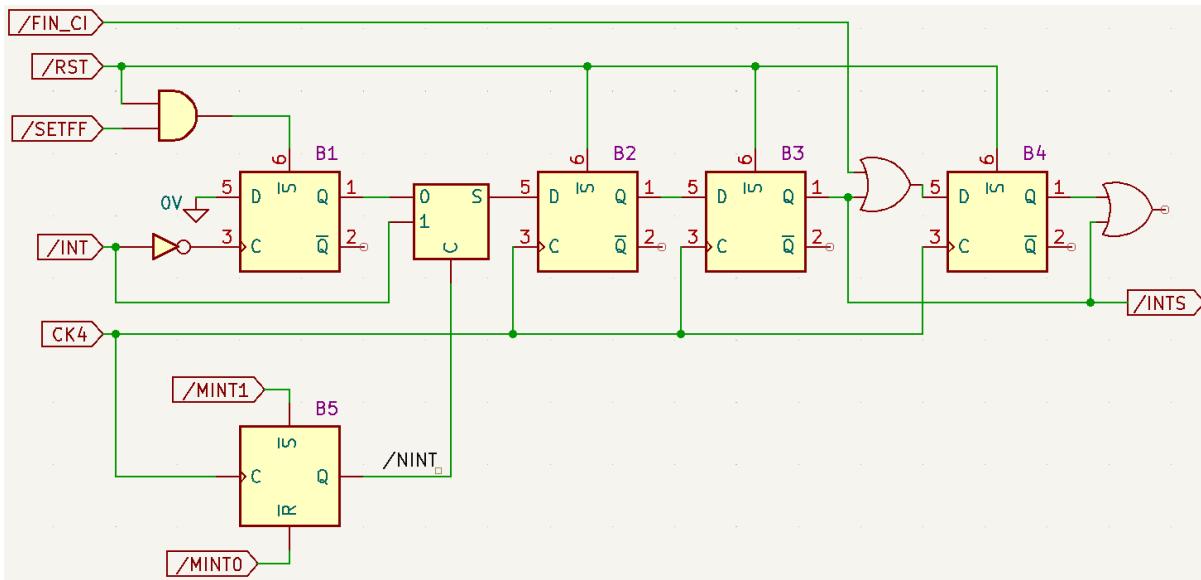


Figura 21. Circuito de control de la señal /INT.

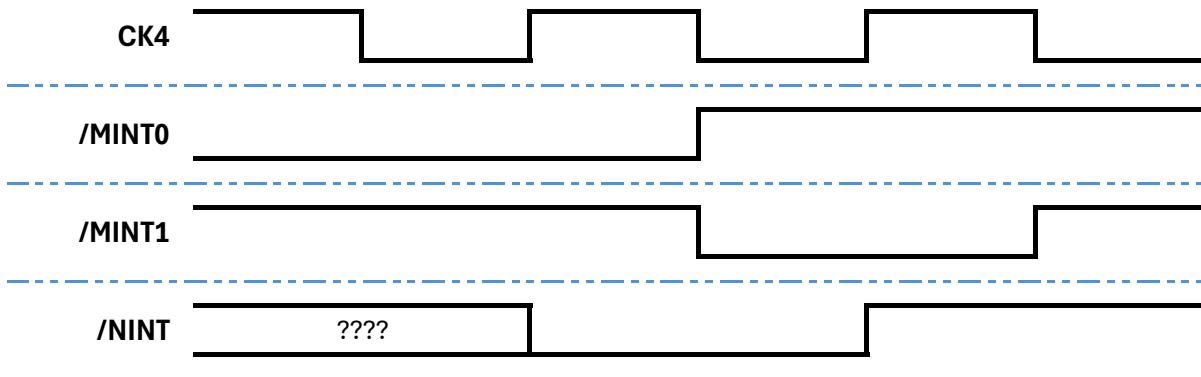


Figura 22. Cronograma de funcionamiento del controlador de interrupciones.

4.2.4. Memoria de la unidad de control (UCROM).

Para poder hacer uso de los bloques de memorias integrados en la FPGA, se ha tenido que cambiar varias partes de la memoria. Una de las consideraciones para tener en cuenta es que el número máximo de señales de salida es de 96, por lo que se ha tenido que eliminar algunas señales que no son imprescindibles, siendo la Tabla 2 las señales que se asignan a cada una de las salidas del dispositivo.

Pos.	Señal	Pos.	Señal	Pos.	Señal	Pos.	Señal
0	START	24	/INC_IX	48	/OE_AUX7	72	/CLR_F1
1	WR_AR	25	WR_AUX0	49	CK_AUX7	73	/SET_F2
2	WR_IR	26	/OE_AUX0	50	/INC_AUX7	74	/CLR_F2
3	/EN_RDY	27	/MSB_AUX0	51	/OE_A	75	/SET_F3
4	/FIN_CM	28	WR_AUX1	52	WR_A	76	/CLR_F3
5	/FIN_CI1	29	/OE_AUX1	53	/OE_B	77	/SET_IM
6	/FIN_CI2	30	/MSB_AUX1	54	WR_B	78	/CLR_IM
7	/OE_DR	31	WR_AUX2	55	/OE_C	79	/MINT0
8	WR_DR	32	/OE_AUX2	56	WR_C	80	/MINT1
9	DIR_DR	33	/MSB_AUX2	57	/OE_D	81	/CTL_PSW
10	/R_MEM	34	WR_AUX3	58	WR_D	82	/SET_IF
11	/W_MEM	35	/OE_AUX3	59	/OE_E	83	/CLR_IF
12	/R_ES	36	/MSB_AUX3	60	WR_E	84	ULA3
13	/W_ES	37	WR_AUX4	61	/OE_PSW	85	ULA2
14	CK_PC	38	/OE_AUX4	62	WR_PSW	86	ULA1
15	/OE_PC	39	/MSB_AUX4	63	WR_CF	87	ULA0
16	WR_PC	40	WR_AUX5	64	/SET_CF	88	WR_ULA1
17	CK_SP	41	/OE_AUX5	65	/CLR_CF	89	WR_ULA2
18	/OE_SP	42	/MSB_AUX5	66	WR_ZF	90	/OE_ULA
19	WR_SP	43	WR_AUX6	67	/SET_ZF	91	WR_IVEC
20	/INC_SP	44	/OE_AUX6	68	/CLR_ZF	92	/OE_IVEC
21	CK_IX	45	CK_AUX6	69	/SET_F0	93	/OE_INI
22	/OE_IX	46	/INC_AUX6	70	/CLR_F0	94	INTA
23	WR_IX	47	WR_AUX7	71	/SET_F1	95	/OE_IAUT

Tabla 2. Lista señales de salida de la UCROM.

Otro de los cambios que se ha debido de realizar, es el de disminuir el número de señales de direcciones, de 16 bits a tan solo 14 bits, ya que no se podía añadir más señales el bloque de memoria. Esto supone una desventaja, ya que no se pueden implementar instrucciones de más de 16 periodos de reloj, sin contar el ciclo de búsqueda que va en otra parte de la memoria. Esto no es algo crítico, ya que solo se han eliminado 5 instrucciones de usuario, que se pueden crear concatenando dos o más instrucciones. También se ha tenido que cambiar las instrucciones de las interrupciones vectorizadas y autovectorizadas, porque las vectorizadas necesitaban 20 periodos. Por lo que para poder usar las interrupciones se ha tenido que añadir un nuevo estado a la SUC, que se ejecuta justo antes del estado de interrupción. Y se ha creado una nueva instrucción básica que se ejecuta cuando el dispositivo se encuentra en este nuevo estado, que se encarga de guardar el valor del registro PC en la pila del procesador.

Disminuyendo de este modo el tiempo de ejecución de estas instrucciones, pero esto se expondrá de manera más detallada en apartados posteriores.

La razón principal para hacer uso de los bloques de memoria, de la FPGA, es que el tiempo que tarda el software Vivado en generar los archivos de salida se disminuye en gran medida. Sin hacer uso de estos bloques el tiempo que tardaba era de aproximadamente 8 horas, siendo ahora de tan solo de 2 minutos. Por lo que tiene sentido realizar este cambio, a pesar de incrementar el trabajo.

4.2.5. Bus interno de señales de control (ICB).

Se han añadido algunas señales a este bus, ya que eran necesarias en varios bloques del dispositivo. También se han eliminado otras que no son necesarias que estén en este bus, ya que se encuentran solamente dentro de un bloque. Quedando este bus con las señales que se ven en la Tabla 3, donde están todas las señales ordenadas con su número de señal dentro del bus.

Pos.	Señal	Pos.	Señal	Pos.	Señal	Pos.	Señal	Pos.	Señal
0	WR_PC	24	WR_D	48	/OE_AUX6	72	/SET_F2	96	/OE_IVEC
1	/OE_PC	25	/OE_D	49	/INC_AUX6	73	/CLR_F2	97	/OE_INI
2	CK_PC	26	WR_E	50	CK_AUX6	74	F2	98	/OE_IAUT
3	WR_SP	27	/OE_E	51	WR_AUX7	75	/SET_F3	99	ULA3
4	/OE_SP	28	/OE_CR	52	/OE_AUX7	76	/CLR_F3	100	ULA2
5	/INC_SP	29	WR_AUX0	53	/INC_AUX7	77	F3	101	ULA1
6	CK_SP	30	/OE_AUX0	54	CK_AUX7	78	/SET_IM	102	ULA0
7	WR_AR	31	/MSB_AUX0	55	WR_PSW	79	/CLR_IM	103	WR_ULA1
8	/OE_AR	32	WR_AUX1	56	/OE_PSW	80	IM	104	WR_ULA2
9	WR_DR	33	/OE_AUX1	57	CTL_PSW	81	/SET_IF	105	/OE_ULA
10	DIR_DR	34	/MSB_AUX1	58	/SET_CF	82	/CLR_IF	106	/INTSS
11	/OE_DR	35	WR_AUX2	59	/CLR_CF	83	IFlag	107	/OE_BUSQ
12	WR_IR	36	/OE_AUX2	60	CF	84	/R_MEM	108	/OE_VECT
13	/OE_RI	37	/MSB_AUX2	61	WR_CF	85	W_MEM	109	/OE_AUTO
14	WR_IX	38	WR_AUX3	62	/SET_ZF	86	/R_ES	110	/OE_INIC(Ins.)
15	/OE_IX	39	/OE_AUX3	63	/CLR_ZF	87	/W_ES	111	/OE_DataReg
16	/INC_IX	40	/MSB_AUX3	64	ZF	88	/MINT0	112	/OE_INI_INT
17	CK_IX	41	WR_AUX4	65	WR_ZF	89	/MINT1		
18	WR_A	42	/OE_AUX4	66	/SET_F0	90	/EN_RDY		
19	/OE_A	43	/MSB_AUX4	67	/CLR_F0	91	/FIN_CM		
20	WR_B	44	WR_AUX5	68	F0	92	/FIN_CI1		
21	/OE_B	45	/OE_AUX5	69	/SET_F1	93	/FIN_CI2		
22	WR_C	46	/MSB_AUX5	70	/CLR_F1	94	/FIN_CI		
23	/OE_C	47	WR_AUX6	71	F1	95	WR_IVEC		

Tabla 3. Señales del ICB.

4.2.6. Registro de interrupción vectorizada (IVECT).

Como se ha comentado antes, se ha tenido que disminuir las señales de salida de la UCROM, tenido que eliminar algunas de ellas que no eran imprescindibles. En este caso, la señal /INT+1 de este registro se ha eliminado, ya que la función de incrementar el valor de un dato de 16 bits se puede hacer usando los registros auxiliares de 16 bits o el registro IX.

Es por eso por lo que se ha decidido eliminar esta señal, ya que no supondría ningún impacto en la implementación, solamente habría que cambiar la instrucción de las interrupciones vectorizadas, sin necesidad de incrementar el tiempo de ejecución. El registro quedaría como se muestra en la Figura 23.

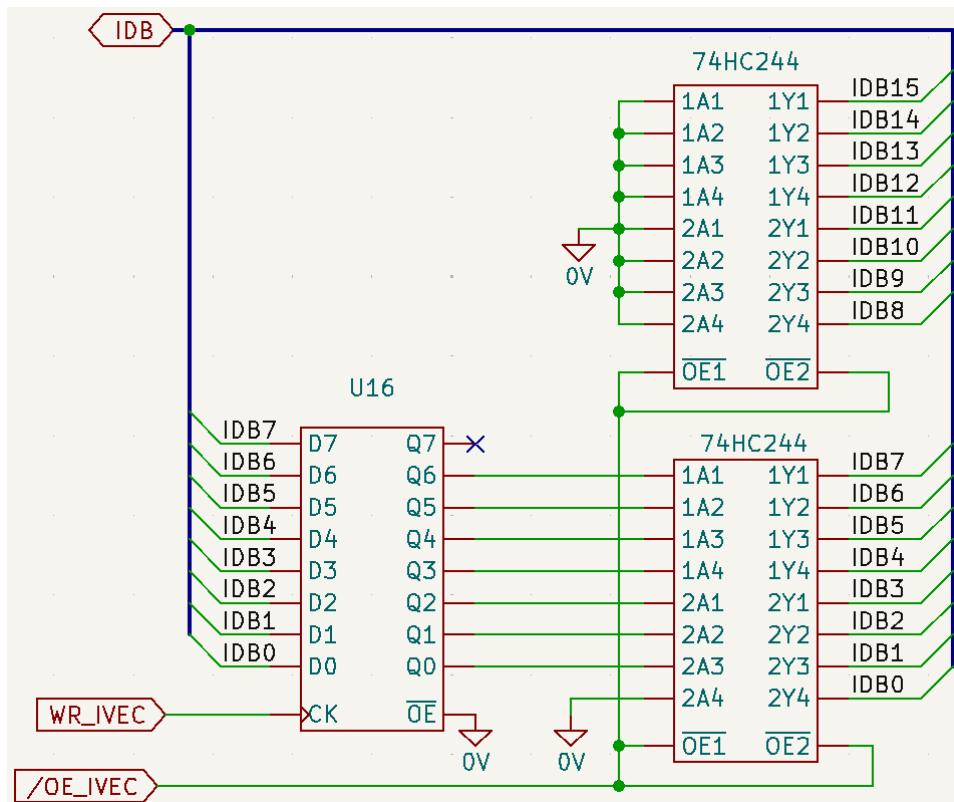


Figura 23. Esquema del registro IVEC.

4.2.7. Registro de interrupción autovectorizada (IAUTO).

En este caso, se ha cambiado este bloque, porque produce un error de funcionamiento. Este registro entrega la dirección de la memoria donde se encuentra la rutina para atender las interrupciones autovectorizadas. Pero como está implementado en el HTDI20 teórico produciría un fallo, ya que la dirección que otorga se encuentra en la memoria RAM. De modo que, al apagar el dispositivo se eliminaría la rutina de las interrupciones autovectorizadas. Entonces, se ha cambiado el dato que contiene de 0x0010 a 0xF010, pasando a verse el esquema como en la Figura 24.

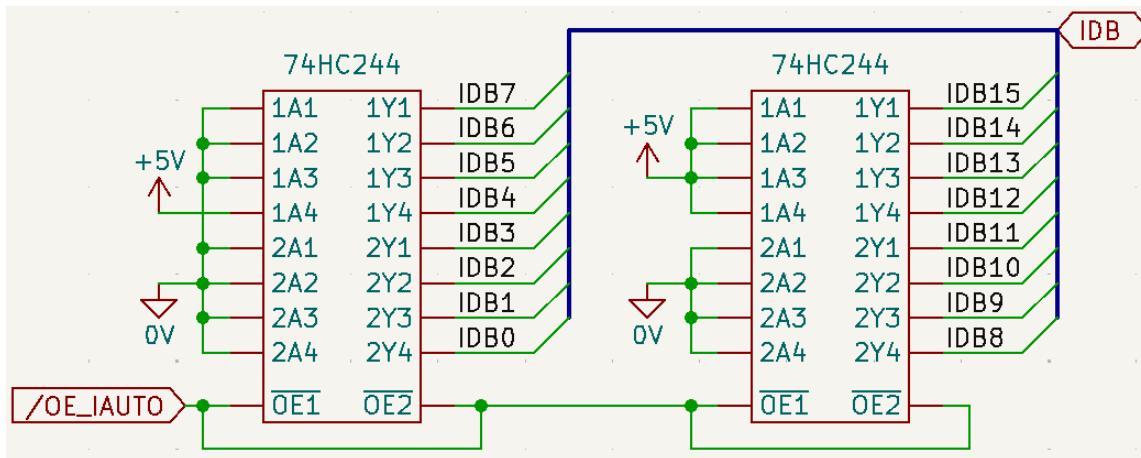


Figura 24. Esquema del registro IAUTO.

4.2.8. Selector de zona.

Como se ha añadido una nueva instrucción básica, se ha tenido que agregar un nuevo bloque para seleccionar la zona de la UCROM donde se encuentra. Esta nueva instrucción comienza en la dirección 0x3EC0, por lo que el selector de zona debe aportar el dato 0xFB. En la Figura 25 se muestra el esquema de este nuevo bloque, que se ha añadido al selector de zona.

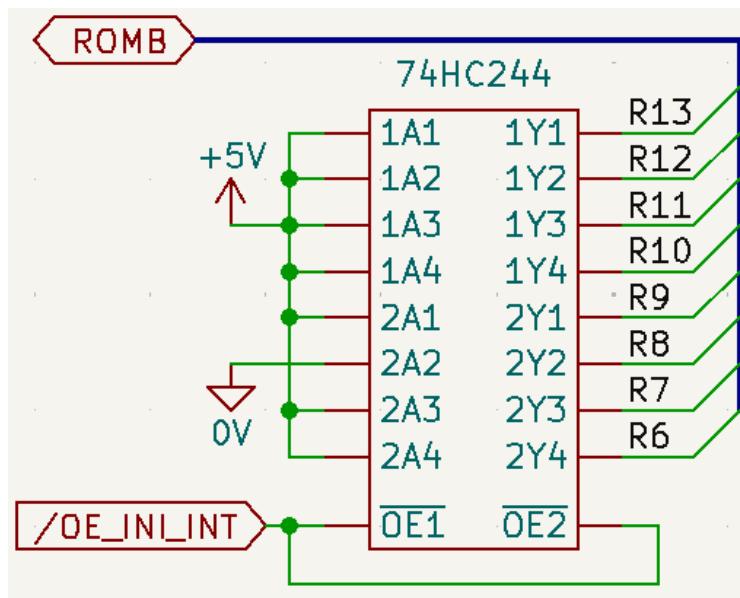


Figura 25. Conexión del bloque de vector inicio de interrupción.

4.2.9. Contador de zona.

Al haber tenido que reducir el tamaño del bus de direcciones de la memoria UCROM, se ha tenido que modificar el contador de zona para que ahora sea de tan solo 6 bits, contando de 0 a 64.

4.2.10. Reinicio del contador de zona.

Se ha añadido este bloque, ya que no está presente en el libro (Sotorriño, 2021). Este bloque es imprescindible, porque se encarga de reiniciar el contador de zona una vez se ha terminado de ejecutar una instrucción. En la Figura 26 se observa cómo se ha implementado este nuevo bloque y en la Figura 27 su cronograma de funcionamiento.

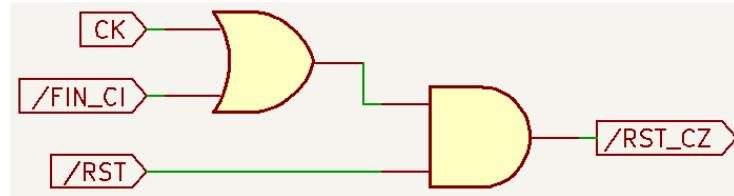


Figura 26. Esquema del reinicio del contador de zona.

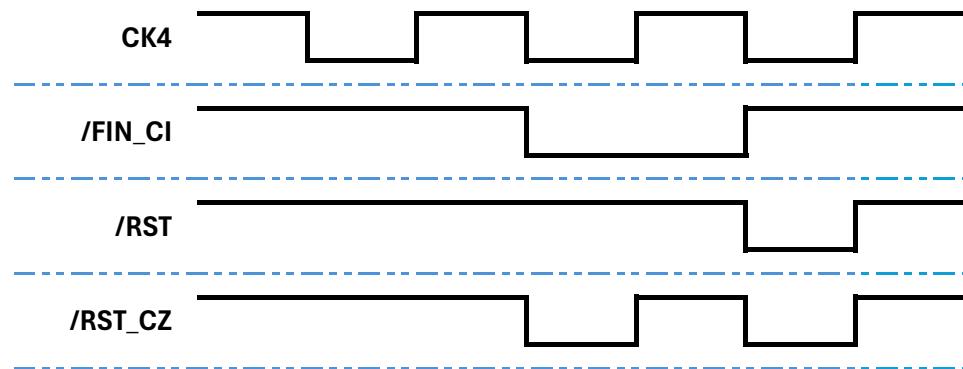


Figura 27. Cronograma de funcionamiento del bloque de reinicio del contador de zona.

4.2.11. Señal de reloj del SUC (CKSUC).

Para conseguir el correcto funcionamiento del microprocesador, este ha sido uno de los esquemas que ha sido necesario modificar. El motivo de esto es que se busca que el flanco ascendente de CKSUC se produzca antes de que termine el periodo de reloj, para que el siguiente estado esté elegido antes de que acabe la ejecución de la instrucción. El nuevo esquema de este bloque es el de la Figura 28.

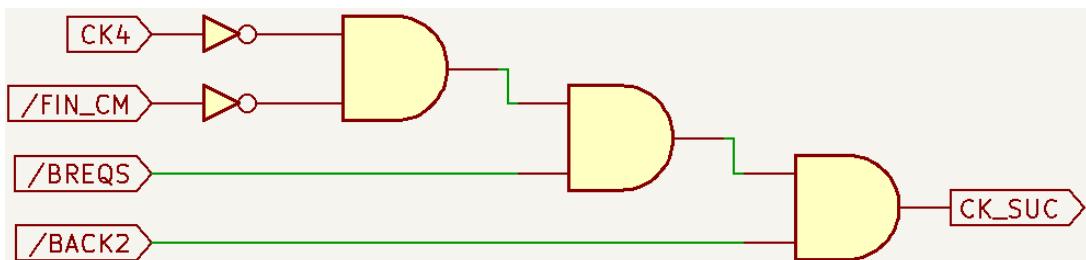


Figura 28. Esquema del generador de CKSUC.

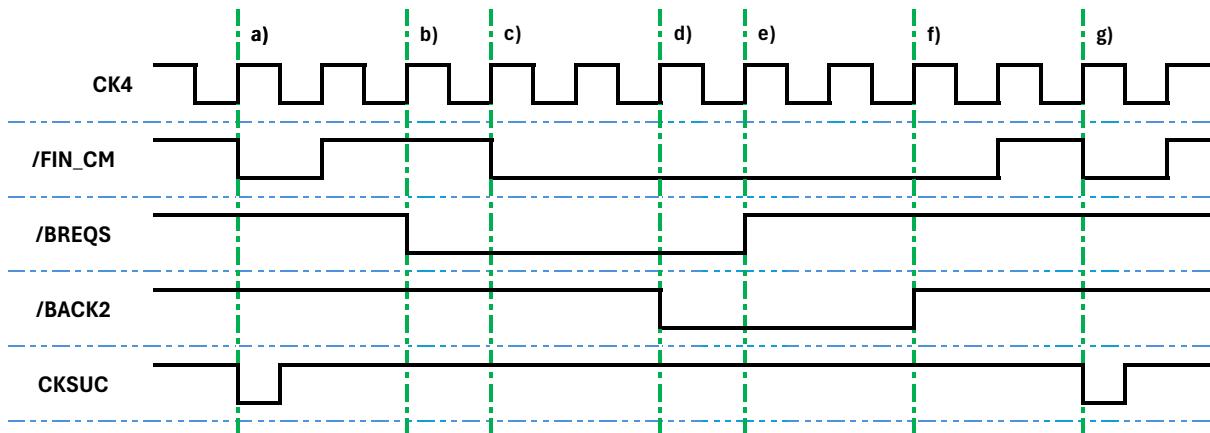


Figura 29. Cronograma de funcionamiento del generador de la señal CKSUC.

En la Figura 29 se puede comprobar el funcionamiento de este bloque, siendo cada uno de los estados los descritos a continuación:

- Funcionamiento normal.** En este punto, se produce un nivel bajo en la señal de CKSUC cuando la señal de reloj está a nivel alto y la que indica el final de un ciclo maquina está a nivel bajo.
- Señal de petición de bus, BREQS, activa. En este caso, en el siguiente nivel bajo de la señal /FIN_CM se entrará en estado de ADM y la señal de reloj de la SUC permanecerá a nivel alto.
- Entrada en ADM.** Tras ponerse a nivel bajo la señal de /FIN_CM, se entra en estado de ADM.
- Dos periodos de reloj después se activa la señal /BACK2.
- El dispositivo que solicito la entrada en ADM desactiva la petición, poniendo a nivel alto la señal /BREQS.
- Salida de ADM.** Tras dos periodos de reloj desde la desactivación de la señal de petición de bus, se desactiva la señal /BACK2.
- Vuelta al funcionamiento normal.** Se continúa generando la señal de CKSUC, tras salir del estado de ADM.

4.2.12. Secuencial de la unidad de control (SUC).

A este bloque, como se ha comentado anteriormente, ha sido necesario incluirle un nuevo estado para el inicio de las interrupciones, que se ejecuta justo antes que el estado de interrupción. Por lo que ha sido necesario añadirle una nueva señal de salida, que activa el registro “INI_INT”, descrito en el apartado 4.2.8., cuando se encuentra en este nuevo estado. De modo que, el nuevo diagrama de estados sería el de la Figura 30 y también se ha creado la Tabla 4 con la evolución de los estados, donde cada una de las transacciones son:

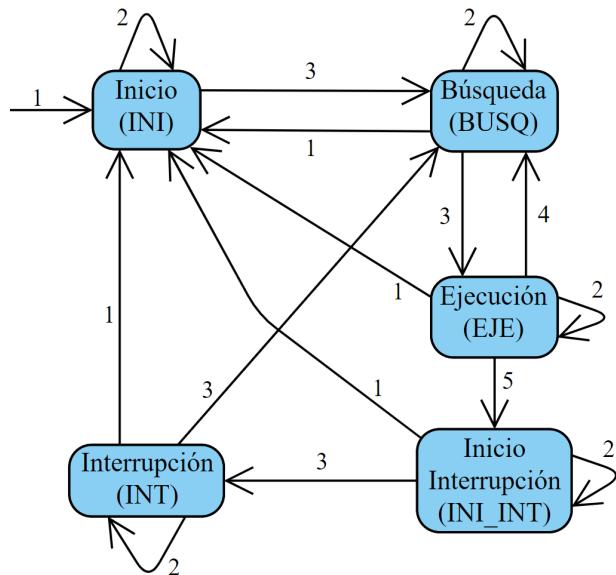


Figura 30. Diagrama de estados del SUC.

1. **Reinicio del dispositivo.** Se pone la señal de reinicio a nivel bajo, /RST = '0'. Siempre que se produzca este caso, se vuelve al estado de INICIO.
2. **No ha terminado la instrucción.** El dispositivo no cambia de estado, ya que esta activada la señal que indica el final de la instrucción, /FIN_CI = '1'.
3. **Ha terminado la instrucción.** Cuando finaliza una instrucción, la señal que lo indica se activa poniéndose a nivel bajo , /FIN_CI = '0'. Una vez se produce esto, el dispositivo cambia de estado, a no ser que se encontrara en "Ejecución".
4. **Finaliza la "Ejecución" y se busca el nuevo código de instrucción.** Cuando se termina de ejecutar una instrucción, se comprueba si hay una interrupción pendiente de atender, /INT = '0', y si no están enmascaradas las interrupciones , IF = '0'. En caso de que no haya una solicitud de interrupción o estén enmascaradas, se pasará al estado de "Búsqueda", para comenzar la ejecución de una nueva instrucción.
5. **Entrada en modo interrupción.** Sería el caso contrario al anterior, las interrupciones no están enmascaradas y además hay una petición de interrupción. En este caso, se iniciaría la interrupción con el estado "Inicio Interrupción" y cuando termine esa instrucción, /FIN_CI = '0', se cambia al estado interrupción.

Cada uno de los estados de la SUC realiza las siguientes funciones:

- **Inicio (INI):** Este estado se encarga de poner el microprocesador con todos los valores iniciales, es decir, reiniciarlo. Establece el registro PC en la dirección de inicio (0xFFFF0), se enmascaran las interrupciones y se ponen por flaco descendente.
- **Búsqueda (BUSQ):** Como su nombre bien indica, en este estado el HTDI20 busca el código de la instrucción que tiene que ejecutar y lo almacena en el registro IR.

- **Ejecución (EJE):** Una vez se tiene el código de la instrucción que se va a ejecutar, se pasa a este estado y se comienza con la ejecución de la instrucción.
- **Inicio Interrupción (INI_INT):** Es un nuevo estado y se encarga de almacenar el valor actual de registro PC en la pila del procesador. Esto se debe a que las interrupciones vectorizadas duraban más de 16 períodos de reloj y no cabía en la memoria UCROM. Por lo que se les ha eliminado esta parte a las instrucciones, para dividirlas en dos y que duren menos de 16 períodos de reloj cada una.
- **Interrupción (INT):** Este estado se continuaría ejecutando la parte de las interrupciones restantes, que no se han ejecutado en el estado INI_INT.

Nº	Estado Inicial	Señales	Estado final
1	---		
1	INICIO		
1	BUSQUEDA		
1	EJECUCIÓN	/RST = '0'	INICIO
1	INICIO INTERRUPCIÓN		
1	INTERRUPCIÓN		
2	INICIO	/FIN_CI = '1'	INICIO
3		/FIN_CI = '0'	BUSQUEDA
2	BUSQUEDA	/FIN_CI = '1'	BUSQUEDA
3		/FIN_CI = '0'	EJECUCIÓN
2	EJECUCIÓN	/FIN_CI = '1'	EJECUCIÓN
4		$[(/FIN_CI = '0') \& (IF = '1')] \parallel$ $[(/FIN_CI = '0') \& (IF = '0') \& (/INT = '1')]$	BÚSQUEDA
5		$(/FIN_CI = '0') \& (IF = '0') \& (/INT = '0')$	INICIO INTERRUPCIÓN
2	INICIO INTERRUPCIÓN	/FIN_CI = '1'	INICIO INTERRUPCIÓN
3		/FIN_CI = '0'	INTERRUPCIÓN
2	INTERRUPCIÓN	/FIN_CI = '1'	INTERRUPCIÓN
3		/FIN_CI = '0'	BUSQUEDA

Tabla 4. Evolución de los estados de la SUC.

4.2.13. Controlador UART.

La última novedad que se le ha incorporado al HTDI20 ha sido un controlador de puerto serie tipo UART, que recibe datos enviados por el compilador y los almacena en la memoria de programa. Los pasos que sigue este bloque son los siguientes:

- 1) Cuando detecta un flaco de bajada en la señal de recepción, RX, comienza la recepción de los datos y solicita la entrada en ADM al dispositivo, para poder hacer uso de los buses locales y escribir en la memoria del programa. Cuando recibe la respuesta del microprocesador, activando la señal /BACK, comienza a usar los buses locales. Permaneciendo en estado de ADM hasta que reciba el ultimo byte del compilador y los escriba en la memoria.
- 2) Tras el flanco descendente, comienza una señal de frecuencia 1 MHz con ciclo de trabajo del 50%, que comienza a nivel alto. Esta señal indicaría cuando se debe leer el bit, que sería siempre en el centro del bit, como se ha comentado en el apartado 3.4.1. Es decir, hay que leer los bits en el flaco descendente de esta señal, desechando el primero que es el bit de inicio y leyendo los 8 siguientes que son el dato, comenzando con el bit menos significativo.
- 3) Una vez se ha recibido un byte, termina la recepción de datos y se espera al siguiente flanco descendente de la señal RX, que indica que se está enviando otro dato.
- 4) El primer byte que se envía es el MSB de la dirección de memoria donde va a guardar el dato, el segundo byte es el LSB de la dirección y el tercer byte es el dato que se va a guardar. Por lo tanto, por cada byte que se quiera guardar en la memoria se deben enviar 3 bytes.
- 5) El controlador espera el tiempo que se tarda en enviar 4 datos, es decir 12 bytes, para empezar a guardar los datos que se han recibido en la memoria, no se espera hasta terminar toda la recepción de datos, porque si no se necesitaría un buffer muy grande. En caso de que uno de esos bytes que se vaya a guardar, se ubique en una dirección con los 10 bits más significativos (A15-A6) distintos a los demás que se están guardando en este tiempo de guardado, se detendrá el guardado y se esperará el tiempo necesario para que la memoria guarde los anteriores bytes, que será el tiempo que se tarda en recibir 4 datos.
- 6) Una vez se han guardado todos los datos y no se están recibiendo más datos, se pone la señal de reinicio del microprocesador a nivel bajo y se desactiva la señal de petición de buses, /BREQ.

En el cronograma de la Figura 31 se puede comprobar el funcionamiento de este bloque.

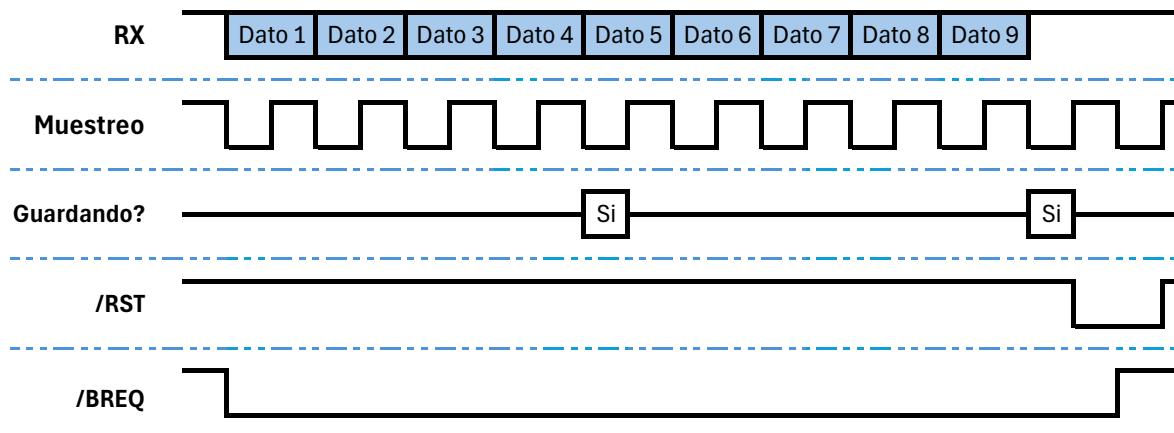


Figura 31. Cronograma funcionamiento del controlador UART.

Como este bloque hace uso de los buses externos del microprocesador, se debe conectar en el exterior del dispositivo, quedando el esquema de conexión como se muestra en la Figura 32. Como se puede apreciar se han añadido dos puertas AND, para las señales /BREQS y /RST. La finalidad de esto es que el controlador UART pueda hacer uso de esas señales, sin quitar la posibilidad de que el usuario también pueda hacerlo de ellas externamente. De modo que, si la UART o el usuario activan las señales, las ponen a nivel bajo, la puerta AND produzca un nivel bajo en la salida que indique un reinicio o una petición de bus, dependiendo de cuál de las señales sea.

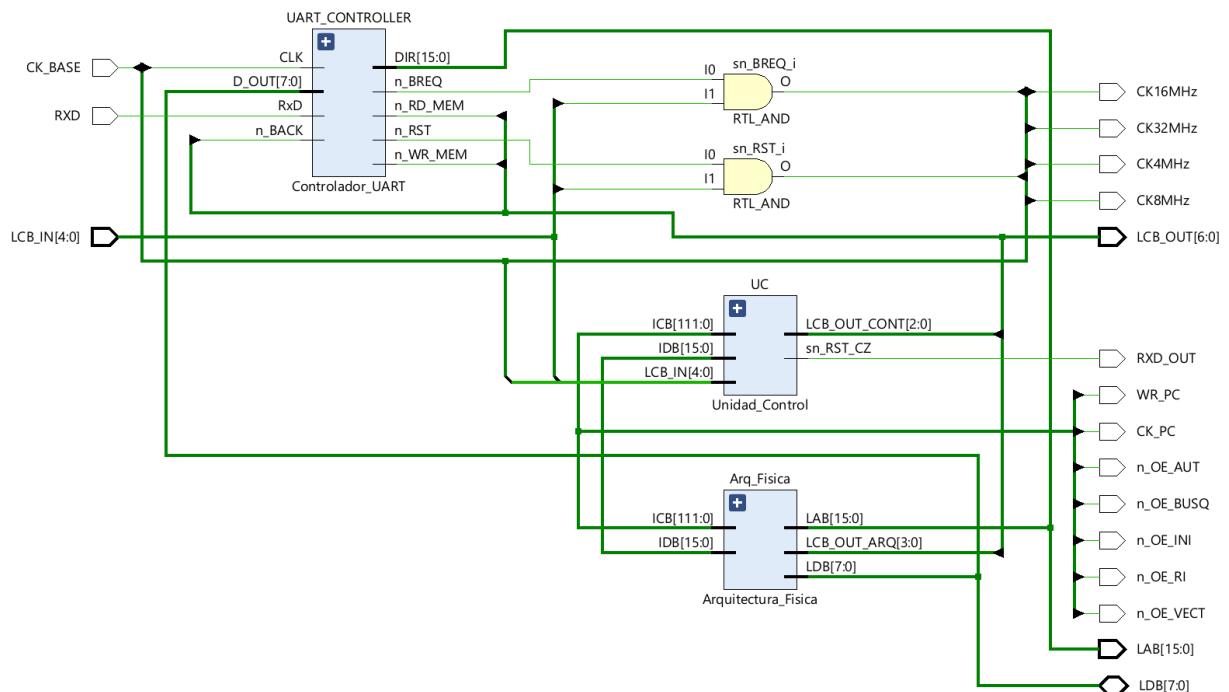


Figura 32. Esquema de conexión del controlador UART.

4.3. Códigos de instrucción.

Para hacer funcionar el HTDI20, una de las partes más importantes y tediosas a la vez, ha sido la de diseñar los cronogramas de funcionamiento de cada uno de los códigos de instrucción, que se muestran en la Tabla 9. Como se puede ver, son un total de 197 instrucciones, que se encarga de manejar cada una de ellas 96 señales. Con la finalidad de hacer el trabajo más fácil, se parte de un cronograma con las 96 señales en su estado de reposo, donde solo habría que modificar las señales que usa esa instrucción en concreto.

Como estos datos se guardan en la memoria UCROM de la unidad de microprograma, es necesario crear un archivo COE para declarar los datos que almacenan cada una de las direcciones. Para hacer este archivo del modo más sencillo, se ha diseñado un archivo Excel donde cada una de las filas es una señal de la memoria, cada columna es una dirección de la UCROM y cada celda es un bit que almacena la memoria. Para simplificar el proceso, en función del valor binario que contenga una celda, se dibuja el borde superior o inferior de esta, para que toda la tabla tenga aspecto de cronograma. Un ejemplo de esto es el que aparece en la Figura 33.

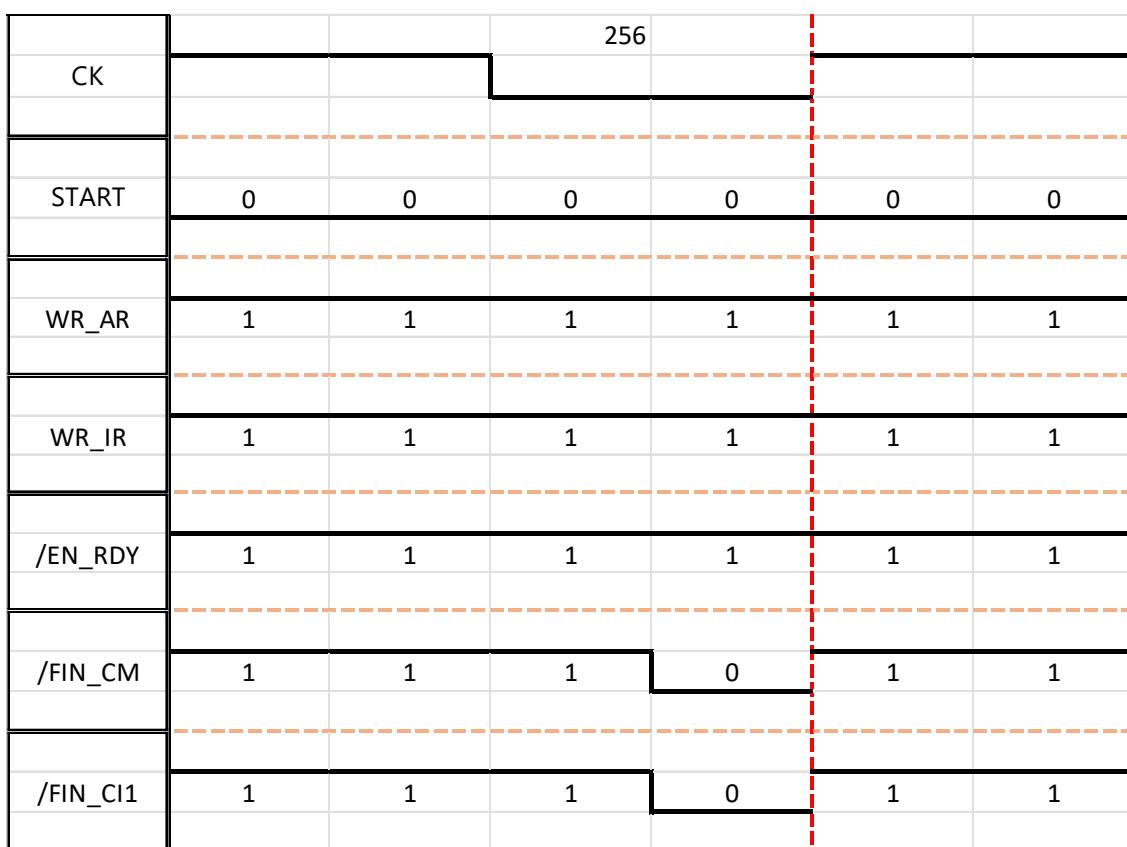


Figura 33. Ejemplo del archivo Excel para diseñar los códigos de instrucción.

Además de esto, al final de cada una de las columnas se genera el valor hexadecimal que tiene esa dirección y se agrupan todos en orden en una celda. Para de este modo poder copiarlo y pegarlo en el archivo COE. Todo esto se puede comprobar en la Figura 34. Este archivo se puede encontrar en el repositorio del trabajo (Escobar, 2024), en un archivo llamado “Blanco.xlsx”, que tiene todas las señales de reposo y del que se ha partido para crear cada una de las instrucciones.

Codificación hexadecimal columnas:	DFF93FFFFFFF F77FFFF77BF FE	FFF93FFFFFFF F77FFFF77FF FE	FFFDFFFFFFFF F77FFFF76FF FE	FFFDFFFFFFFF F77FFFF76FF CE	FFFDFFFFFFFF F77FFFF76FF FE	FFFDFFFFFFFF F77FFFF76FF FE	FFFDFFFFFFFF F77FFFF76FF FE	FFFDFFFFFFFF F77FFFF76FF FE
Codificación cada T:	F77FFFF77BF FE, DFF93FFFFFF				F77FFFF76FF FE, FFFDFFFFFFFF			
Codificación Total:	DFF93FFFFFFF77FFFF77BFFE, DFF93FFFFFFF77FFFF77FFFE, FFFDFFFFFFFF77FFFF76FFFE, FFFDFFFFFFFF77FFFF76FFCE							

Figura 34. Codificación de las instrucciones en el archivo Excel.

Para simplificar la tarea de copiado, se han agrupado las instrucciones separadas por grupos, donde cada una de las instrucciones es una hoja del archivo Excel. Y después se añade una hoja al principio, donde se ha insertado una tabla que agrupa los códigos de instrucción. Un ejemplo de esto se ve en la Tabla 5 con las instrucciones básicas.

Numero Instrucción	Codificación	Codificación total	Número de datos
252	FFF9FFFFFFF77FFAD767FFE, FF	FFF9FFFFFFF77FFAD767FFE, FFF9FFFFFFF	64
253	7FFDFFFFFFF77FFFF77BFFE, 7F		
254	BFFDFFFFFFF77FFFF76FFFF, BF		
255	FFFDFFFFFFFF77FFFF767BFD, FF	FFFDFFFFFFFF77FFFF76FFFE, FFFDFFFFFFFF	256
256	DFF93FFFFFFF77FFFF77BFFE, DF		
NUMERO DE CARACTERES			320

Tabla 5. Codificación de las instrucciones básicas.

Una vez se tienen todas las tablas, se crea un archivo de texto para cada grupo de instrucciones y se copia y pega en orden, en los archivos de texto, las celdas de codificación total de ese grupo. Para verificar que se han copiado todas las celdas se comprueba que el número de caracteres del archivo de texto y el del calculado en Excel coincidan.

Una vez se tienen todos los archivos de texto, se copia su contenido por orden de grupo y se pega todo en un archivo de texto que contiene todas las instrucciones. También se ha creado un archivo Excel, que agrupa la cantidad de caracteres que debe tener el archivo de texto tras pegar cada uno de los grupos, que es la Tabla 6.

Con este archivo de texto ya se puede crear el COE file y de este modo general la memoria de la unidad de control, UCROM.

Grupo Instrucciones	Numero Comas	Nº Caracteres	Suma Caracteres
1_Saltos_Condicionales	768	19968	
2_LLamadas_Condicionales	256	6656	26624
3_Copia_De_Datos	2816	73216	99840
4_Manejo_De_ES	832	21632	121472
5_Copia_De_Datos	1088	28288	149760
6_Manejo_De_Pila	1024	26624	176384
7_Logicas	1344	34944	211328
8_Aritmeticas	2816	73216	284544
9_Salto	64	1664	286208
10_Llamada	64	1664	287872
11_Retornos	128	3328	291200
12_Control	1088	28288	319488
13_Vacio	3840	99840	419328
14_Instrucciones_basicas	256	6656	425984
	16384	425984	

NUMERO DE CARACTERES	425984
----------------------	--------

Tabla 6. Suma de caracteres de los distintos grupos de instrucciones.

4.4. Cronogramas de funcionamiento.

Los cronogramas de funcionamiento de cada uno de los bloques se pueden encontrar en el libro (Sotorriño, 2021), excepto los de los bloques que se han modificado, que se muestran en el apartado 4.2. Además, los cronogramas de cada una de las instrucciones se encuentran en la carpeta “Códigos de Instrucción” del repositorio del GitHub del proyecto (Escobar, 2024).

Aun así, en este documento se va a exponer un cronograma de funcionamiento de la ejecución del código mostrado en la Figura 35. En el que se puede observar que es un simple comando de salto.

```

;-----;
;Se declaran las etiquetas constantes.
;-----;
DIRBASE EQU 0xFFFF0

;-----;
; Se comienza el programa
;-----;

MAIN: ORG DIRBASE
      JMP MAIN

```

Figura 35. Programa para el cronograma.

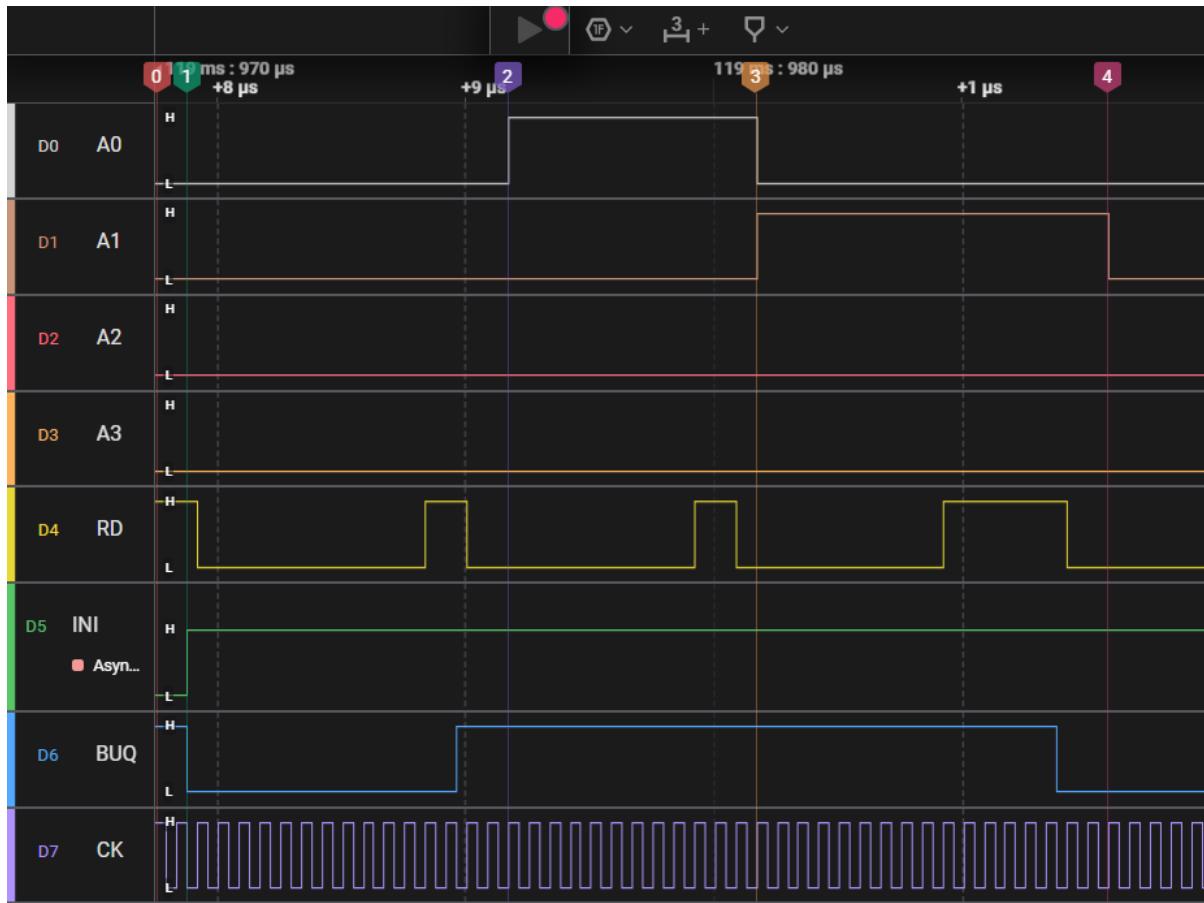


Figura 36. Cronograma de funcionamiento.

En la Figura 36 se puede ver el cronograma de funcionamiento de esta instrucción, donde en cada punto sucede lo siguiente:

0. Aquí se está reiniciando el dispositivo, ya que esta activa la señal que indica la ejecución de la instrucción de inicio.
1. Una vez termina de reiniciarse, se puede apreciar que las direcciones están en 0x0, esto se debe a que el registro PC está en la dirección de inicio, 0xFFFF. Y, además, comenzaría el ciclo de búsqueda para obtener el código de instrucción de esta dirección.
2. Finaliza el ciclo de búsqueda e incrementa la dirección. Como esta instrucción es de 3 bytes, es necesario leer los 2 bytes de datos que se encuentran en las siguientes direcciones.
3. Se lee el ultimo byte guardado, que indica la dirección a la que se quiere saltar.
4. Finaliza la instrucción y como se puede ver ya se ha saltado de nuevo a la dirección de inicio.

Capítulo 5. Compilador.

5.1. Introducción.

Para facilitar la programación de del dispositivo, se ha diseñado un compilador de lenguaje ensamblador que usa el juego de instrucciones de la Tabla 9. Este programa genera dos archivos de salida, uno de ellos es la codificación del programa con la dirección de la memoria donde se almacena cada dato, y en el otro se muestran los errores que contiene el código. Una vez se han generados estos archivos, el usuario puede elegir si desea enviar el programa al dispositivo y cargarlo en la memoria.

5.2. Implementación del compilador.

Para implementar el compilador se utiliza una herramienta llamada Flex, que es una variante open-source del analizar léxico llamado Lex. Estos programas generan analizadores léxicos(“Lexers” o “Scanner”), que son capaces de interpretar léxicamente textos y en función del contenido realiza una acción u otra. Para redactar este código se usa el programa Visual Studio, con una extensión para Flex. Es importante guardar el archivo con la extensión “.l”.

```
%{
    /*Zona de definiciones*/
}

/*
/*Zona de Patrones*/

%%
    /*Zona de reglas*/
%%
    /*Zona de código de usuario(MAIN)*/
}
```

Figura 37. Estructura de archivo Flex.

La estructura de los archivos Flex deben ser la de la Figura 37, donde la función de cada apartado es:

- **Definiciones.** Aquí se declaran variables globales, funciones que se vayan a utilizar en el programa y se incluyen las librerías. De igual modo que antes del “Main” en un código C.

- **Patrones.** En este apartado se establecen los patrones de caracteres que se deseen. En la Tabla 7 se muestran algunos ejemplos de patrones. A estos patrones se le puede asignar un nombre para llamarlos más adelante, como en el siguiente caso:

```
ALFANUMERICO [A-Z0-9a-z]
DATOS_1BYTE (0x([A-F0-9]{2}))
```

En estas dos líneas de código se declara el patrón “ALFANUMERICO”, que incluye a todos los caracteres alfanuméricos en mayúscula y minúscula. Y en la otra línea se declara un patrón que es “DATOS_1BYTE”, que sería una cadena de caracteres que tenga el patrón de un dato hexadecimal. Es decir, que comience por “0x” y los sigan dos caracteres que sean del ‘0’ al ‘9’ o de ‘A’ a ‘F’ en mayúscula.

Expresión	Descripción
.	Cualquier carácter excepto un salto de línea.
[0-9]	Un número del 0 al 9.
\n	Un salto de línea.
\t	Una tabulación.
[0+9]	Los caracteres ‘0’, ‘+’ o ‘9’.
[-0-9]	Un número del 0 al 9 o ‘-’.
[0-9]+	Uno o más dígitos del 0 al 9.
[0-9]*	Cero o más dígitos del 0 al 9.
[^a]	Cualquier carácter excepto la ‘a’.
[A-Z]{2,4}	De dos a cuatro caracteres de la ‘A’ a la ‘Z’.
w(x y)z	Una cadena que sea “wxz” o “wyz”.

Tabla 7. Patrones básicos en Lex.

- **Reglas.** En esta parte del documento se declaran las acciones que se van a realizar cuando se detecte un patrón determinado. Por lo que, se definiría el patrón que se está buscando y tras una tabulación se establecería la acción que se va a llevar a cabo. En las siguientes líneas se muestra un ejemplo de esto:

```
%{
#include <stdio.h>

Imprimir(){
    printf("La línea es: %s", yytext)
}
```

```
%}

ALFANUMERICO      [A-Z0-9a-z]{1,20}

%%

{ALFANUMERICO}:""  Imprimir();
```

En este pequeño código, si una cadena de caracteres está compuesta por el patrón “ALFANUMERICO” seguido de dos puntos, se ejecutaría la función llamada “Imprimir()”. Que imprimiría en el terminal la cadena de caracteres que coincide con el patrón, ya que en lenguaje Flex la cadena que se está interpretando en cada instante se almacenada en la variable “yytext”.

- **Código de usuario.** Aquí es donde se declara el “main” del programa, es decir donde se coloca el programa que se va a ejecutar, y donde debe ejecutarse el interpretador léxico para que procese todo el texto. Un pequeño ejemplo de esta parte del programa sería el siguiente:

```
int main(){
    /*Se declaran las variables necesarias*/
    char filename_in[255] = "C:/TFG/Prueba2/Prueba2.txt";

    /*Se abre el archivo dado por el directorio*/
    in = fopen (filename_in, "r");

    /*Se pone el contenido del archivo de entrada como entrada de texto
    para flex*/
    yyin = in;

    /*Se ejecuta Flex*/
    yylex();

    /*Se espera a que se pulse cualquier tecla para cerrar la consola*/
    system("pause");
    return 0;
}
```

Como se aprecia, se abre un archivo de texto que es el que va a analizar Flex, y una vez abierto se establece como fichero de entrada para el analizador léxico. Este se encargará de analizar todo el archivo en busca de los patrones establecido, y en caso de encontrarlos realizará la acción especificada por el usuario.

5.2.1. Compilación de un archivo Flex.

Como se ha comentado antes, para obtener el ejecutable del compilador, es necesario compilar el archivo Flex para obtener un fichero C de salida y este procesarlo con un compilador C para obtener el ejecutable, siendo los pasos mostrados en la Figura 38.

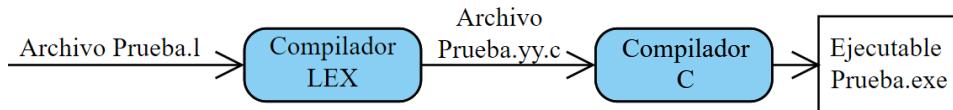


Figura 38. Esquema de compilación de un archivo Flex.

El software Flex fue diseñado para correr en el sistema operativo Unix. Por lo tanto, para llevar a cabo esta tarea se ha instalado el programa MSYS2, que es una plataforma de desarrollo de software que permite ejecutar código Unix en Windows. Esta herramienta permite ejecutar Flex en el sistema operativo para el que fue diseñado, Unix.

Para trabajar con Flex en MSYS2, es necesario instalarlo, además de todos los paquetes que necesita para funcionar. Para instalar algo en Unix, se usa el comando “pacman -S” seguido del paquete que se desea instalar, que son los siguientes:

```

pacman -S clang
pacman -S gcc
pacman -S Mingw
pacman -S Flex
  
```

En caso de que se hayan instalado estos paquetes antes, se pueden actualizar con el comando:

```
pacman -Syu
```

Una vez ejecutados estos comandos uno a uno, se tendrían instaladas todas las herramientas necesarias para empezar a usar Flex. Cuando se desee ejecutar Flex para compilar un archivo, se deben seguir los siguientes pasos:

1. Lo primero de todo es abrir el directorio donde se encuentra el archivo “.l” que se desea compilar. Esto se hace ejecutando el siguiente comando, donde se usa un directorio ficticio de ejemplo:

```
cd C:/Ejemplo/Prueba
```

2. Una vez que se está en el directorio donde se encuentra el archivo que se desea compilar, se puede ejecutar Flex para generar el archivo C de salida, para lo que se debe ejecutar el siguiente comando:

```
flex -o Compilador.yy.c Ejemplo.l
```

Aquí se compilaría el archivo Flex llamado “Ejemplo.l” y generaría un archivo de salida llamado “Compilador.yy.c”.

3. Ya que se ha generado el fichero C, ahora se debe compilar con un compilador de lenguaje C para obtener el ejecutable. Esto se hace ejecutando el siguiente comando, que usa el compilador gcc para procesar el fichero C:

```
gcc Compilador.yy.c -o Compilador_HTDI20
```

Una vez hecho esto se obtiene el ejecutable del analizar léxico que en este ejemplo se llamaría “Compilador_HTDI20.exe”.

5.3. Manual del compilador del HTDI20.

Al usar el compilador debe tenerse en cuenta algunos aspectos para que funcione de manera correcta y no se produzcan errores. Además, se debería conocer los posibles errores que se pueden producir al usarse y su causa, para de este modo poder subsanarlos de una manera rápida y sencilla.

Por lo tanto, en este manual de uso se va a explicar cómo se deben redactar los archivos de programación, los posibles fallos que se pueden producir al compilar un programa y los pasos a seguir para ejecutar el compilador.

5.3.1. Estructura de los archivos de programación.

Los programas que se desean compilar con este software se escriben en un archivo de texto, en el bloc de notas. Las reglas que se deben seguir para la sintaxis de uno de estos programas son:

1. El programa solo puede contener caracteres alfanuméricos, en mayúscula o minúscula.
2. El formato numérico admitido es el hexadecimal. No se admite el binario, decimal, etc.
3. Los operandos de las instrucciones se separan con una coma sin espacios.
4. Se han agregado varias funciones específicas para el compilador, que no ejecuta el microprocesador, estas son:
 - a. **EQU.** Con esta se asigna un valor numérico a una etiqueta, que no puede exceder los 20 caracteres. Por ejemplo:

```
INI      EQU  0xFFFF
```

En esta línea de código se crea una etiqueta llamada “INI” que almacena el valor numérico 0xFFFF.

- b. **ORG.** Indica la dirección de la memoria donde se comenzará a guardar las siguientes líneas de código. Por ejemplo:

```
INI      EQU 0xFFFF0
        ORG  INI
```

En estas se establece que la dirección de comienzo para las siguientes instrucciones es la 0xFFFF0.

- c. **DB.** Almacena valores numéricos, especificados por el usuario, en la memoria. Estos tienen que ser de como máximo un byte y estar separados por una coma. Por ejemplo:

```
TABLA      EQU 0x00F0
        ORG  TABLA
        DB   0x00,0xFF,0xAA,0xAF,0x12,0x23,0xBC
```

En este extracto de código, se almacenan los datos que están a continuación de “DB” en direcciones sucesivas de la memoria, partiendo de la dirección 0x00F0 establecida por la instrucción “ORG”.

5. Las instrucciones admitidas aparte de las específicas del compilador son las que contiene la Tabla 5.
6. Los operadores de las instrucciones deben ser etiquetas declaradas previamente, no se pueden usar valores numéricos, solamente etiquetas.
7. El contenido de una etiqueta es constante y no se puede variar en el programa. Dicho de otro modo, las etiquetas solo se pueden declarar una vez.
8. Las etiquetas siempre deben comenzar por una letra, ya sea mayúscula o minúscula, nunca pueden comenzar por un número.
9. Hay dos tipos distintos de etiquetas:
 - a. **Etiquetas estáticas.** Se declaran por medio de EQU y son de 1 o 2 bytes.
 - b. **Etiquetas dinámicas.** Se coloca en una línea del programa antes de la instrucción, y en la etiqueta se almacena el valor de la dirección de memoria donde se encuentra esa línea de código. Son siempre etiquetas de dos bytes, porque guardan direcciones de memoria. Por ejemplo:

```
INI      EQU 0xFFFF0
        ORG  INI
MAIN:    COP  A,B
        AND  A, C
```

En este caso, la etiqueta "MAIN" contendría la dirección de memoria donde se almacena la instrucción "COP A,B", que es la dirección 0xFFFF0.

10. El compilador distingue entre mayúsculas y minúsculas, es decir, no es lo mismo "INI" que "Ini".

11. Las etiquetas estáticas se deben declarar siempre al principio del programa, en las primeras líneas.

12. El valor que tenga una etiqueta puede modificarse en un rango de 0 a 9, tanto sumando como restando, pero solo en los operandos de las instrucciones. Por ejemplo:

```
VAR1 EQU 0x04  
COP A,VAR1+1  
COP B,VAR-2
```

En estas líneas de código se almacena en el registro A la variable "VAR1" más uno, es decir 0x05. Y en el registro B sería lo mismo, pero restándole 2, por lo que se guardaría el dato 0x02.

13. Para realizar comentarios se debe colocar un punto y coma antes de estos.

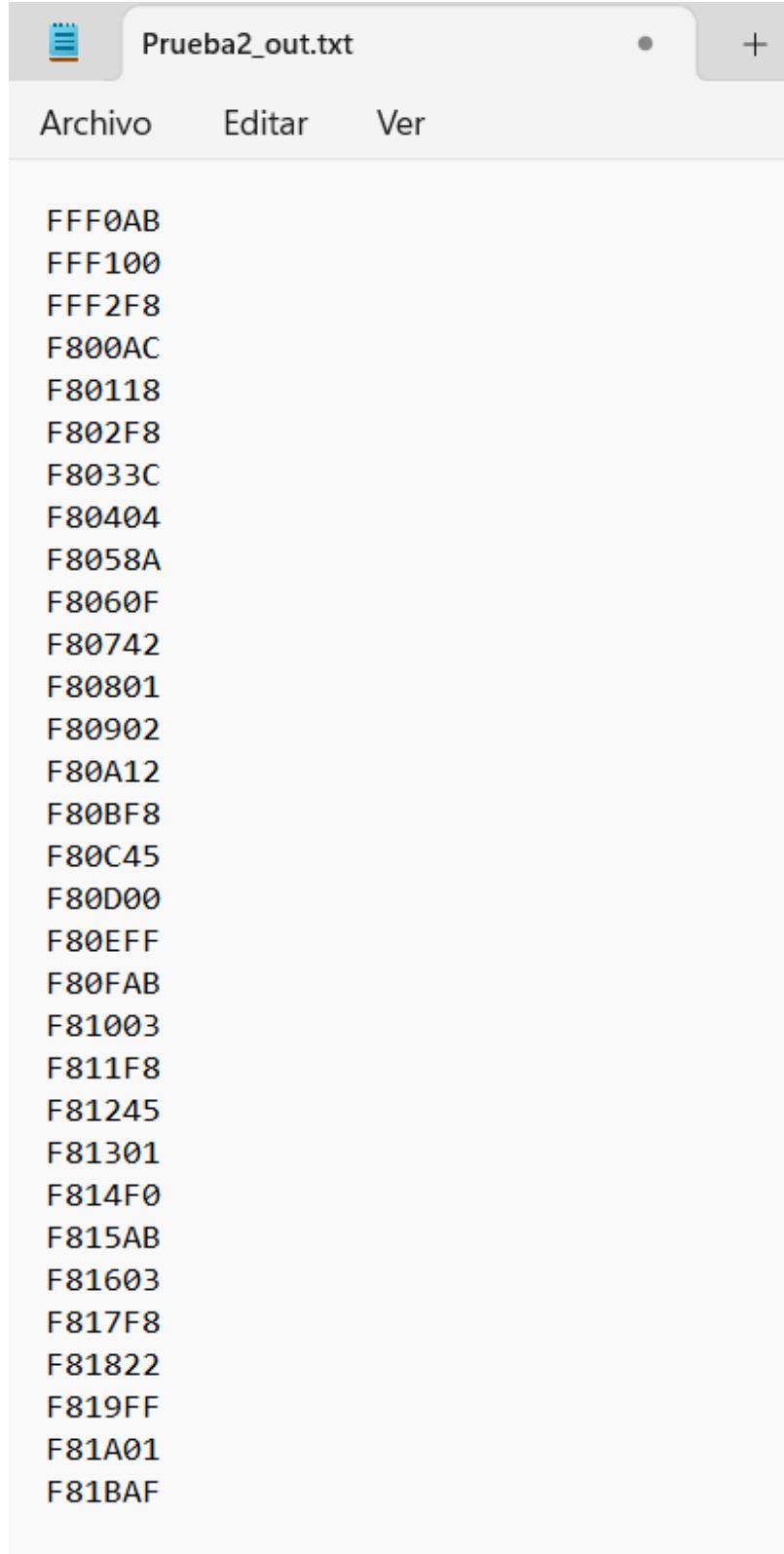
14. No se pueden usar espacios para separar los operandos de las instrucciones, ni las instrucciones de las etiquetas, solo pueden usarse tabulaciones. En otras partes del código si se pueden usar.

15. El código se divide en columnas, donde cada una de las líneas debe seguir el siguiente orden:

- a. La primera columna de la izquierda debe ser mayor de 2 tabulaciones. Esta se usa para declarar las etiquetas.
- b. La segunda columna se usa para el código de instrucción. Ocupa el espacio que sea necesario y termina siempre en una tabulación.
- c. En la tercera columna se colocan los operandos.
- d. En la cuarta columna se escriben los comentarios, empezando siempre con un ";". Aunque también pueden ocupar una línea entera.

16. El compilador genera un fichero("Nombre de fichero de origen"_errores.txt) de errores en el directorio del archivo de programa.

El compilador genera un fichero("Nombre de fichero de origen"_out.txt) con los datos que va a guardar en la memoria, que tiene el aspecto que se muestra en la Figura 39. Donde los dos primeros bytes de cada línea indican la dirección, de la memoria del programa, donde se guarda el tercer byte de la línea.



The screenshot shows a simple text editor window titled "Prueba2_out.txt". The menu bar includes "Archivo", "Editar", and "Ver". The main content area displays a series of memory addresses in hexadecimal format, each followed by a colon and a two-letter suffix. The list starts with FFF0AB and continues through various memory locations up to F81BAF.

Address
FFF0AB
FFF100
FFF2F8
F800AC
F80118
F802F8
F8033C
F80404
F8058A
F8060F
F80742
F80801
F80902
F80A12
F80BF8
F80C45
F80D00
F80EFF
F80FAB
F81003
F811F8
F81245
F81301
F814F0
F815AB
F81603
F817F8
F81822
F819FF
F81A01
F81BAF

Figura 39. Aspecto del archivo de salida de codificación del programa.

Por tanto, un fichero de programación debería verse como en la Figura 40.

```

;-----;
;      Definiciones de constantes
;-----;
INIPILA      EQU      0x01FF ;cte. dir. pila procesador
INIMP        EQU      0xFFFF0 ;cte. dir. inicio HTDI20
INIPRG        EQU      0xF800 ;cte. dir. inicio programa
TBLINT        EQU      0x00F0 ;dir. tabla INTs
INITBLDAC    EQU      0xFF00 ;dir. inicio tabla valores de DAC

DIR0          EQU      0x00    ;dir. espacio E/S 0
DIR1          EQU      0x01    ;dir. espacio E/S 1
DIR4          EQU      0x04    ;dir. espacio E/S 4

DATA1         EQU      0xFF    ;Dato 1
DATA2         EQU      0xF0    ;Dato 2
DATA3         EQU      0XF     ;Dato 3
;-----;

;-----;
;      Definiciones de variables
;-----;
CONT_TX1      EQU      0x0001 ;1B cont. pila Tx1
;-----;

;-----;
;      Iniciacion del sistema
;-----;
INICIO:       ORG      INIMP   ;situá primera instr. del programa
              JMP      INIPRG  ;salta al principio del programa real
              ORG      INIPRG  ;situá el comienzo del programa
              CALL   INI_HTDI ;inicia el HTDI20

MAIN:         IN       A,(DIR4)      ;Se guarda el puerto 4 en A
              SUB      A,DATA3      ;Se resta el dato 3 a A
              OUT      (DIR1),A      ;Se envia A al puerto 1
              JPZ      PPAL      ;¿A = DATA3? Si, salta a PPAL
              OUT      (DIR0),DATA1  ;(DIR0) puerto = DATA1
              JMP      MAIN      ;Se salta a MAIN
PPAL:         OUT      (DIR1),DATA2  ;(DIR1) puerto = DATA2
              JMP      MAIN      ;Se salta a MAIN
;-----;

;-----;
;      Rutina de iniciación de las variables del programa.
;-----;
INI_HTDI:    COP16    SP,INIPILA  ;inicia la pila del procesador
              EI       ;habilita las interrupciones
;-----;

```

Figura 40. Ejemplo de código para el compilador del HTDI20.

5.3.2. Tipos de errores.

Como se ha comentado antes, al ejecutar el compilador se genera un archivo de errores que contiene todos los fallos en la programación que se han cometido, y que hacen que no se pueda compilar el código. Todos estos fallos se indica en la línea en la que se produce y pueden ser de los siguientes tipos:

- **No existe la instrucción.** En caso de que se introduzca una línea de código que no permita el compilador, se imprimirá este error en el archivo.
- **No existe la variable.** Si se usa una variable que no existe como operador, se indicara en el fichero de errores.
- **La variable no tiene el tamaño buscado.** Si una instrucción usa datos de 1 byte y el operando que se utiliza es de 2 bytes o viceversa, se generara este código de error.
- **No se admiten valores numéricos.** Como bien indica, no se pueden usar valores numéricos como operandos, se tienen que declarar antes con una etiqueta.
- **Ya existe la etiqueta.** Como indica la regla 7., no se puede declarar la misma etiqueta más de una vez, en caso de hacerse, se producirá este error.

En la Figura 41 se puede apreciar uno ejemplo de archivo de errores generado por el compilador.

```
Línea 37: No existe la instrucción COPIA      A,B.  
Línea 40: La variable INIPILA es de 2 bytes y el nemonico usa una variable de 1 byte.  
Línea 43: No se admiten los valores numéricos en la dirección de destino.  
Línea 47: No existe la variable DAT09.
```

Figura 41. Ejemplo de archivo de errores.

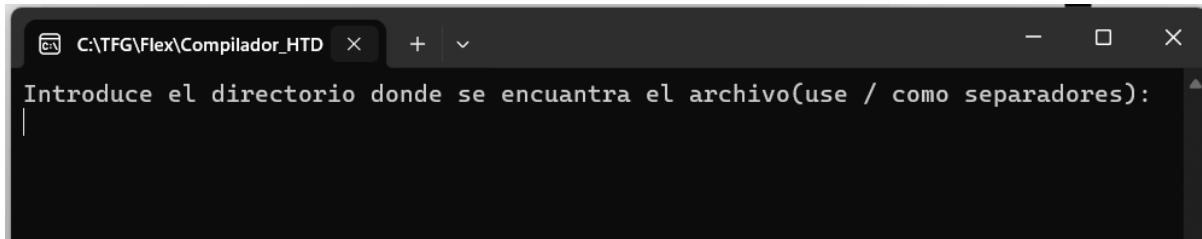
Además de estos errores, también se pueden producir en la ejecución del compilador, pero esos se expondrán más adelante, cuando se explique los pasos a seguir para ejecutar el compilador.

5.3.3. Instrucciones de uso del compilador.

Para procesar un programa en este compilador, se deben seguir los siguientes pasos:

1. Lo primero de todo es ejecutar el ejecutable del compilador, que se llama “Compilador_HTDI20.exe” y se puede encontrar en la carpeta llamada “Compiler” del repositorio del GitHub del proyecto (Escobar, 2024). Además, en esta carpeta se puede obtener el fichero Flex que se ha usado para crear este compilador, el cual está completamente comentado.

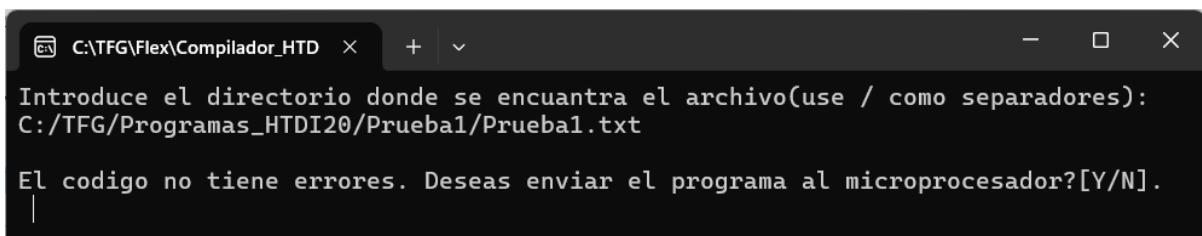
2. Despues de ejecutar el compilador, aparecerá la ventana que se muestra en la Figura 42, donde se debe proporcionar la ubicación del archivo de programa que se desea compilar. Es importante destacar que, se deben separar las distintas carpetas con el símbolo ‘/’, no se puede usar el ‘\’.



```
C:\TFG\Flex\Compilador_HTD + - □ ×  
Introduce el directorio donde se encuantra el archivo(use / como separadores):
```

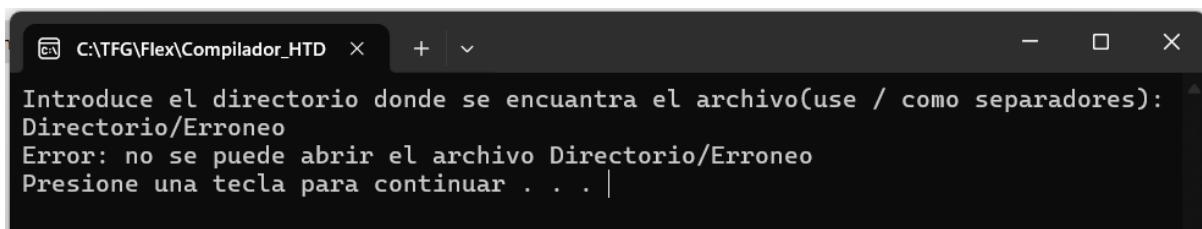
Figura 42. Ejecución del compilador (Solicitud de archivo).

3. Tras proporcionarle el archivo, el programa codifica el archivo y genera los archivos de salida correctamente, quedando la venta como se aprecia en la Figura 43. En caso de que no se pudiera abrir el archivo o sea incorrecto el directorio, se indicaría como en la Figura 44. Y si hay un fallo en la programación se advertirá como en la Figura 45, además de enumerar los errores en el archivo de salida “_errores.txt”.



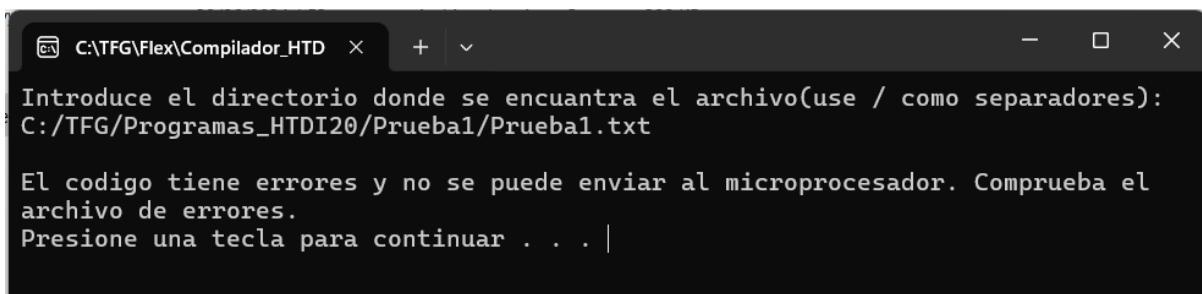
```
C:\TFG\Flex\Compilador_HTD + - □ ×  
Introduce el directorio donde se encuantra el archivo(use / como separadores):  
C:/TFG/Programas_HTDI20/Prueba1/Prueba1.txt  
  
El codigo no tiene errores. Deseas enviar el programa al microprocesador? [Y/N]..
```

Figura 43. Ejecución del compilador (El código no tiene errores).



```
C:\TFG\Flex\Compilador_HTD + - □ ×  
Introduce el directorio donde se encuantra el archivo(use / como separadores):  
Directorio/Erroneo  
Error: no se puede abrir el archivo Directorio/Erroneo  
Presione una tecla para continuar . . .
```

Figura 44. Ejecución del compilador (Error al leer el archivo).



```
C:\TFG\Flex\Compilador_HTD + - □ ×  
Introduce el directorio donde se encuantra el archivo(use / como separadores):  
C:/TFG/Programas_HTDI20/Prueba1/Prueba1.txt  
  
El codigo tiene errores y no se puede enviar al microprocesador. Comprueba el  
archivo de errores.  
Presione una tecla para continuar . . .
```

Figura 45. Ejecución del compilador (El código tienen errores).

4. Una vez corregidos los errores, si los había, se puede elegir si se quiere programar el microprocesador. Si no se quiere hacer, se escribe ‘N’ y se pulsa intro y termina la ejecución. Si se quiere programar el HTDI20, se tiene que responder con ‘Y’.
5. Si se ha decidido enviar el programa, se solicitará el número del puerto serie al que está conectado el HTDI20, como se ve en la Figura 46. Si el puerto serie es correcto, el resultado será el de la Figura 47 y en caso contrario se verá como en la Figura 48.

```
C:\TFG\Flex\Compilador_HTD
Introduce el directorio donde se encuentra el archivo(use / como separadores):
C:/TFG/Programas_HTDI20/Prueba1/Prueba1.txt

El codigo no tiene errores. Deseas enviar el programa al microprocesador?[Y/N].
y

Introduce el numero del puerto serie COM (solo el numero). Es el puerto COM
```

Figura 46. Ejecución del compilador (Solicitud del número del puerto serie).

```
C:\TFG\Flex\Compilador_HTD
Introduce el directorio donde se encuentra el archivo(use / como separadores):
C:/TFG/Programas_HTDI20/Prueba1/Prueba1.txt

El codigo no tiene errores. Deseas enviar el programa al microprocesador?[Y/N].
y

Introduce el numero del puerto serie COM (solo el numero). Es el puerto COM8
Abriendo puerto serie...OK
Enviando el programa...Se ha escrito el programa.. 36 Bytes

Cerrando el puerto serie...OK
Presione una tecla para continuar . . . |
```

Figura 47. Ejecución del compilador (Puerto serie correcto).

```
C:\TFG\Flex\Compilador_HTD
Introduce el directorio donde se encuentra el archivo(use / como separadores):
C:/TFG/Programas_HTDI20/Prueba1/Prueba1.txt

El codigo no tiene errores. Deseas enviar el programa al microprocesador?[Y/N].
y

Introduce el numero del puerto serie COM (solo el numero). Es el puerto COM8
Abriendo puerto serie...Error abriendo
Presione una tecla para continuar . . . |
```

Figura 48. Ejecución del compilador (Puerto serie incorrecto).

Capítulo 6. PCBs del HTDI20.

Para este proyecto se han diseñado dos PCBs, una de ellas es necesaria para que funcione el dispositivo y la otra es para poder interactuar de una manera fácil con el microprocesador.

6.1. Placa de memorias.

Esta placa es imprescindible para que funcione el dispositivo, ya que como su nombre indica, entre otras cosas tiene la memoria EEPROM y RAM de programa. Además, cuentan con otras características que hacen la placa más versátil y segura.

6.1.1. Características.

Todas las utilidades que tiene esta PCB son:

- 32 kB de memoria EEPROM, ubicada en la dirección 0x8000 hasta la 0xFFFF.
- 32 kB de memoria RAM, ubicada desde la dirección 0x0000 hasta la dirección 0x7FFF.
- Alimentación externa con hasta una tensión de 12 V.
- Protección rearmable contra picos de corriente .
- Protección contra picos tensiones inversas.
- Selector de velocidad de reloj. (4MHz, 8 MHz, 16 MHz, 32 MHz o frecuencia proporcionada por el usuario).
- Fuente de tensión de salida de 3,3V y 5V, con hasta 1 A cada una.
- Indicador luminoso de encendido.
- Botón de reinicio.
- Conector DIP para acoplar otras placas que hagan uso del dispositivo.

6.1.2. Selección de las memorias.

Uno de los aspectos fundamentales para tener en cuenta, al elegir las memorias, es que la tensión de alimentación debe ser de 3,3 V, porque el microcontrolador trabaja a esta tensión. Otro aspecto fundamental es que deben ser con salida en paralelo y como mínimo con un bus de datos de 8 bits de tamaño. Una vez establecidos estos parámetros, ya se puede elegir cualquier memoria que cumpla con esto.

Para elegir las memorias, se ha buscado en el distribuidor de componentes electrónico Mouser, memorias con estas características en los apartados de SRAM y EEPROM. Además, se ha añadido un filtro adicional a la búsqueda y es que las memorias sean de 32 kB. Con estos

filtros, en el caso de las EEPROM solo se ha encontrado un modelo, el AT28BV256 del fabricante Microchip. Y en el caso de la memoria SRAM, se han obtenido 15 resultados, donde se ha elegido el modelo con menos tiempo de lectura y escritura, el modelo IS61LV256AL-10JLI-TR del fabricante ISSI.

Una vez se han elegido las memorias, se puede calcular la velocidad de frecuencia máxima a la que puede trabajar el microprocesador, sin que se produzcan fallos de lectura o escritura en las memorias. A la hora de calcular esta frecuencia se necesita conocer los periodos de reloj que tarda el HTDI20 en leer o escribir un dato en la memoria, que en cada caso estos tiempos son:

- **Ciclo de lectura.**

- **Dirección a salida, 7 períodos.** Este tiempo es el que tarda la memoria desde que pone la dirección de memoria deseada, hasta que lee el dato de la memoria.
- **/OE a salida, 8 períodos.** Desde que se activa la señal /OE hasta que se obtiene la salida.
- **/CE a salida, 7 períodos.** Igual que en los casos anteriores, pero con la señal /CE.

- **Ciclo de escritura.**

- **Dirección a flanco descendente de /WR, 1 periodo.** Tiempo que hay que esperar desde que se pone la dirección en el bus de salida, hasta que se pueda producir un flanco descendente en la señal de guardado.
- **Tamaño del pulso de /WR, 8 períodos.**

Esta comprobación solo es necesario hacerlo para la memoria EEPROM, ya que estos tipos de memorias son más lentas que las SRAM, por lo que van a ser el factor limitante. En el caso de este modelo de EEPROM, el tiempo de cada uno de estos parámetros es:

- **Ciclo de lectura.**

- Dirección a salida, 200 ns máximo.
- /OE a salida, 80 ns máximo.
- /CE a salida, 200 ns máximo.

- **Ciclo de escritura.**

- Dirección a flanco descendente de /WR, 0 ns .
- Tamaño del pulso de /WR, 200 ns.

Conociendo todos estos datos ya se puede calcular la frecuencia máxima del procesador, que se hace dividiendo el tiempo por el número de períodos en cada uno de los casos y el

resultado es la inversa de este valor calculado. Haciendo todos los cálculos, se tiene que el factor más limitante es la lectura con el parámetro “Dirección a salida”, que daría una frecuencia de 35 MHz.

6.1.3. Conexión de la etapa de alimentación.

El circuito cuenta con dos reguladores de tensión que son los encargados de alimentar todos los componentes de la placa.

Uno de ellos suministra una tensión de 3,3 V a partir de una de 5 V, siendo este regulador el que alimenta a las memorias. Esta tensión de entrada se puede proporcionar mediante el conector micro USB del CMOD A7 o con el conector de 5 voltios de la PCB.

El otro regulador genera una señal de salida de 5 V, que alimenta la placa CMOD A7 y el regulador de 3,3 V. De este regulador se hace uso cuando la PCB se alimenta por el conector circular o por el pin de 12V, con un rango de tensión de entrada de 6,5 V a 12 V. Esta parte del circuito está protegida contra tensiones negativas, mediante un diodo. Y también está protegido de los picos de corriente con un fusible rearmable.

El esquema de conexión del regulador de 5 voltios se puede ver en la Figura 49, y el del otro regulador en la Figura 50.

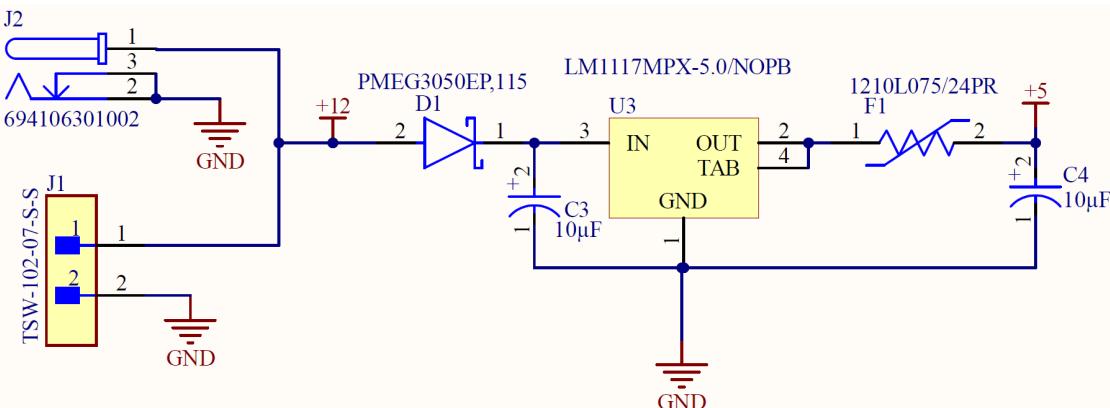


Figura 49. Regulador de tensión de 5 V.

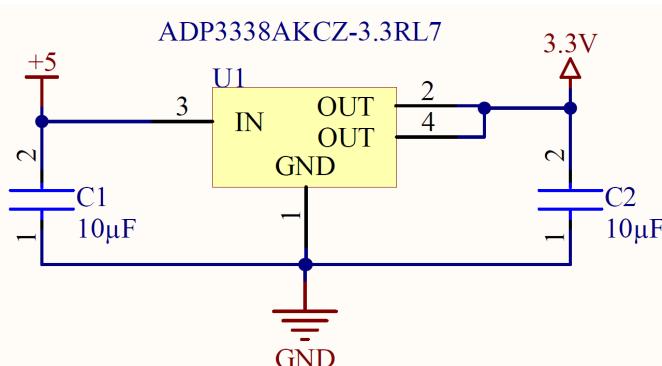


Figura 50 . Regulador de tensión de 3,3 V.

6.1.4. Conexión de las memorias.

Como se ha comentado antes, las memorias se encuentran ubicadas en distintas partes del espacio de memoria, que como se puede apreciar la SRAM está en las direcciones en las que el bit más significativo se encuentra a nivel bajo y en el caso de la EEPROM este bit está a nivel alto. Para conseguir esto es necesario diseñar un decodificador que seleccione una memoria u otra en función del valor de este bit.

En este caso, esto se hace con un circuito muy simple, que se puede apreciar en la Figura 51, que solo necesita una puerta NOT. Lo que hace es habilitar la memoria EEPROM cuando el bit de direcciones “A15” es ‘1’, y esto lo consigue negando este bit para obtener un ‘0’ que activa el “Chip Selecta” de la memoria. Y en el caso de la SRAM no es necesario negarla, ya que este bit es ‘0’ cuando se quiere activar esta memoria, siendo la Tabla 8 la distribución de las memorias.

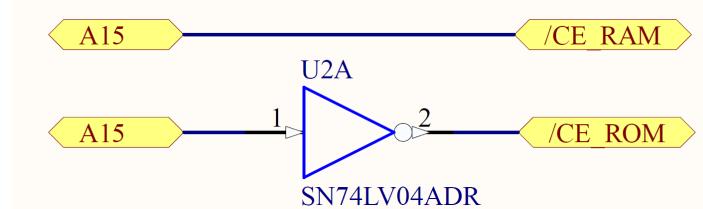


Figura 51. Selector de memorias.

A15	Rango de direcciones	Dispositivo
0	0x0000 – 0x7FFF	SRAM
1	0x8000 – 0xFFFF	EEPROM

Tabla 8. Direccionamiento del espacio de memorias.

6.1.5. Conexión de las entradas del procesador.

Las entradas del procesador deben contar con una resistencia del pull-up o pull-down, en función del valor de reposo de esta señal. Esto se debe hacer para que no se produzcan fallos en el dispositivo, ya que por ejemplo si no se usa la señal /INT, esta entrada se quedaría al aire, pudiendo interpretar el microprocesador que esta entrada está a nivel bajo, lo que produciría que entrara en modo de interrupción. Esto hace que sea necesario colocar una resistencia de pull-up en la placa, para que no se produzca este fallo, forzando un ‘1’ lógico cuando no hay nada conectado.

6.1.6. Planos.

La Figura 52 muestra el aspecto de la modelo 3D de la placa y la PCB fabricada se puede ver en la Figura 53.

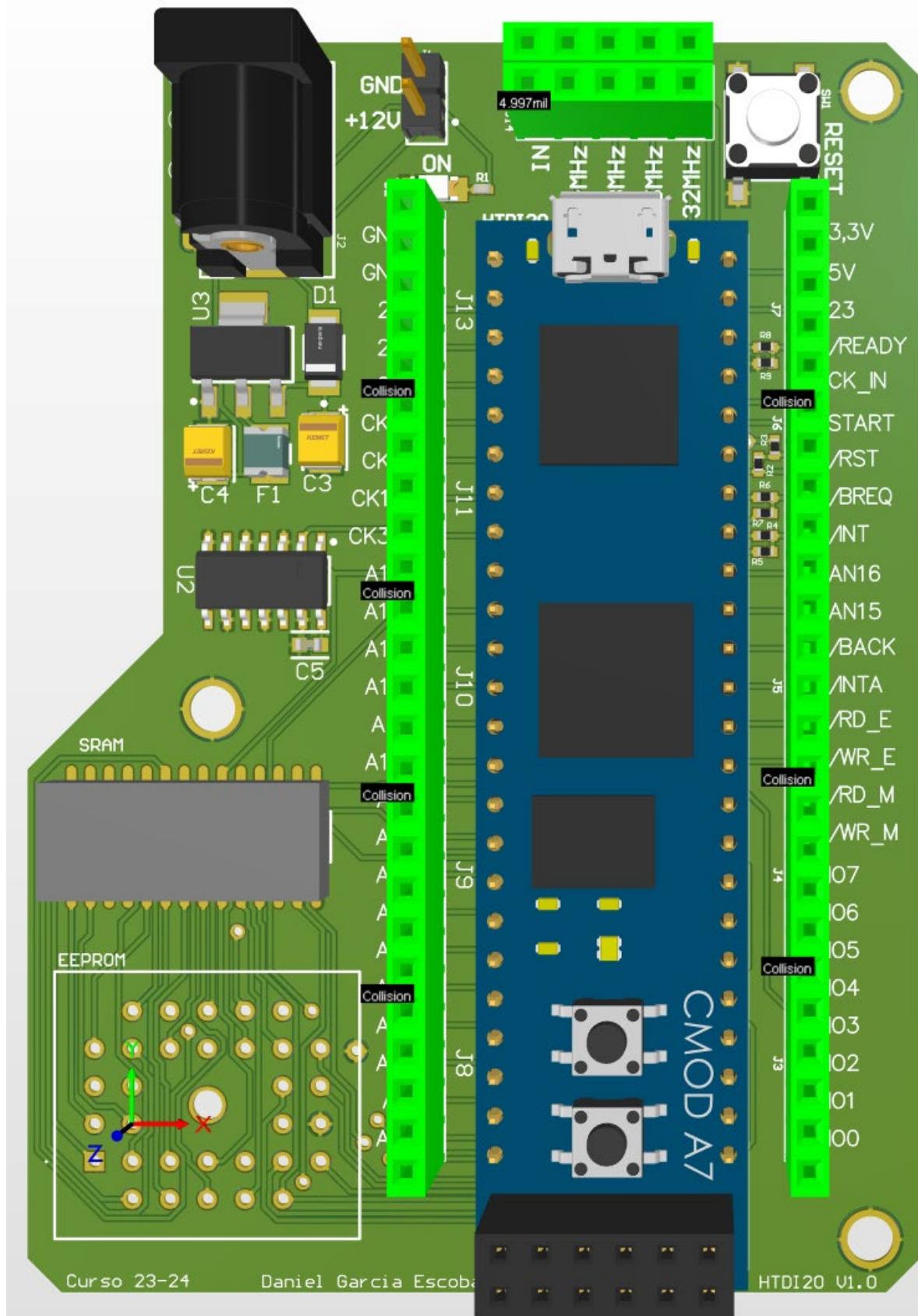


Figura 52. Modelo 3D de la PCB de memorias.

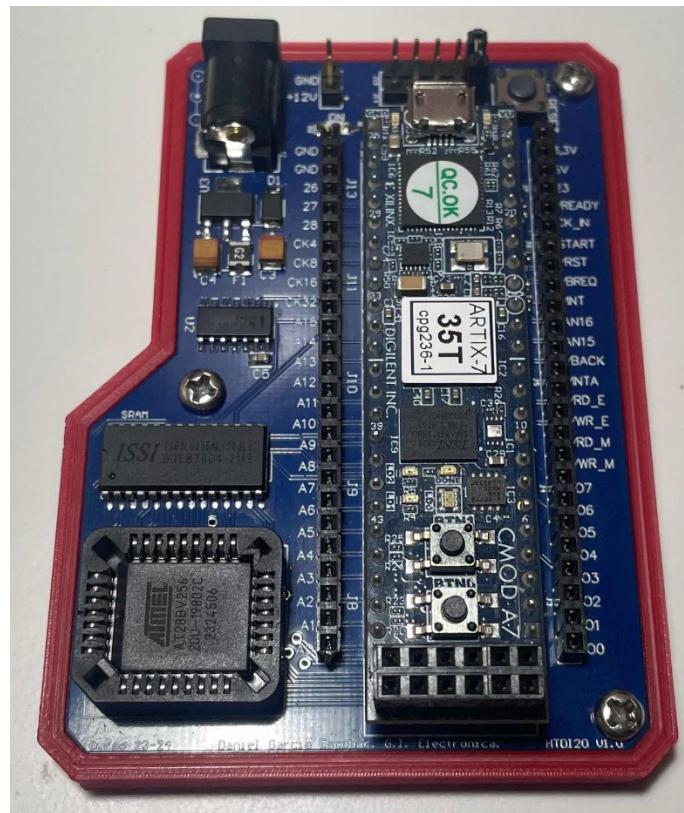


Figura 53. PCB de memorias ensamblada.

6.2. Placa de puerto paralelo.

La finalidad de esta PCB es tener un modo más fácil de interactuar con el HTDI20, agregándole un puerto paralelo de 4 entradas y 4 salidas de 8 bits. Para conseguir que el usuario pueda interactuar más fácil con este puerto paralelo, a las entradas se le han añadido interruptores para seleccionar el valor deseado, y a las salidas se les han conectado unos LEDs. Esta PCB se puede conectar sin necesidad de cable, ya que se acopla en la parte de arriba de la placa de memorias.

6.2.1. Características.

Las características que tiene esta PCB son las siguientes:

- Cuatro puertos paralelos de salida de 8 bits, que ocupan de la dirección 0x00 a la dirección 0x03. Cada bit del puerto cuenta con un led que indica su estado lógico.
- Cuatro puertos paralelos de entrada de 8 bits, que ocupan de la dirección 0x04 a la dirección 0x07. Cada bit del puerto cuenta con un interruptor que modifica su estado lógico.
- Acoplable en la PCB de memorias, permitiendo conectar otra PCB encima de esta o leer los terminales con un analizador lógico.

6.2.2. Esquema de los puertos de entrada.

En este caso, se usa un buffer que cuando tiene activa la salida, vuelca en el bus de datos los valores que ha elegido el usuario con los interruptores. Pero cuando están desactivadas las salidas, las ponga en alta impedancia y no se produzcan colisiones en el bus. El esquema mostrado en la Figura 54, es el circuito que usan los puertos de entrada.

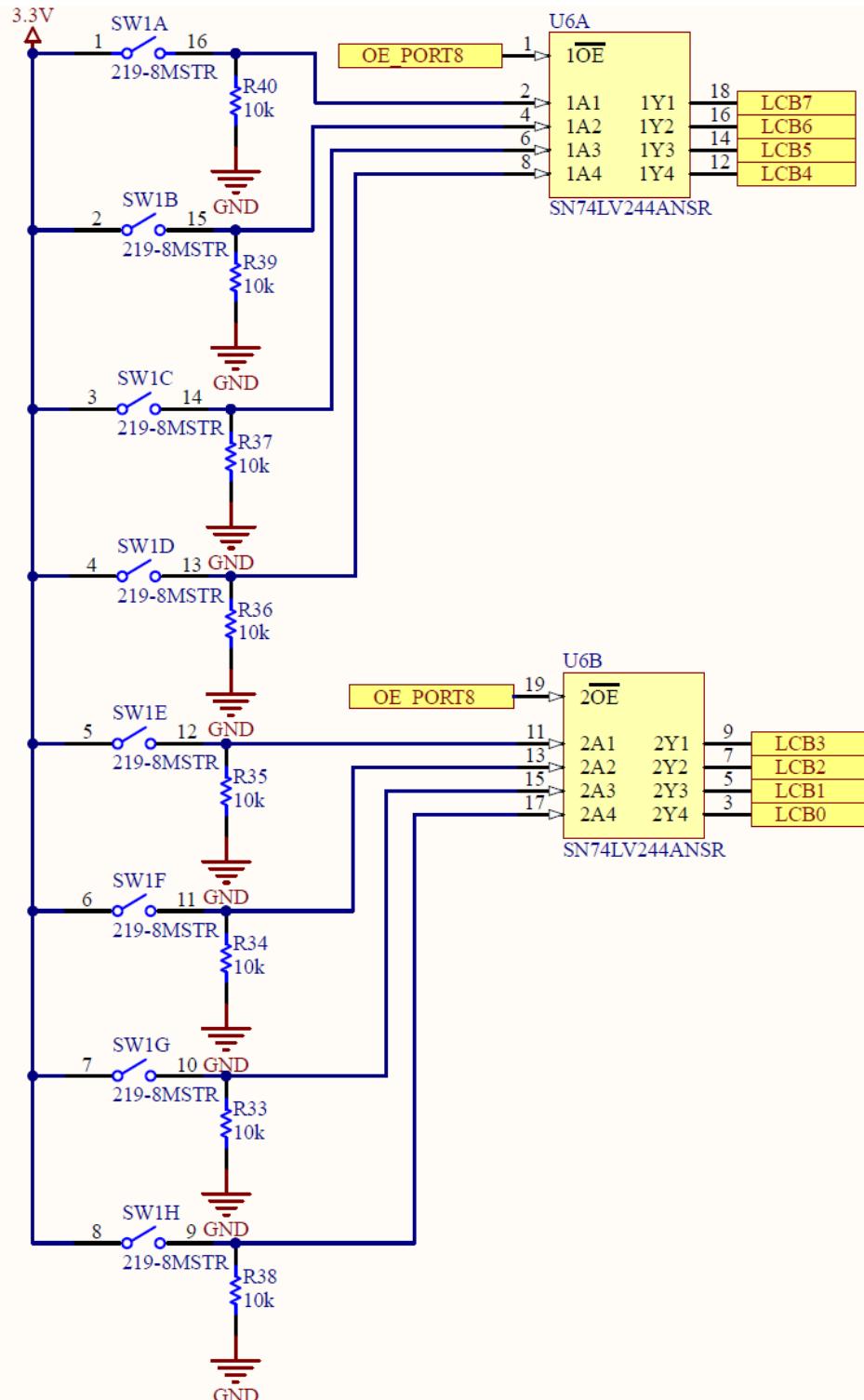


Figura 54. Estructura de los puertos de entrada.

6.2.3. Esquema de los puertos de salida.

Para implementar los puertos paralelos de salida, se usa un registro de 8 bits con la salida siempre activada que, tras un flanco ascendente de la señal de reloj, almacena los datos del bus de datos y enciende unos LEDs en función de su contenido. El esquema de estos puertos sería el que se muestra en la Figura 55.

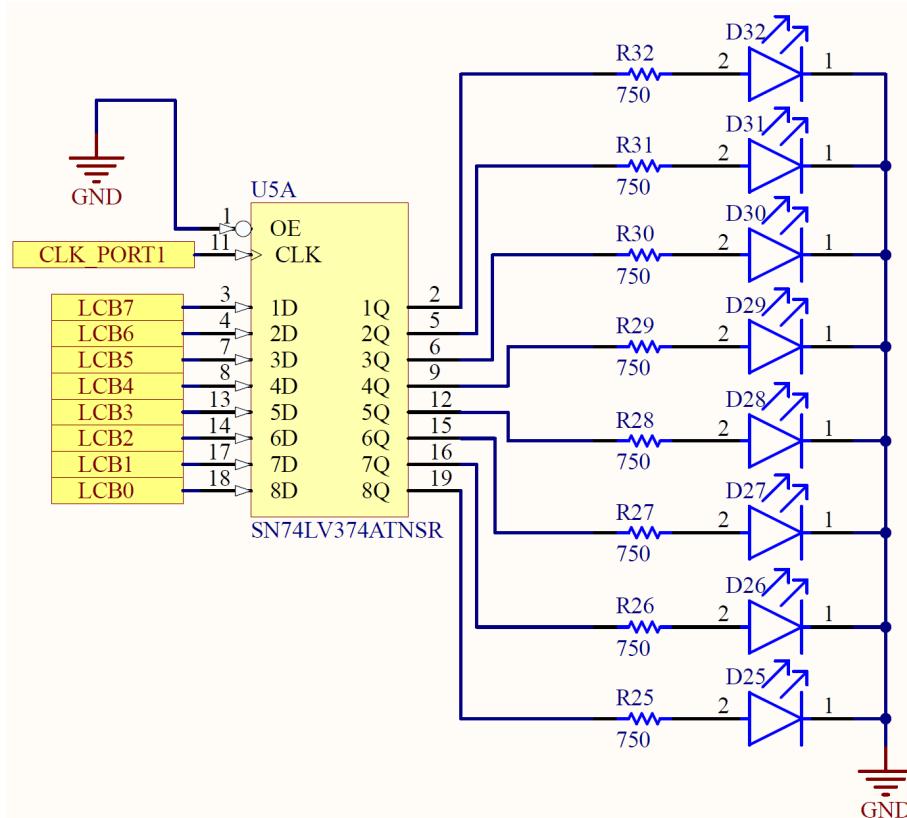


Figura 55. Estructura de los puertos de salida.

6.2.4. Decodificador del puerto paralelo.

Es de vital importancia que a cada dirección solo se le asigne un dispositivo del puerto paralelo, para que no se produzcan colisiones de datos en el bus. Esto se consigue con ayuda de un decodificador de binario a decimal, el cual asigna una dirección a cada uno de los dispositivos disponibles.

Como se ha visto antes, el puerto paralelo se encuentra en las ocho direcciones más bajas del espacio de E/S. Lo que significa que solo se accede a él cuando los 5 bits más significativos de bus de direcciones están a nivel bajo. Esto se implementa con un conjunto de puertas OR, que activan o desactivan el decodificador en función del bus de direcciones, que en este caso solo se activaría cuando los 5 bits más significativos de este bus están a nivel bajo.

Además, es necesario que el decodificador solo se active cuando se está usando el espacio de E/S y no se active cuando se usa el espacio de memorias. Lo que se consigue con una puerta AND que tiene como entradas las señales de escritura y lectura de E/S, y la salida se conecta a la señal de activación del decodificador. Quedando todo el circuito como se ve en la Figura 56.

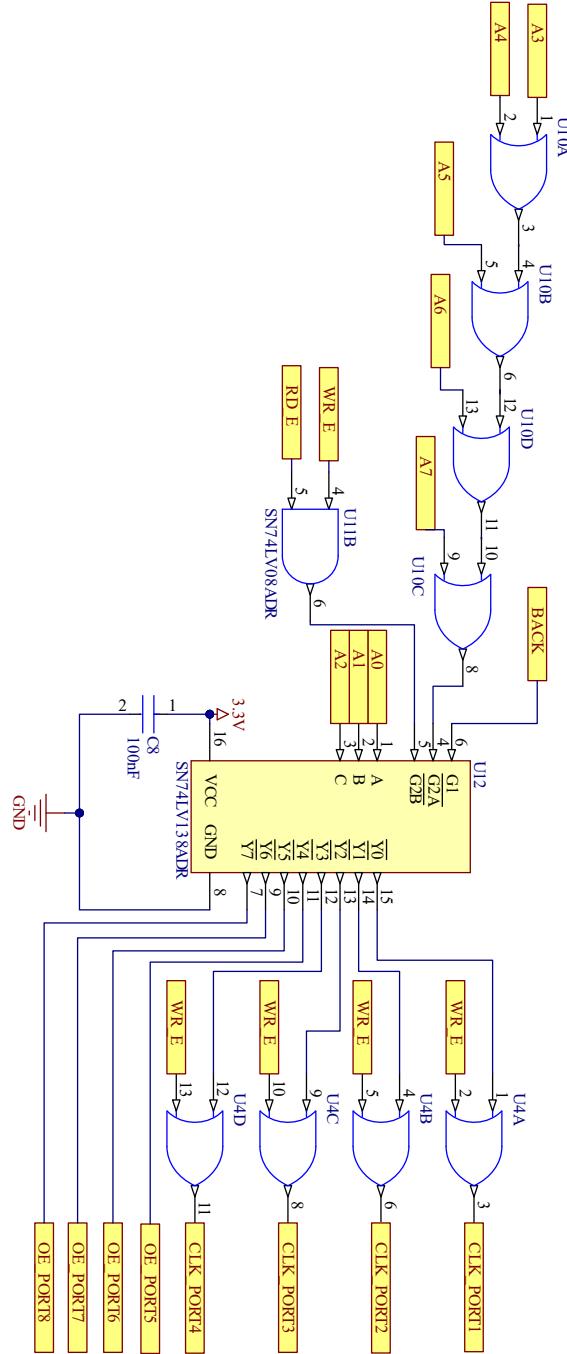


Figura 56. Esquema del decodificador del puerto paralelo.

6.2.5. Planos.

La Figura 57 se muestra el aspecto del modelo 3D de la placa y la PCB fabricada se puede ver en la Figura 58.

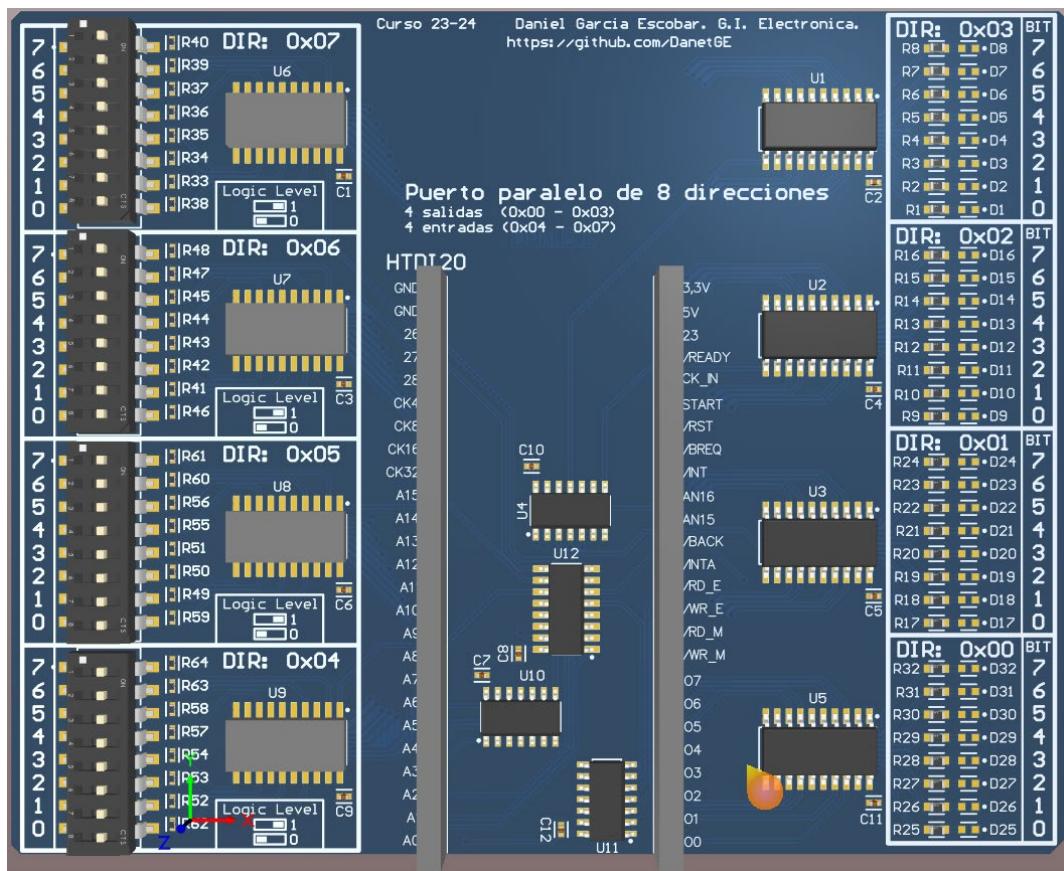


Figura 57. Modelo 3D de la PCB de puerto paralelo.

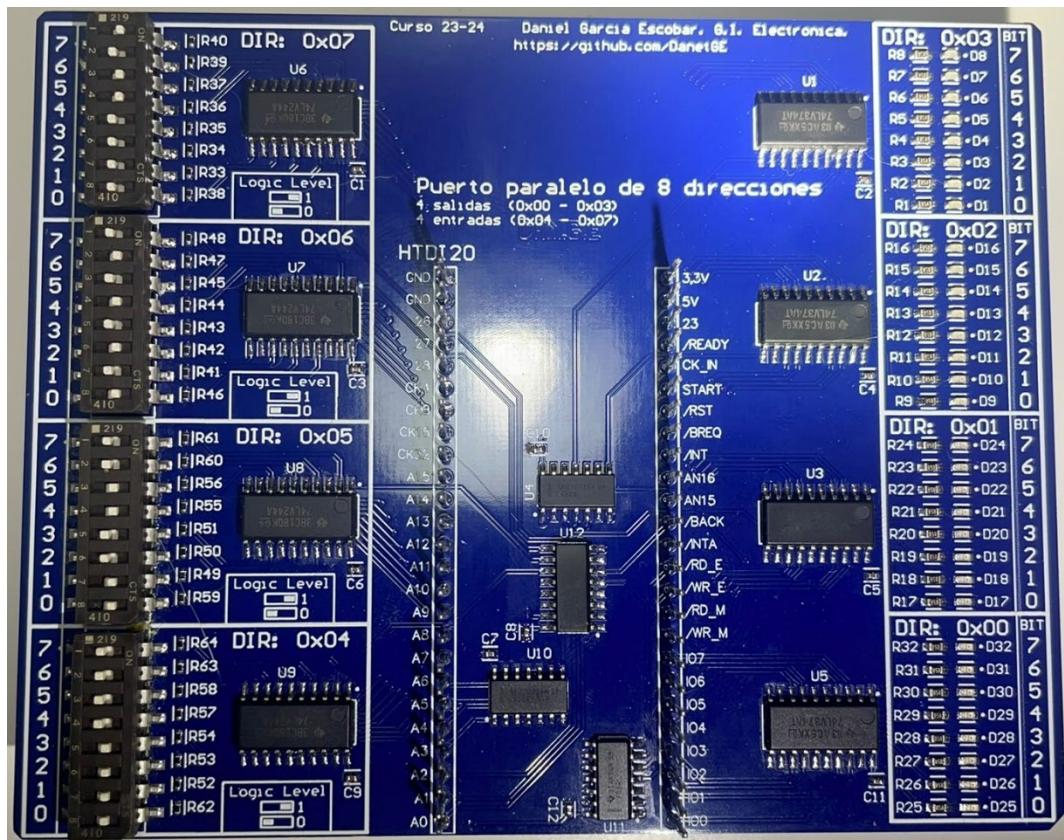


Figura 58. PCB de puerto paralelo ensamblada.

Capítulo 7. Conclusiones.

7.1. Introducción.

Una vez implementado el microprocesador teórico HTDI20 junto con su propio compilador y PCBs, en el presente capítulo se comentan las diferentes competencias adquiridas, así como las posibles investigaciones y desarrollos futuros que se pueden llevar a cabo sobre este proyecto.

7.2. Conclusiones.

El objetivo principal de este Trabajo de Fin de Grado ha sido adquirir conocimientos sobre la programación VHDL y la implementación de dispositivos electrónicos complejos en una FPGA, para llevar a cabo la implementación del HTDI20. Además de esto, también se han adquirido conocimientos en el diseño de PCBs y en la programación C.

Además, como se comenta al principio del documento, los campos que se trabajan en este proyecto son muy necesarios en la actualidad. Como por ejemplo el trabajo con FPGAs, que empieza a ser bastante demandado por las empresas, y se encuentran pocos graduados que hayan invertido mucho tiempo en realizar trabajo práctico con ellas.

7.3. Líneas de trabajo futuras.

Este proyecto tiene una infinidad de campos en los que seguir trabajando, para aumentar sus características y que pueda equiparse a plataformas como Arduino. Algunas de las posibles mejoras que pueden realizar otros alumnos en próximos años, son:

- Aumentar la cantidad de núcleos del dispositivo para incrementar su potencia de cálculo.
- Diseñar nuevos periféricos que amplíen la versatilidad del dispositivo, como puede ser el caso de un controlador de protocolo CAN o timers. Además, como el proyecto se implementa en una FPGA, se pueden diseñar todo tipo de periféricos y que el usuario decida cual añadirle a su proyecto. Pudiendo de este modo adaptarse a gran cantidad de aplicaciones.
- Crear nuevos tipos de compiladores, como puede ser uno para lenguaje C o uno con el lenguaje que usa Arduino, para que lo puedan usar gente que no tengan mucho conocimiento de microprocesadores.

- Incrementar el tamaño de los buses de datos, así como de todos los dispositivos que manejan estos datos, para que pueda procesar datos de mayor tamaño y reducir los tiempos de ejecución.
- Añadirle nuevas Unidades Lógico Aritméticas (ALU) que puedan trabajar con datos de coma flotante o añadirle núcleos de DSP, con los que cuenta el chip de la FPGA.
- Crear un pequeño ordenador con controladores VGA, de teclado y ratón, además de diseñar un pequeño sistema operativo.

Esto se comenzó a hacer al principio del proyecto, diseñando un controlador VGA que permitía mostrar imágenes a 720p. Además de crear un controlador de video inspirado en el de la GameBoy Color.

Bibliografía

- [1] AMD. (s.f.). Obtenido de 7 Series DSP48E1 Slice User Guide:
https://docs.amd.com/v/u/en-US/ug479_7Series_DSP48E1
- [2] Diego, U. C. (s.f.). *LUTs*. University California San Diego.
- [3] Digilent. (04 de Octubre de 2019). *CMOD A7 Reference Manual*. Obtenido de <https://digilent.com/reference/programmable-logic/cmod-a7/reference-manual>
- [4] Escobar, D. G. (2024). *Repositorio de Github del HTDI20*. Obtenido de <https://github.com/DanetGE/HTDI20.git>
- [5] Intel 4004 . (s.f.). Obtenido de The Intel 4004 Microprocessor and the Silicon Gate Technology: <http://intel4004.com>
- [6] Maxfield, M. (s.f.). *Fundamentos de los FPGA - Parte 4: Introducción de los FPGA de Xilinx*. Obtenido de <https://www.digikey.es/es/articles/fundamentals-of-fpgas-part-4-getting-started-with-xilinx-fpgas>
- [7] Paxson, V., Estes, W., & Millaway, J. (10 de Noviembre de 2015). *Lexical Analysis with Flex*. Obtenido de <https://epaperpress.com/lexandyacc/download/flex.pdf>
- [8] PFEIFFER, D. C. (s.f.). *Wikimedia*. Obtenido de <https://commons.wikimedia.org/w/index.php?curid=1441423>
- [9] shivani.mittal. (10 de Abril de 2023). *Flex (Fast Lexical Analyzer Generator)*. Obtenido de Geeks for Geeks: <https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/>
- [10] Sotorrío, P. J. (2021). *Desmitificando el Microprocesador*. Málaga.

ANEXOS

Anexo A: Juego de instrucciones del HTDI20.

A.1. Juego de instrucciones y características.

Grupo	Nº	Nemónico	Operandos	Tiempo	Bytes	Código
Saltos Condicionales	1	JPC	dd	10/9	3	0x00
	2	JPNC	dd	10/9	3	0x01
	3	JPZ	dd	10/9	3	0x02
	4	JPNZ	dd	10/9	3	0x03
	5	JPF0	dd	10/9	3	0x04
	6	JPNF0	dd	10/9	3	0x05
	7	JPF1	dd	10/9	3	0x06
	8	JPNF1	dd	10/9	3	0x07
	9	JPF2	dd	10/9	3	0x08
	10	JPNF2	dd	10/9	3	0x09
	11	JPF3	dd	10/9	3	0x0A
	12	JPNF3	dd	10/9	3	0x0B
Llamadas Condicionales	13	CALLC	dd	15/9	3	0x0C
	14	CALLNC	dd	15/9	3	0x0D
	15	CALLZ	dd	15/9	3	0x0E
	16	CALLNZ	dd	15/9	3	0x0F
Copía Datos	17	COP	B, A	1	4	0x10
	18	COP	C, A	1	4	0x11
	19	COP	A, B	1	4	0x12
	20	COP	C, B	1	4	0x13
	21	COP	A, C	1	4	0x14
	22	COP	B, C	1	4	0x15
	23	COP16	E, D	1	4	0x16
	24	COP16	IX, D	1	4	0x17
	25	COP16	D, E	1	4	0x18
	26	COP16	IX, E	1	4	0x19
	27	COP16	D, IX	1	4	0x1A

	28	COP16	E, IX	1	4	0x1B
	29	COP	A, d	2	6	0x1C
	30	COP	B, d	2	6	0x1D
	31	COP	C, d	2	6	0x1E
	32	COP16	D, dd	3	10	0x1F
	33	COP16	E, dd	3	10	0x20
	34	COP16	IX, dd	3	10	0x21
	35	COP16	SP, dd	3	10	0x22
	36	COP	A, (dd)	3	12	0x23
	37	COP	B, (dd)	3	12	0x24
	38	COP	C, (dd)	3	12	0x25
	39	COP	(dd), A	3	12	0x26
	40	COP	(dd), B	3	12	0x27
	41	COP	(dd), C	3	12	0x28
	42	COP	(dd), d	4	15	0x29
	43	COP	A, (D)	1	6	0x2A
	44	COP	B, (D)	1	6	0x2B
	45	COP	C, (D)	1	6	0x2C
	46	COP	A, (E)	1	6	0x2D
	47	COP	B, (E)	1	6	0x2E
	48	COP	C, (E)	1	6	0x2F
	49	COP	A, (IX)	1	6	0x30
	50	COP	B, (IX)	1	6	0x31
	51	COP	C, (IX)	1	6	0x32
	52	COP	(D), A	1	6	0x33
	53	COP	(E), A	1	6	0x34
	54	COP	(IX), A	1	6	0x35
	55	COP	(D), B	1	6	0x36
	56	COP	(E), B	1	6	0x37
	57	COP	(IX), B	1	6	0x38
	58	COP	(D), C	1	6	0x39
	59	COP	(E), C	1	6	0x3A

	60	COP	(IX), C	1	6	0x3B
Manejo de ES	61	IN	A, (d)	2	9	0x3C
	62	IN	B, (d)	2	9	0x3D
	63	IN	C, (d)	2	9	0x3E
	64	IN	A, (B)	1	6	0x3F
	65	IN	A, (C)	1	6	0x40
	66	IN	(dd), (d')	4	18	0x41
	67	OUT	(d), A	2	9	0x42
	68	OUT	(d), B	2	9	0x43
	69	OUT	(d), C	2	9	0x44
	70	OUT	(d), d'	3	12	0x45
	71	OUT	(B), A	1	6	0x46
	72	OUT	(C), A	1	6	0x47
	73	OUT	(d), (dd')	4	18	0x48
Copia de Datos	75	COP	(E), (D)	1	9	0x4A
	76	COP	(IX), (D)	1	9	0x4B
	77	COP	(D), (E)	1	9	0x4C
	78	COP	(IX), (E)	1	9	0x4D
	79	COP	(D), (IX)	1	9	0x4E
	80	COP	(E), (IX)	1	9	0x4F
	81	COP16	(dd), D	3	16	0x50
	82	COP16	(dd), E	3	16	0x51
	83	COP16	(dd), IX	3	16	0x52
	84	COP16	(dd), SP	3	16	0x53
	85	COP16	D, (dd)	3	16	0x54
	86	COP16	E, (dd)	3	16	0x55
	87	COP16	IX, (dd)	3	16	0x56
	88	COP16	SP, (dd)	3	16	0x57
	91	PUSH	A	1	6	0x5A
	92	PUSH	B	1	6	0x5B
	93	PUSH	C	1	6	0x5C
	94	PUSH	PSW	1	6	0x5D

Manejo de pila	95	PUSH	D	1	10	0x5E
	96	PUSH	E	1	10	0x5F
	97	PUSH	IX	1	10	0x60
	98	PUSH	PC	1	10	0x61
	99	POP	A	1	7	0x62
	100	POP	B	1	7	0x63
	101	POP	C	1	7	0x64
	102	POP	PSW	1	7	0x65
	103	POP	D	1	11	0x66
	104	POP	E	1	11	0x67
	105	POP	IX	1	11	0x68
	106	POP	PC	1	11	0x69
Funciones Lógicas	107	AND	A, A	1	4	0x6A
	108	AND	A, B	1	5	0x6B
	109	AND	A, C	1	5	0x6C
	110	AND	A, d	2	6	0x6D
	111	OR	A, A	1	4	0x6E
	112	OR	A, B	1	5	0x6F
	113	OR	A, C	1	5	0x70
	114	OR	A, d	2	6	0x71
	115	XOR	A, A	1	4	0x72
	116	XOR	A, B	1	5	0x73
	117	XOR	A, C	1	5	0x74
	118	XOR	A, d	2	6	0x75
	119	INV	A	1	4	0x76
	120	SL	A	1	4	0x77
Operaciones de rotación	121	SR	A	1	4	0x78
	122	SLC	A	1	4	0x79
	123	SRC	A	1	4	0x7A
	124	RL	A	1	4	0x7B
	125	RR	A	1	4	0x7C
	126	RLC	A	1	4	0x7D

	127	RRC	A	1	4	0x7E
Funciones Aritméticas	128	ADD	A, A	1	4	0x7F
	129	ADD	A, B	1	5	0x80
	130	ADD	A, C	1	5	0x81
	131	ADD	A, d	2	6	0x82
	132	ADD	A, (dd)	3	12	0x83
	133	ADD	A, (D)	1	6	0x84
	134	ADD	A, (E)	1	6	0x85
	135	ADD	A, (IX)	1	6	0x86
	136	SUB	A, A	1	4	0x87
	137	SUB	A, B	1	5	0x88
	138	SUB	A, C	1	5	0x89
	139	SUB	A, d	2	6	0x8A
	140	SUB	A, (dd)	3	12	0x8B
	141	SUB	A, (D)	1	6	0x8C
	142	SUB	A, (E)	1	6	0x8D
	143	SUB	A, (IX)	1	6	0x8E
	144	INC	A	1	4	0x8F
	145	INC	B	1	4	0x90
	146	INC	C	1	4	0x91
	147	INC	(D)	1	9	0x92
	148	INC	(E)	1	9	0x93
	149	INC	(IX)	1	9	0x94
	150	INC	(dd)	3	15	0x95
	151	INC16	D	1	4	0x96
	152	INC16	E	1	4	0x97
	153	INC16	IX	1	4	0x98
	154	INC16	(D)	1	16	0x99
	155	INC16	(E)	1	16	0x9A
	156	INC16	(IX)	1	16	0x9B
	158	DEC	A	1	4	0x9D
	159	DEC	B	1	4	0x9E

Funciones Aritméticas	160	DEC	C	1	4	0x9F
	161	DEC	(D)	1	9	0xA0
	162	DEC	(E)	1	9	0xA1
	163	DEC	(IX)	1	9	0xA2
	164	DEC	(dd)	3	15	0xA3
	165	DEC16	D	1	4	0xA4
	166	DEC16	E	1	4	0xA5
	167	DEC16	IX	1	4	0xA6
	168	DEC16	(D)	1	16	0xA7
	169	DEC16	(E)	1	16	0xA8
	170	DEC16	(IX)	1	16	0xA9
Salto	172	JMP	dd	3	10	0xAB
Llamada	173	CALL	dd	3	15	0xAC
Retornos	174	RETURN	---	1	11	0xAD
	175	RETINT	---	1	11	0xAE
Control	176	EI	---	1	4	0xAF
	177	DI	---	1	4	0xB0
	178	SIM0	---	1	4	0xB1
	179	SIM1	---	1	4	0xB2
	180	NOP	---	1	4	0xB3
	181	SCF	---	1	4	0xB4
	182	CCF	---	1	4	0xB5
	183	SF0	---	1	4	0xB6
	184	CF0	---	1	4	0xB7
	185	SF1	---	1	4	0xB8
	186	CF1	---	1	4	0xB9
	187	SF2	---	1	4	0xBA
	188	CF2	---	1	4	0xBB
	189	SF3	---	1	4	0xBC
	190	CF3	---	1	4	0xBD
	191	MINT0	---	1	4	0xBE
	192	MINT1	---	1	4	0xBF

Básicas	252	Inicio Interrupción	1	7	0xFB
	253	Autovectorizadas	1	1	0xFC
	254	Vectorizadas	1	13	0xFD
	255	Búsqueda	1	3	0xFE
	256	Inicio	1	1	0xFF

Tabla 9. Juego de instrucciones con sus características.

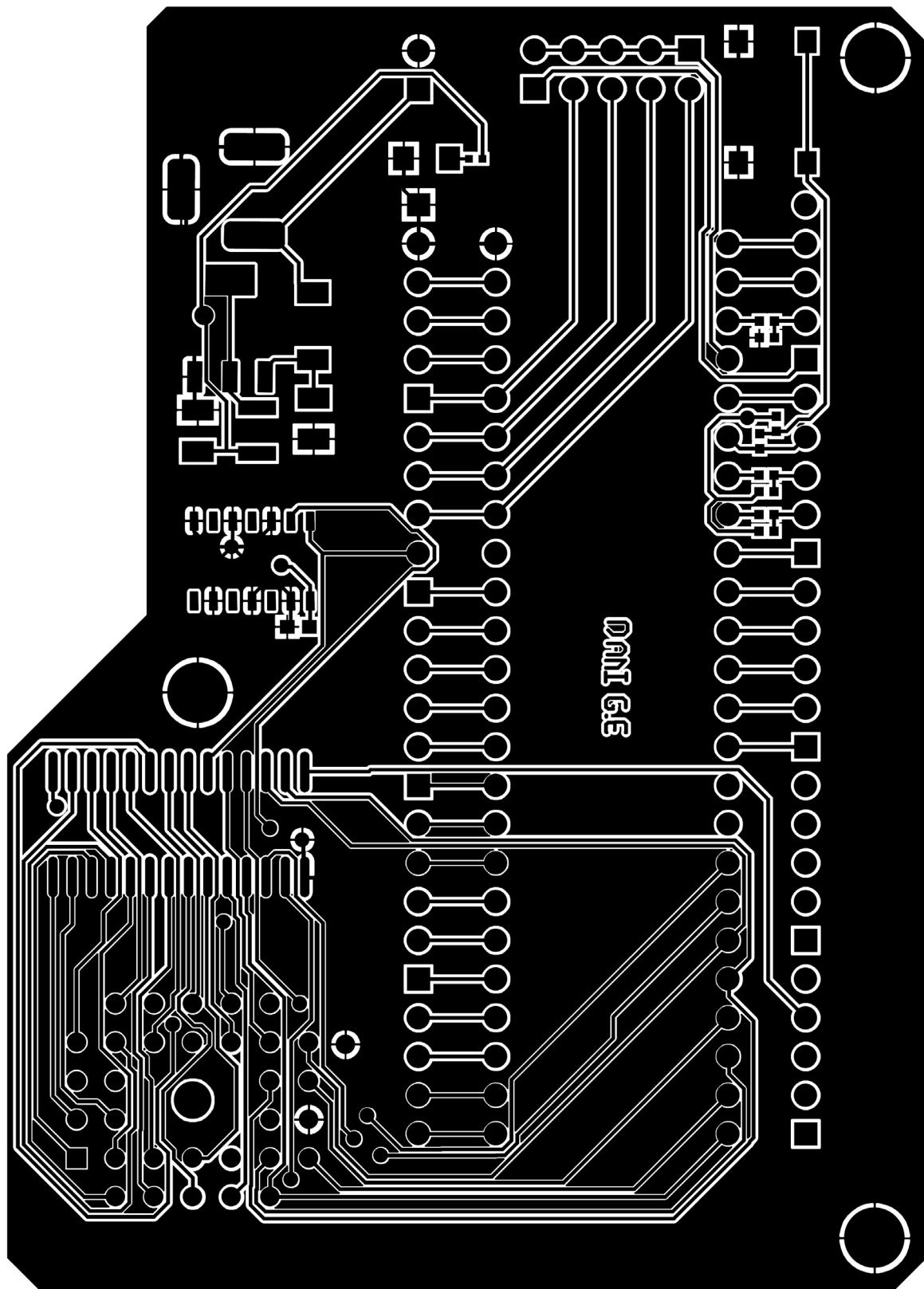
Anexo B: Archivos de fabricación de las PCBs.**B.1. *PCB de memorias.***

Figura 59. Top layer PCB de memorias.

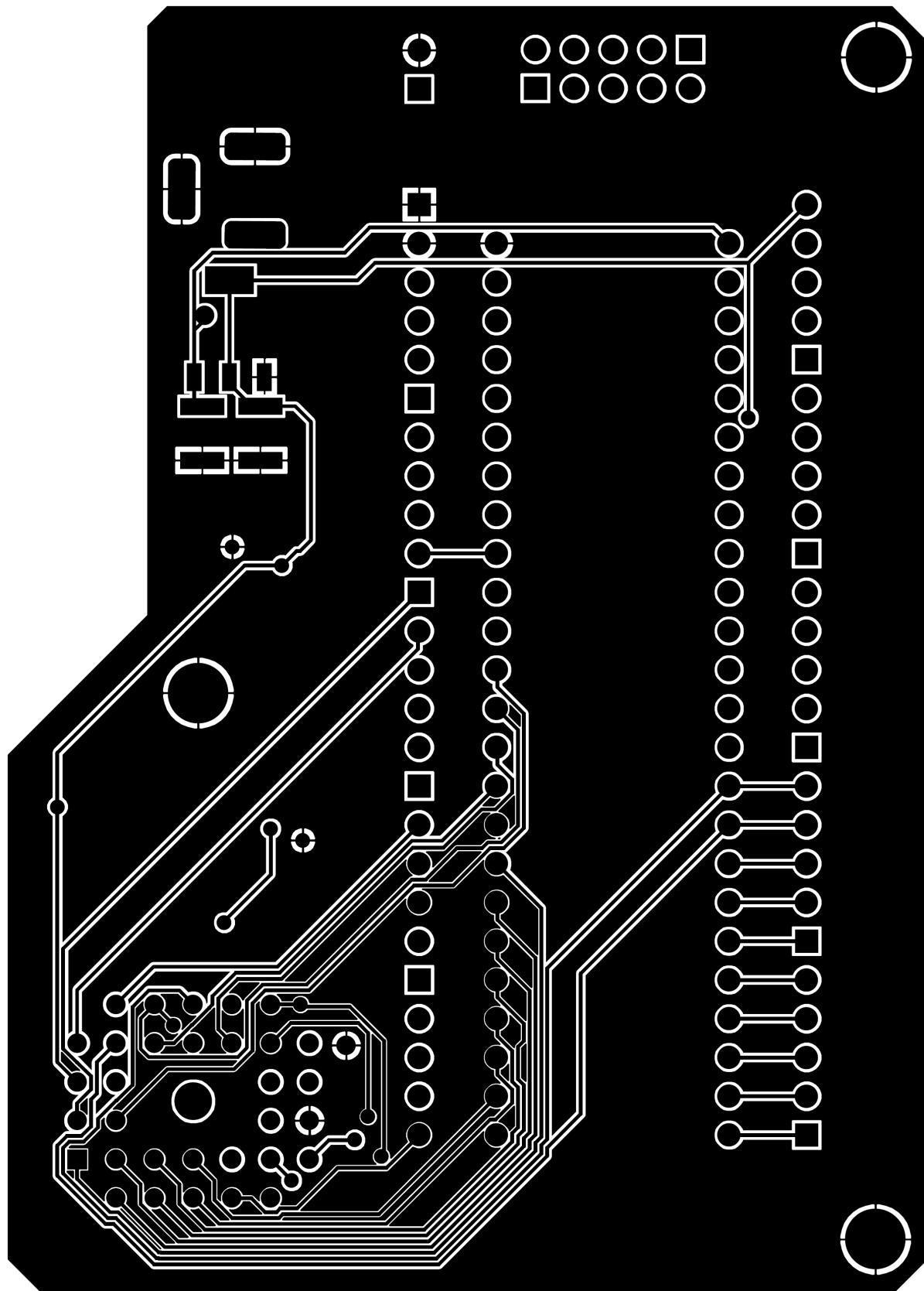


Figura 60. Bottom layer PCB de memorias.

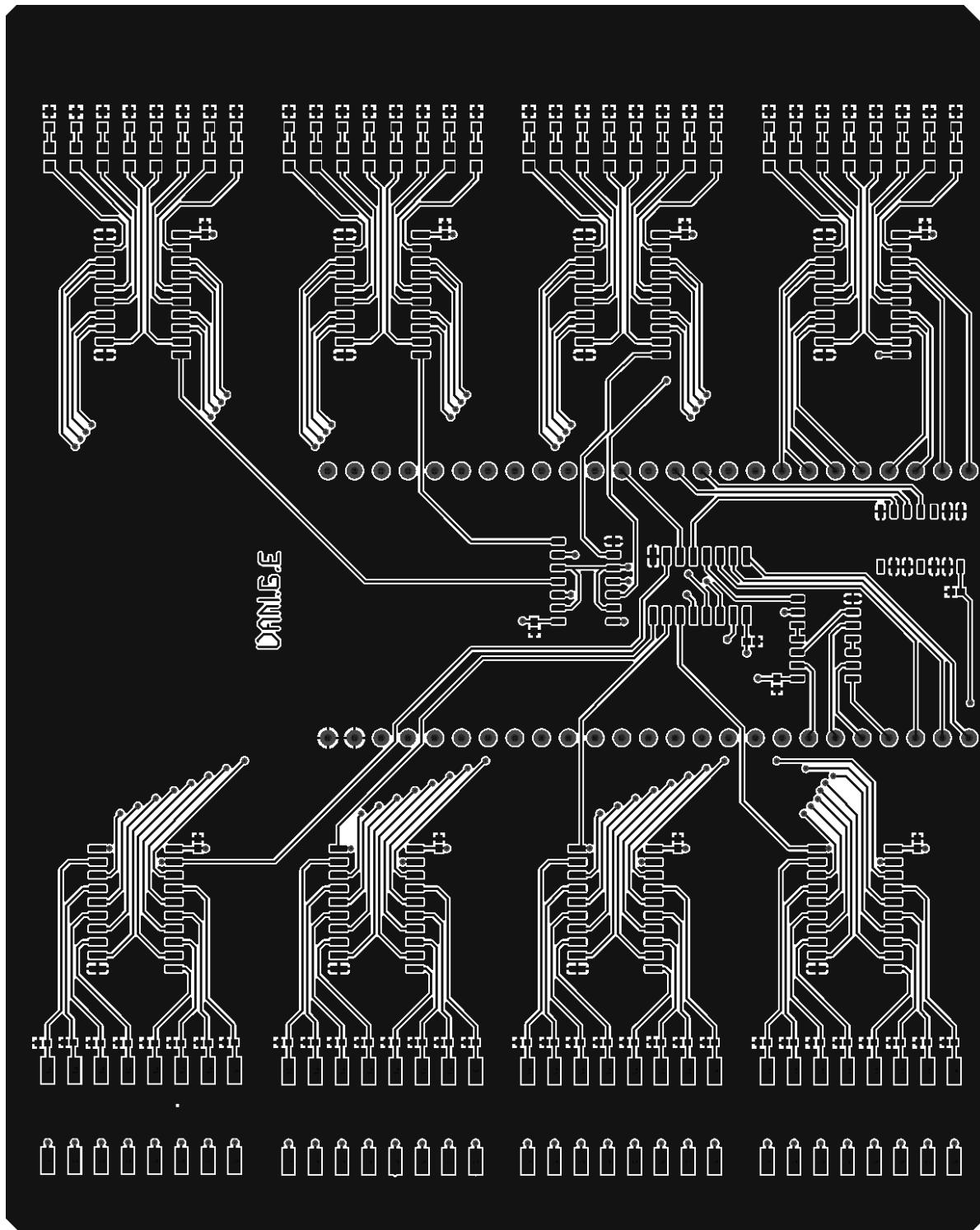
B.2. PCB de puerto paralelo.

Figura 61. Top layer PCB de puerto paralelo.

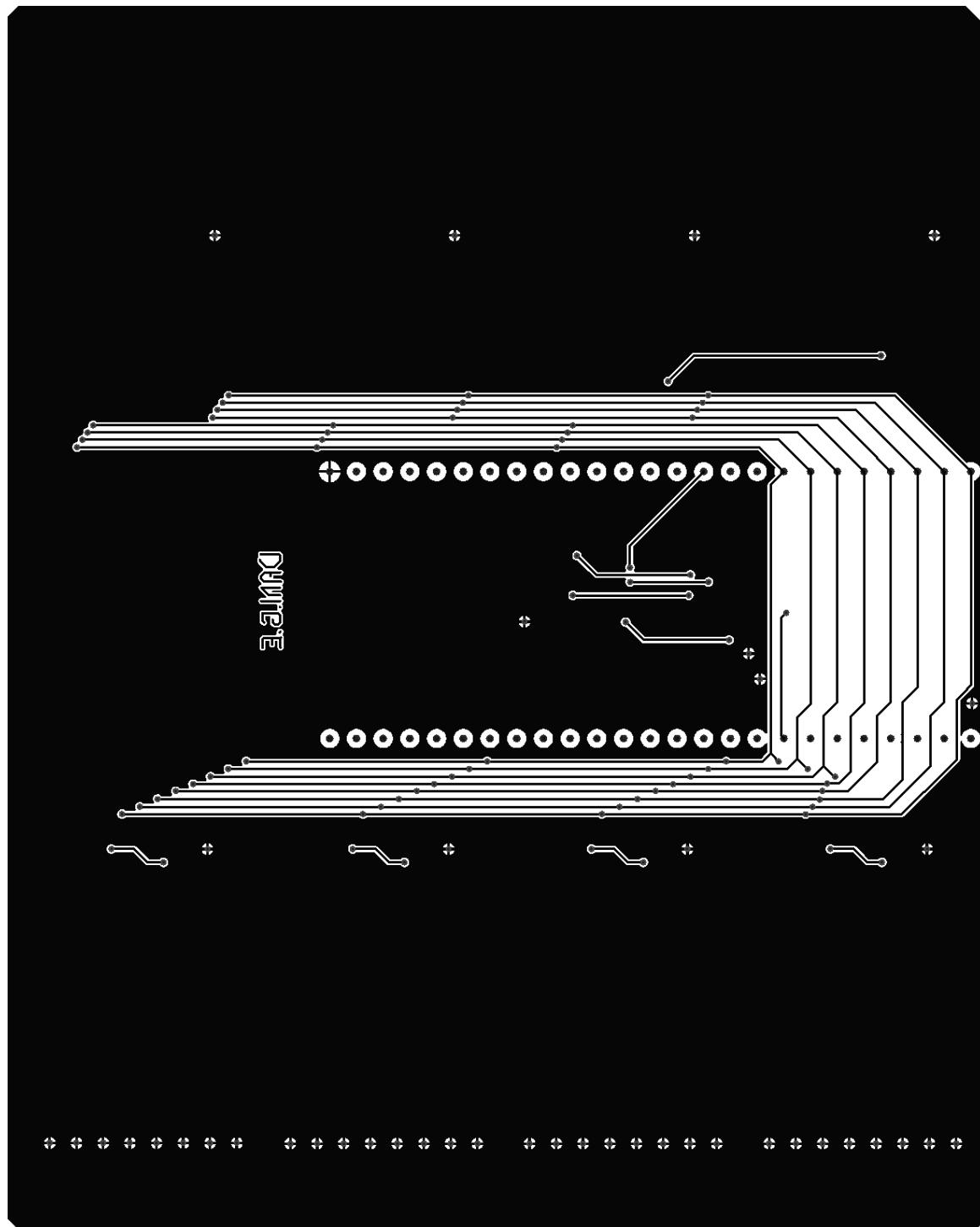


Figura 62. Bottom layer PCB de puerto paralelo.

Anexo C: Bill of materials (BOM)

Referencia	Fabricante	Descripción	Cantidad	Precio Unidad	Precio Total	PCB
C1005X7R1C104K050BC	TDK	Cond. 100 nF, 0402	12	0,028 €	0,336 €	Paralelo
LS L29K-G1J2-1	OSRAM	LED rojo, 0402	32	0,272 €	8,704 €	Paralelo
CPF0603F750RC1	TE Connectivity	Res. 750 Ω, 0603	32	0,048 €	1,536 €	Paralelo
ERA-2AED103X	Panasonic	Res. 10 kΩ, 0402	32	0,054 €	1,728 €	Paralelo
219-8MSTR	CTS	Interruptor 8 bits SMD	4	0,986 €	3,944 €	Paralelo
SN74LV374ATNSR	Texas Instruments	Registro de 8 bits	4	0,539 €	2,156 €	Paralelo
SN74LV32ADR	Texas Instruments	Puertas OR	1	0,512 €	0,512 €	Paralelo
SN74LV244ANSR	Texas Instruments	Buffer de 8 bits	4	0,809 €	3,236 €	Paralelo
SN74LV08ADR	Texas Instruments	Puertas AND	1	0,316 €	0,316 €	Paralelo
SN74LV138ADR	Texas Instruments	Decodificador BCD	1	0,502 €	0,502 €	Paralelo
TSW-150-13-T-S	Samtec	Conecotor DIP macho de 52 pines	2	2,890 €	5,780 €	Paralelo
PCB puerto paralelo	PCBWAY	PCB del puerto paralelo	5	5,344 €	26,720 €	Paralelo
Stencil puerto paralelo	PCBWAY	Stencil del puerto paralelo	1	9,360 €	9,360 €	Paralelo
410-328-35	Digilent	CMOD A7	1	119,620 €	119,620 €	Memorias
AT28BV256-20JU	Microchip	EEPROM 32 kB, 3,3 V	1	11,170 €	11,170 €	Memorias
IS61LV256AL-10JLI-TR	ISSI	SRAM 32 kB, 3,3 V	1	1,280 €	1,280 €	Memorias
LM1117MPX-5.0/NOPB	Texas Instruments	Regulador de 5 V	1	1,060 €	1,060 €	Memorias
SN74LV04ADR	Texas Instruments	Puertas NOT	1	0,474 €	0,474 €	Memorias
ADP3338AKCZ-3.3RL7	Analog Devices	Regulador de 3,3 V	1	3,390 €	3,390 €	Memorias
430182043816	Wurth Elektronik	Pulsador Táctil SMD	1	0,493 €	0,493 €	Memorias
ERJ-2RKF1002X	Panasonic	Res. 10 kΩ, 0402	8	0,015 €	0,120 €	Memorias
ERJ-2RKF91R0X	Panasonic	Res. 91 Ω, 0402	1	0,093 €	0,093 €	Memorias
694106301002	Wurth Elektronik	Conecotor circular DC	1	0,949 €	0,949 €	Memorias
TSW-102-07-S-S	Samtec	Conecotor DIP macho de 2 pines	5	0,195 €	0,975 €	Memorias
1210L075/24PR	Littelfuse	Fusible PTC 0,7 A. 1210	1	0,902 €	0,902 €	Memorias
150120GS75000	Wurth Elektronik	LED verde, 0402	1	0,214 €	0,214 €	Memorias
PMEG3050EP,115	Nexperia	Diodo Schottky	1	0,502 €	0,502 €	Memorias
T491B106K025AT	KEMET	Cond. 10 uF, tántalo, 1311	2	0,846 €	1,692 €	Memorias
C3225X7R1E106K250AC	TDK	Cond. 10 uF, X7R, 1210	2	0,577 €	1,154 €	Memorias
SLW-150-01-T-S	Samtec	Conecotor DIP hembra de 50 pines	2	3,530 €	7,060 €	Memorias
RMA591NL-5M	Chip Quik	Flux para soldadura	1	9,250 €	9,250 €	Memorias
0603ZC104JAT2A	KYOCERA	Cond. 0,1 uF, X7R, 0603	1	0,233 €	0,233 €	Memorias
PCB Memorias	PCBWAY	PCB de memorias	5	0,936 €	4,680 €	Memorias
609002115121	Wurth Elektronik	Jumper de 2 pines	1	0,283 €	0,283 €	
TS391AX50	Chip Quik	Pasta de soldadura	1	12,970 €	12,970 €	
Mano de Obra		Mano de obra	48	11,250 €	540,000 €	
Total				783,394 €		

Tabla 10. Bill of Material del trabajo.