UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Adaptación de una implementación básica de microprocesador RISC-V segmentado para ser usada en sistemas SOC basados en bus Avalon

Pablo Martínez Sánchez

Tutor: Dr. José María Monzó Ferrer

Cotutor: Dr. Antonio Martínez Millana

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2020-21

Valencia, 02 de Julio de 2021

Resumen

El objetivo del presente trabajo es adaptar un core IP de un microprocesador RISC-V segmentado para que pueda ser usado en sistemas SOC basados en buses Avalon.

El core IP adaptado implementa una arquitectura RISC-V segmentada con soporte parcial del juego de instrucciones (ISA) RV32IM, y está desarrollado completamente en System Verilog. La característica principal de esta implementación es el uso de una arquitectura abierta y su bajo consumo [1]; lo que permitiría su uso en sistemas SOC desarrollados sobre dispositivos FPGA de bajo coste. El objetivo principal de este proyecto es diseñar y verificar nuevos módulos para la adaptación del microprocesador al estándar de bus Avalon y su integración en el entorno de desarrollo de sistemas SOC de Intel FPGA Platform Designer.

El resultado del trabajo permitirá crear sistemas SOC donde el microprocesador es interconectado con diferentes periféricos de forma sencilla a través de la herramienta Platform Designer, para lo cual será también necesario el desarrollo de un software y un hardware que permita realizar la programación, el control y la depuración del microprocesador desde un sistema externo.

Resum

L'objectiu d'aquest treball es adaptar un core IP d'un microprocessador RISC-V segmentat per a poder ser utilitzat en sistemes SOC basats en busos Avalon.

El core IP adaptat implementa una arquitectura RISC-V segmentada amb suport parcial del joc d'instruccións (ISA) RV32IM, i es troba desenvolupat completament en System Verilog. La característica principal d'aquesta implementació es l'us d'una arquitectura oberta i el seu baix consúm [1]; el que permet el seu us en un sistema SOC desenvolupat sobre dispositius FPGA de baix cost. L'objectiu principal d'aquest projecte es disenyar i verificar nous móduls per a l'adaptació del microprocessador al estàndard de busos Avalon i la seua integració en l'entorn de desenvolupament de sistemes SOC de Intel FPGA Platform Designer.

El resultat del treball permetrà crear sistemes SOC on el microprocessador es interconectat amb diferents perifèrics de forma senzilla a través de la ferramenta Platform Designer, per cual tambè serà necessari el desenvolupament d'n software i un hardware que permeta realitzar la programació, el control i depuració del microprocessador des d'un sistema extern.

Abstract

The objective of this work is to adapt a core IP of a segmented RISC-V microprocessor to be used in SOC systems based on Avalon buses.

The adapted core IP implements a segmented RISC-V architecture with partial support to the (ISA) RV32IM instruction set, and it is completely developed on System Verilog. The main characteristic of this implementation is the usage of an open architecture and its low consume [1]; this would allow its use in SOC systems developed over low-cost FPGA devices. The main objective of this project is to design and verify new modules to adapt the microprocessor to an Avalon bus standard and its integration in a SOC systems development environment of Intel FPGA Platform

Designer.

The result of this work would allow to create SOC systems where the microprocessor is interconnected with different peripherals easily through the Platform Designer tool, for which is also needed the development of a software and hardware to allow to program, to control and to debug the microprocessor from an external device.

Índice general

1. Introducción **1**

1.1. Objetivos 2

1.2. Plan de trabajo 3

2. Marco teórico **4**

2.1. Avalon Memory Mapped 8

2.1.1. Funcionamiento del módulo Avalon Memory Mapped Slave 8

2.1.2. Funcionamiento del módulo Avalon Memory Mapped Master 10

3. Desarrollo práctico **13**

4. Interfaz Avalon MM Master en RISC-V **15**

4.1. Integración a Platform Designer 18

5. Módulo de Debug **22**

6. Periférico de comunicación serie UART **27**

6.1. Aplicación de control 30

7. Verificación **33**

7.1. Simulaciones del sistema 33

7.1.1. Adaptación del microprocesador segmentado 34

7.1.2. Periférico UART 35

7.1.3. Sistema de debug integrado en el microprocesador 37

8. Implementación Hardware **40**

9. Conclusiones **45**

10. Lineas futuras **47**

Bibliografía **48**

Índice de figuras

1.1. Ejemplo de un SoC [2] 1
1.2. Diagrama de Gantt 3

2.1. Formatos de instrucciones 4
2.2. Analogía pipeline [6] 6
2.3. Esquema del microprocesador segmentado [7] 6
2.4. Ejemplo microprocesador segmentado [7] 7
2.5. Ejemplo de un sistema Avalon [9] 8
2.6. Estructura interface Slave [10] 9
2.7. Ejemplo periférico Avalon MM Slave [11] 10
2.8. Ejemplo conexión entre diferentes buses [12][11] 10
2.9. Ejemplo conexión *interface Master* [12] 11

3.1. Esquema general del sistema a implementar 13

4.1. Esquema microprocesador 15
4.2. Módulo de control del Program Counter (PC) 16
4.3. Esquema del microprocesador integrando los buses Avalon 17
4.4. Señales Platform Designer 18
4.5. Temporización del módulo Avalon MM Master 19
4.6. Módulo microprocesador en Platorm Designer 20
4.7. Sistema completo Platform Designer 21

5.1. Esquema general del sistema final 22
5.2. Esquema de bloques interfaz Avalon MM Slave debug 23
5.3. Distribución de bits del registro 0 23
5.4. Señales pertenecientes al sistema de debug 26

6.1. Esquemático del periférico UART 27
6.2. Esquema transmisión estándar de 8 bits 28
6.3. Aplicación de control debug 30
6.4. Ventana de recepción de datos 31

7.1. Modificación del archivo msim_setup.tcl 34
7.2. Código a ejecutar en primera instancia 34
7.3. Resultado de la simulación del módulo **controlEnables** 35
7.4. Código que simula el comportamiento de una UART de transmisión 36
7.5. Resultado de la simulación del periférico UART 36
7.6. Resultado de la simulación de escritura de datos 37

7.7. Contenido de las memorias en la simulación 37
7.8. Resultado de la simulación de lectura de datos 38
7.9. Resultado de la simulación de lectura del PC actual y siguiente 38
7.10. Código utilizado finalmente 39
7.11. Resultado de la simulación de ejecución por pasos 39

8.1. Sistema final 41
8.2. Generar el sistema para la compilación en Verilog 41
8.3. Plantilla de instanciación proporcionada 42
8.4. Herramienta Pin Planner 42
8.5. Programación de la placa de evaluación 43

Índice de tablas

2.1. Lista de instrucciones implementadas 5
2.2. Lista de señales integradas en la *interface Slave* 9
2.3. Lista de señales integradas en la *interface Master* I 11
2.4. Lista de señales integradas en la *interface Master* II 12

5.1. Señales de control del módulo **interconexLogic** 24

6.1. Señales de control del periférico UART 29

8.1. Características del sistema final 43

Capítulo 1

Introducción

El objetivo principal de este trabajo es la adaptación del microprocesador previamente implementado a un estándar de buses de interconexión denominado Avalon Memory Mapped Interface, propiedad de Intel. Todo esto tiene como finalidad el establecimiento de una sencilla conexión entre diferentes periféricos sin tener que realizar modificaciones sobre las interfaces de entrada y salida del microprocesador, lo que proporciona una conexión universal entre este y los diferentes componentes externos. Los beneficios recibidos por esta adaptación destacan principalmente a la hora de implementar el código en un hardware específico, ya que se dispone de una fácil integración de todos los periféricos disponibles en este mediante la herramienta integrada en el paquete de software Quartus Prime denominada Platform Designer, que permite la conexión entre diferentes sistemas basados en buses Avalon. Esto facilita la integración de todo tipo de sistemas en un mismo componente, lo que se conoce como un sistema SoC (System on a Chip), el cual está conformado por diferentes sistemas que se integran físicamente en un chip haciendo así que su fabricación sea más barata y su integración en todo tipo de componentes sea más sencilla. Ya que en un solo chip se encuentra un sistema completo conformado por diferentes subsistemas, como puede ser el propio microprocesador y los controladores de los diferentes subsistemas como se observa en la *figura 1.1*.

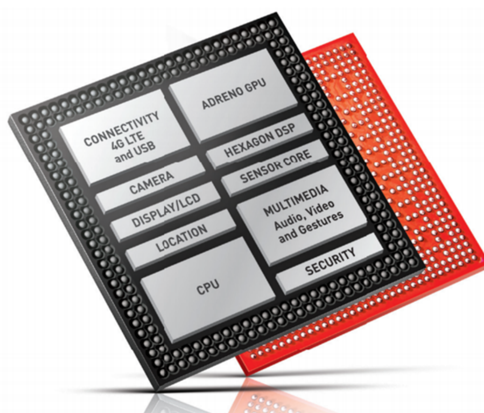

Figura 1.1: Ejemplo de un SoC [2]

Como de punto de partida de este trabajo se parte de un proyecto realizado en el transcurso del grado, concretamente en la asignatura ISDIGI (Integración de sistemas digitales). Este proyecto consta principalmente de un microprocesador con pipeline segmentada basado en la arquitectura RISC-V con soporte parcial para ISA RV32I [3] descrito en el lenguaje de descripción hardware System Verilog. Algunas de las principales ventajas de utilizar esta implementación es la ligereza del sistema, haciendo posible que se integre en cualquier dispositivo sin necesidad de contar con demasiados recursos. Siendo útil para muchas situaciones en las cuales no es necesaria la integración de un microprocesador más pesado, ya que las tareas a llevar a cabo no requieren de estructuras complejas.

Como objetivo principal de este TFG se implementa una herramienta de depuración para controlar el funcionamiento del microprocesador una vez implementado físicamente. Esta herramienta se encuentra completamente desarrollada en el lenguaje de programación Python y permite el control de la ejecución del microprocesador, monitorizar el estado de este, y la carga y descarga de datos, tanto en las memorias del sistema como en los periféricos integrados en este.

Como conclusión, en el capítulo final de la memoria se determinarán posibles mejoras del trabajo así como ampliaciones para mejorar su rendimiento.

1.1. Objetivos

Los objetivos establecidos para este trabajo son los descritos a continuación:

- Realizar una búsqueda de información para documentarse sobre el estándar de buses Avalon Memory Mapped.
- Familiarizarse con el entorno de trabajo Paltform Designer y la interconexión de módulos Avalon.
- Generar los módulos Avalon MM Slave y Master que permitan la interconexión entre el microprocesador y las diferentes memorias pertenecientes al sistema.
- Modificar la implementación del microprocesador para adaptarlo al estándar de buses Avalon MM Slave y Master.
- Definir un módulo debug junto con otros módulos adicionales para asegurar el control de las memorias y periféricos del microprocesador en tiempo real.
- Desarrollo de un módulo independiente de UART (Transmisor-Receptor Asíncrono Universal) para la conexión física a través de un puerto RS232 entre el microprocesador y un sistema externo de control.
- La programación de una aplicación capaz de controlar el funcionamiento del microprocesador a través de una conexión serie USB - RS232.
- Validación del funcionamiento del conjunto del sistema a través de la verificación física en la placa de evaluación prevista.

1.2. Plan de trabajo

Para mantener un flujo de trabajo continuo y ordenado se ha seguido el siguiente diagrama de forma que todos los apartados propuestos son cubiertos con el tiempo necesario.

Figura 1.2: Diagrama de Gantt

Como se puede apreciar en la *figura 1.2*, el trabajo se ha desglosado en diferentes apartados para poder establecer una estructura de trabajo lineal, siempre teniendo en cuenta los objetivos de este y la limitación temporal de cada uno.

Capítulo 2

Marco teórico

A modo de introducción al marco teórico de los componentes principales del sistema desarrollado en el presente TFG, así como su comportamiento dentro del sistema implementado, se realizará un análisis de las diferentes características que conforman el microprocesador segmentado. Posteriormente se detallarán la cualidades que integra la adaptación del sistema al estándar de buses Avalon Memory Mapped.

Para comenzar, este microprocesador parte de una implementación *Single Cycle* la cual está basada en una arquitectura abierta conocida como RISC-V [4][5], de forma que utiliza un conjunto de instrucciones reducido conocido como RISC *(Reduced Instruction Set Computer)* cuyas características principales son: la longitud fija de las instrucciones con un número de formatos reducido, y un acceso a memorias limitado únicamente a las instrucciones de carga y almacenamiento de datos.

Siendo RISC-V un conjunto de instrucciones cuya finalidad principal es el modelado del comportamiento de los microprocesadores, siendo de libre uso y de código abierto no se necesita de licencia para realizar implementaciones con este set. Esto permite que se encuentre en continuo desarrollo por parte de la comunidad, y abre la posibilidad al desarrollo de esta arquitectura en todo tipo de plataformas, siendo así muy versátil [5].

En este caso particular el microprocesador parte de una implementación parcial del set de instrucciones RV32I, lo que implica que este contiene un banco de registros con 32 registros de 32 bits cada uno, junto con soporte aritmético de números enteros (I). Las instrucciones se almacenan en memoria en formato *little-endian* y entre estas podemos diferenciar 6 formatos (R, I, S, SB, U y UJ) con las codificaciones que observamos en la *figura 2.1*.

CORE INSTRUCTION FORMATS

<table>
<tr><th></th><th>31   27 26 25</th><th>24   20</th><th>19   15</th><th>14   12</th><th>11   7</th><th>6   0</th></tr>
<tr><td>R</td><td>funct7</td><td>rs2</td><td>rs1</td><td>funct3</td><td>rd</td><td>Opcode</td></tr>
<tr><td>I</td><td colspan="2">imm[11:0]</td><td>rs1</td><td>funct3</td><td>rd</td><td>Opcode</td></tr>
<tr><td>S</td><td>imm[11:5]</td><td>rs2</td><td>rs1</td><td>funct3</td><td>imm[4:0]</td><td>opcode</td></tr>
<tr><td>SB</td><td>imm[12|10:5]</td><td>rs2</td><td>rs1</td><td>funct3</td><td>imm[4:1|11]</td><td>opcode</td></tr>
<tr><td>U</td><td colspan="4">imm[31:12]</td><td>rd</td><td>opcode</td></tr>
<tr><td>UJ</td><td colspan="4">imm[20|10:1|11|19:12]</td><td>rd</td><td>opcode</td></tr>
</table>

Figura 2.1: Formatos de instrucciones

Teniendo en cuenta las diferentes codificaciones usadas en esta implementación, se disponen de las siguientes instrucciones implementadas:

Instrucción	Formato	Tipo	Operación	Descripción
add	R	Aritmético	+	R[rd] = R[rs1] + R[rs2]
sub	R	Aritmético	-	R[rd] = R[rs1] - R[rs2]
slt	R	Aritmético	(A <B)	R[rd] = (R[rs1] <R[rs2]) ? 1 : 0
sltu	R	Aritmético	(A <B)	R[rd] = (R[rs1] <R[rs2]) ? 1 : 0
sll	R	Aritmético	≪	R[rd] = R[rs1] ≪ R[rs2]
srl	R	Aritmético	≫	R[rd] = R[rs1] ≫ R[rs2]
or	R	Aritmético	|	R[rd] = R[rs1] | R[rs2]
and	R	Aritmético	&	R[rd] = R[rs1] & R[rs2]
xor	R	Aritmético	∧	R[rd] = R[rs1] ∧ R[rs2]
lw	I	Acceso a memoria	+	R[rd] = M[R[rs1]+imm](31:0)
addi	I	Aritmético	+	R[rd] = R[rs1] + imm
slli	I	Aritmético	≪	R[rd] = R[rs1] ≪ imm
srli	I	Aritmético	≫	R[rd] = R[rs1] ≫ imm
slti	I	Aritmético	(A <B)	R[rd] = (R[rs1] <imm]) ? 1 : 0
sltiu	I	Aritmético	(A <B)	R[rd] = (R[rs1] <imm) ? 1 : 0
xori	I	Aritmético	∧	R[rd] = R[rs1] ∧ imm
ori	I	Aritmético	|	R[rd] = R[rs1] | imm
andi	I	Aritmético	&	R[rd] = R[rs1] & imm
jalr	I	Control	+	R[rd] = PC+4; PC = R[rs1] + imm
sw	S	Acceso a memoria	+	M[R[rs1] + imm](31:0) = R[rs2](31:0)
beq	SB	Control	-	if(R[rs1] == R[rs2]); PC = PC +[imm,1'b0]
bne	SB	Control	==	if(R[rs1] != R[rs2]); PC = PC +[imm,1'b0]
blt	SB	Control	<	if(R[rs1] <R[rs2]); PC = PC +[imm,1'b0]
bge	SB	Control	>=	if(R[rs1] >= R[rs2]); PC = PC +[imm,1'b0]
bltu	SB	Control	<	if(R[rs1] <R[rs2]); PC = PC +[imm,1'b0]
bgeu	SB	Control	>=	if(R[rs1] >= R[rs2]); PC = PC +[imm,1'b0]
aiupc	U	Aritmético	+	R[rd] = R[rs1] + [imm, 12'b0]
lui	U	Aritmético	+	R[rd] = [32b'imm[31],imm,12'b0]
jal	UJ	Control	+	R[rd] = PC + 4; PC = PC + [imm,1'b0]

Tabla 2.1: Lista de instrucciones implementadas

Una vez establecidas las instrucciones implementadas cabe resaltar la estructura definida en el microprocesador *Single Cycle* para así posteriormente definir su versión segmentada, también conocida como arquitectura *pipeline*, la cual aporta un mayor rendimiento en comparación con su versión sin segmentar.

Esto es debido a que los procesos en la versión *Single Cycle* siguen un modelo de ejecución serie, es decir, se ejecuta cada vez una instrucción y hasta que esta no termina no empieza la siguiente. Sin embargo, añadiendo la característica *pipeline* se logra una pseudo paralelización del microprocesador, lo que se traduce en la capacidad de procesar diferentes instrucciones. Sin embargo cada una de estas tarda el mismo tiempo que en su versión *Single Cycle*, por lo que la mejoría

de rendimiento de la segmentación del microprocesador únicamente se aprecia en la ejecución de un conjunto de instrucciones, ya que en el momento que se procese la primera instrucción, el resto únicamente necesitará un ciclo de reloj para ser procesadas. Logrando así mejorar así la frecuencia de funcionamiento, ya que en cada ciclo de reloj se procesa una instrucción diferente (*figura 2.2*).

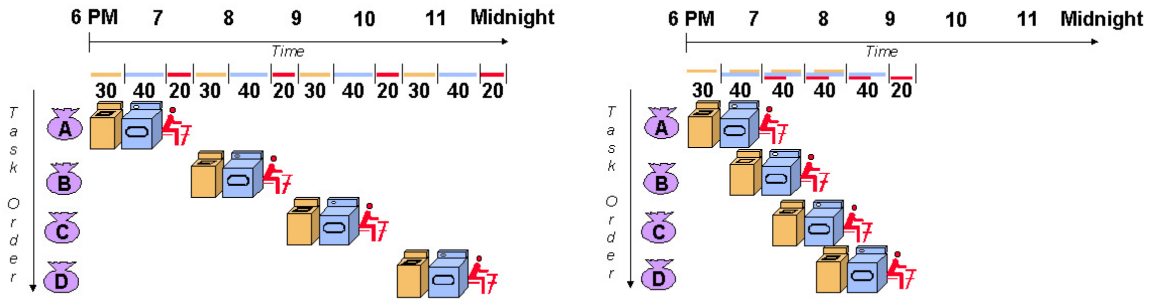

Figura 2.2: Analogía pipeline [6]

Sin embargo, este tipo de arquitectura lleva aparejado un aumento de la complejidad del sistema, debido a que en todo momento deben de considerarse tanto las instrucciones de salto como el uso de datos no que todavía no se ha almacenado. Para que el sistema siga funcionando como debe, una vez es segmentado se implementan dos módulos dedicados a gestionar los saltos de instrucción y el adelantamiento de datos entre las diferentes etapas; estos se denominan *Hazzard Detect Unit* y *Forward Unit* respectivamente [6].

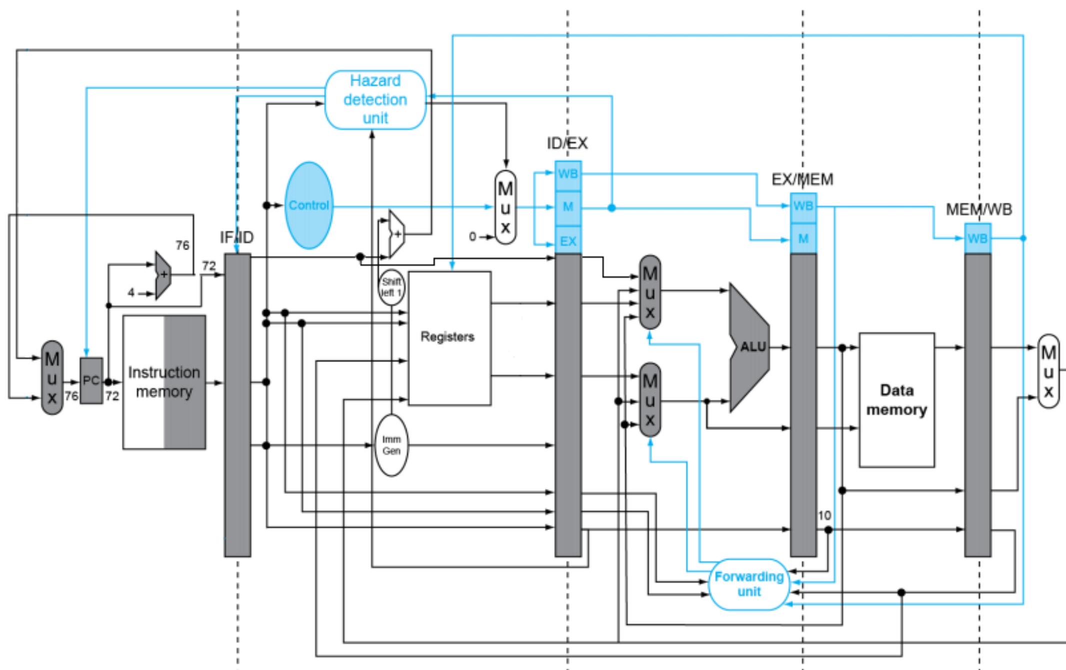

Figura 2.3: Esquema del microprocesador segmentado [7]

En la *figura 2.3* se puede observar la estructura adoptada así como los módulos anteriormente mencionados los cuales se detallarán a continuación.

En el caso del módulo *Hazzard Detect Unit* se evalúa la señal *Brach* de las distintas etapas, incluso del propio módulo de Control. Por lo que si entra en el *pipeline* una instrucción de salto, el módulo *Hazzard Detect Unit* detiene la carga de nuevas instrucciones hasta obtener la resolución sobre si se va a realizar el salto o no, por lo que detiene completamente la primera etapa y evalúa la evolución de la instrucción en las diferentes etapas. Si finalmente se realiza el salto se modificará el valor del PC según el salto establecido y se reactivará la carga de nuevas instrucciones desde la nueva dirección. En el caso de no realizar el salto se reanudara la carga de instrucciones únicamente incrementando el PC.

Por otro lado, el módulo *Forward Unit* determina a través de los registros usados en etapas anteriores si se ha de adelantar algún dato para realizar la operación pertinente en cada caso, es decir, si en la ejecución actual se pretende usar el valor de un registro actualizado en la ejecución previa, lo cual quiere decir que todavía no se ha actualizado en el banco de registros, se ha de adelantar el valor actualizado de este. Esto se realiza puenteando el valor obtenido sobre la ejecución previa hasta la entrada de la ALU, por lo que siempre se obtendrá el valor más actualizado sin depender de que este se encuentre ya almacenado en el banco de registros.

Para concluir este apartado se muestra un breve ejemplo del funcionamiento del módulo *Hazzard Detect Unit*, el cual se puede apreciar en la *figura 2.4*.

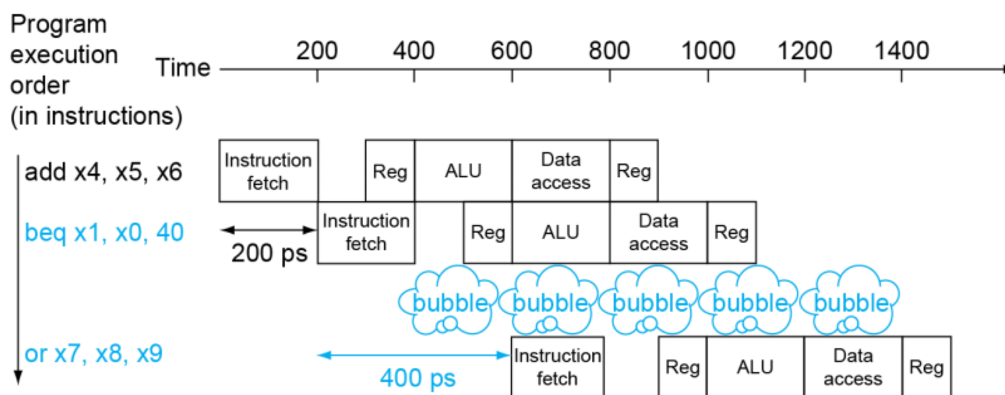

Figura 2.4: Ejemplo microprocesador segmentado [7]

Se puede apreciar como las instrucciones se evalúan por separado en cada etapa de segmentación, optimizando así el uso de la pipeline. Sin embargo cuando se carga una instrucción de salto se introducen ”bubble”, lo que significa que el microprocesador cesa la carga de nuevas instrucciones, en este caso hasta llegar a la etapa de la ALU, ya que en esta se decide si el salto se hará efectivo o no.

2.1. Avalon Memory Mapped

Como objetivo principal de este TFG se integran los módulos Avalon Memory Mapped [8] los cuales permiten la fácil interconexión entre los diferentes módulos integrados en el sistema, lo que hace posible la formación de un sistema SoC que integre distintos tipos de componentes en un solo sistema con una siemple implementación. Estos son generados siguiendo un estándar de conexión definido por Intel. Con ayuda de su herramienta **Platform Designer** se genera un sistema con todos los módulos interconectados entre ellos, únicamente con el uso de dos derivados del estándar Avalon Memory Mapped conocidos como **Slave** y **Master**, los cuales se desarrollarán en más profundidad en sus respectivas subsecciones.

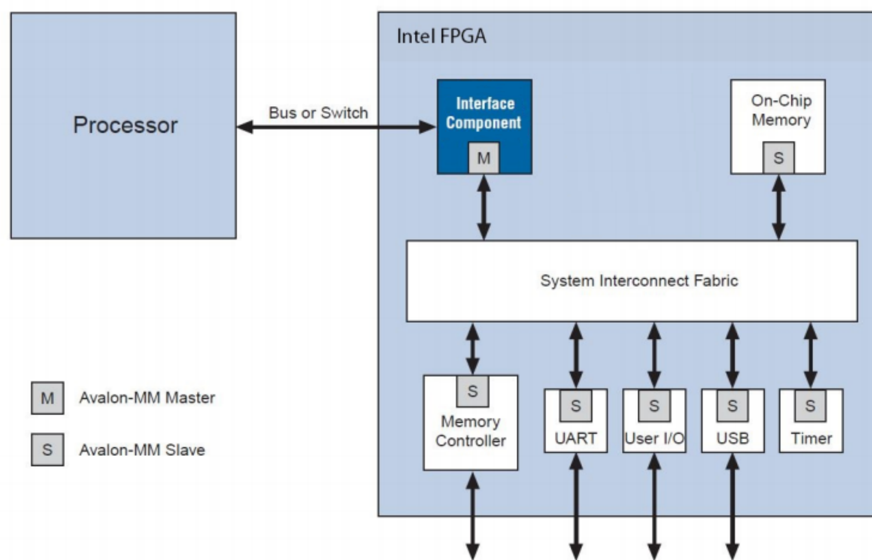

Figura 2.5: Ejemplo de un sistema Avalon [9]

Una de las principales características de los sistemas adaptados a un sistema de buses, es que el acceso a los periféricos se realiza mediante un mapeado en memoria, es decir, para acceder a estos componentes únicamente se ha de acceder mediante su dirección de memoria asignada. Esto simplifica mucho la integración de los periféricos, así como su acceso al microprocesador, necesitando únicamente una interfaz para realizar todas las operaciones necesarias.

2.1.1. Funcionamiento del módulo Avalon Memory Mapped Slave

El módulo **Avalon Memory Mapped Slave**, es usado generalmente en los sistemas que responden a peticiones de otros sistemas, es decir que dependen en cierto grado de otro sistema, en este caso uno que integre una interfaz *Master*.

En este trabajo se ha decidido que el diseño del módulo *Slave* contará con 4 registros internos de 32 bits. Estos registros sirven a modo de interconexión entre el bus que interconecta los diferentes sistemas entre ellos y el propio periférico. Por lo que para modificar cualquier parámetro interno

del periférico se han de modificar los registros internos de su *interface Slave*.

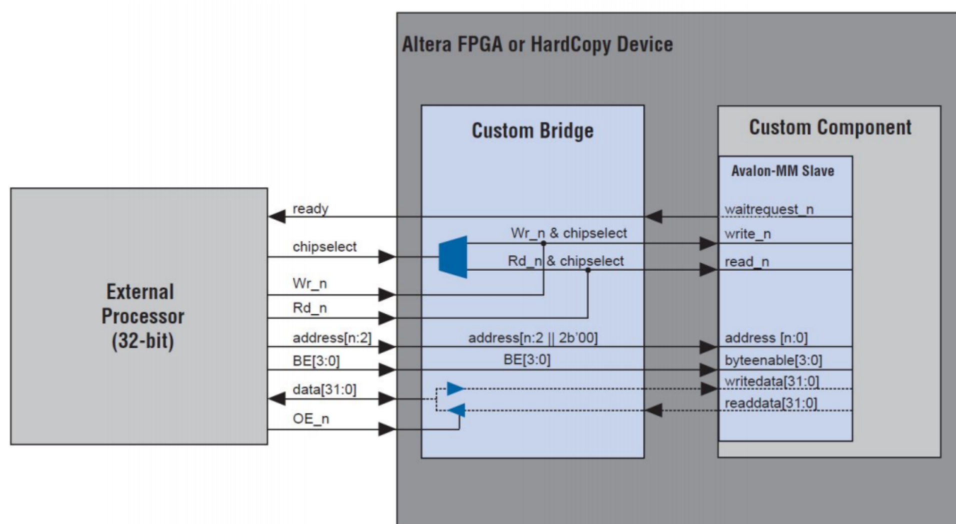

Figura 2.6: Estructura interface Slave [10]

En la *figura 2.6* se pueden apreciar las señales utilizadas comúnmente en la *interface Slave*, las cuales se detallarán a continuación:

Señal	nº bits	Tipo	Conexión	Descripción
waitrequest	1	output	Externa	Indica que el periférico va a estar ocupado bloquea su acceso
write	1	input	Externa	Determina si se va a realizar una operación de escritura
read	1	input	Externa	Determina si se va a realizar una operación de lectura
address	3	input	Externa	Dirección codificada para seleccionar un registro interno
writedata	32	output	Externa	Datos enviados desde los registros internos
readdata	32	input	Externa	Datos a escribir en el registro interno seleccionado
reg[0,1,2]	32	output	Interna	Registros internos que se interconectan con el periférico [11]
data	32	input	Interna	Datos a escribir por el periférico en el registro de solo lectura
we	1	input	Interna	Señal de control que habilita la escritura en el registro 3

Tabla 2.2: Lista de señales integradas en la *interface Slave*

Con estas señales es posible integrar todo tipo de periféricos, ya que a través de los registros (reg0, reg1, reg2) se pueden establecer los parámetros del periférico adaptado, junto con el registro reg3 el cual servirá para almacenar los datos del periférico cuando sea necesario y así poder enviarlos a cualquier sistema que los solicite. Un ejemplo de su funcionamiento es el mostrado en la *figura 2.7*, donde se adaptan diversos decodificadores para controlar los módulos 7 segmentos de la placa de evaluación usando el estándar Avalon MM Slave.

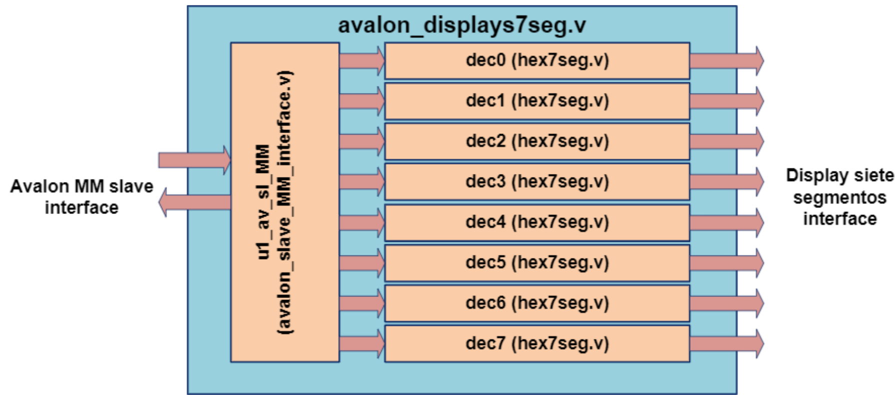

Figura 2.7: Ejemplo periférico Avalon MM Slave [11]

2.1.2. Funcionamiento del módulo Avalon Memory Mapped Master

El módulo **Avalon Memory Mapped Master** es usado en los sistemas que gestionan las transferencias dentro del bus, siendo estos los encargados de gestionar los datos de los sistemas que cuentan con una implementación en modo *Slave*.

En este caso, la *interface Master* controla los accesos de escritura y lectura sobre los sistemas con *interface Slave*. Un ejemplo del funcionamiento de esta es visto en la *figura 2.8*.

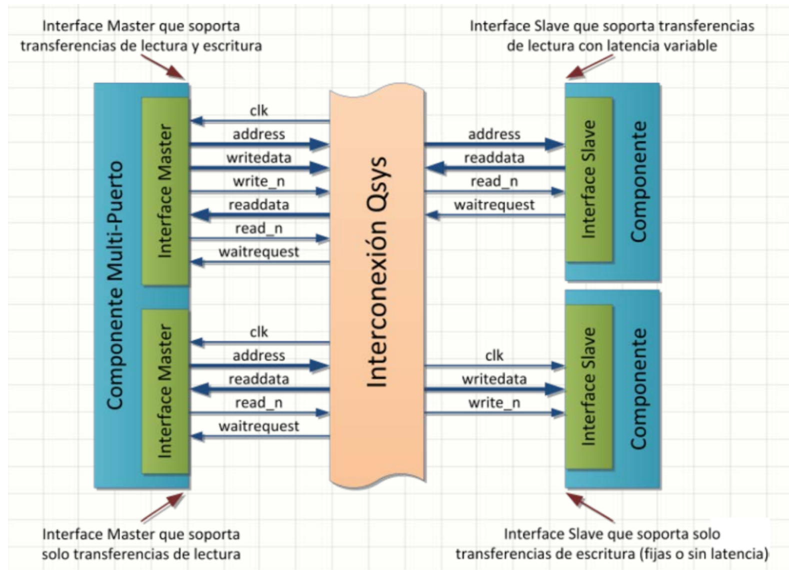

Figura 2.8: Ejemplo conexión entre diferentes buses [12][11]

Donde se aprecian dos tipos de *interface Master* una de ellas únicamente con soporte de transferencias de lectura, y como versión complementaria a esta soporte a transferencias tanto de lectura como de escritura. En este trabajo únicamente nos centraremos en la interface con un uso más ge-

neralizable, como es la versión con soporte de transferencias tanto de lectura como de escritura.

Una de las cualidades principales de este sistema de buses es que únicamente disponiendo de la dirección del sistema con el que se quiere interactuar y del tipo de operación a realizar se puede establecer una conexión entre estos. El encargado de gestionar las direcciones internas es el propio bus de conexiones, denominado en la *figura 2.8* como QSYS, recibiendo la dirección del dispositivo con el que la *interface Master* se quiere comunicar y habilitando la señal *chipselect* de este para interactuar con él. Todo esto se realiza en paralelo junto con la activación de la señal de lectura o escritura, así como con la introducción de los datos en el caso de la escritura, y la lectura de estos si es el caso de la lectura.

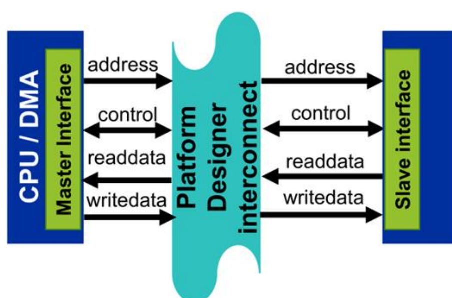

Figura 2.9: Ejemplo conexión *interface Master* [12]

En la *figura 2.9* muestra de forma simplificada el funcionamiento de un sistema con una única *interface Master* y una *interface Slave*, de forma que en el momento que la CPU de la *figura 2.9* requiera de la escritura en el periférico, esta utilizará el rango de direcciones de memoria asignado para el sistema Slave. Variando las direcciones en múltiplos de 4 accederá a los registros internos de la interfaz, pudiendo tanto leer como escribir en ellos, y generando así una completa interacción entre el maestro y el esclavo de este sistema.

Para finalizar este apartado se detallarán las señales del módulo Avalon Memory Mapped Master:

Señal	nº bits	Tipo	Conexión	Descripción
waitrequest	1	input	Externa	Indica que la interfaz va a estar ocupada - bloquea su acceso
write	1	output	Externa	Determina si se va a realizar una operación de escritura
read	1	output	Externa	Determina si se va a realizar una operación de lectura
address	32	output	Externa	Dirección para seleccionar el periférico a usar
writedata	32	output	Externa	Datos enviados hacia los periféricos
readdata	32	input	Externa	Datos a recibir de los periféricos
lock	1	output	Externa	Determina si la interface está ocupada

Tabla 2.3: Lista de señales integradas en la *interface Master* I

Señal	nº bits	Tipo	Conexión	Descripción
dataToWrite	32	input	Interna	Datos a escribir al periférico
addressToAcces	32	input	Interna	Dirección del periférico sin codificar
rnw	1	input	Interna	Señal de control de lectura (1) y de escritura (0)
start	1	input	Interna	Determina el inicio de la transmisión de datos
done	1	output	Interna	Señal que determina el final de una transmisión
dataRead	32	output	Interna	Datos recibidos de la transmisión realizada

Tabla 2.4: Lista de señales integradas en la *interface Master* II

Debido a la interconexión de las interfaces *Slave* y *Master* a través del bus del sistema, se aprecia cierta similitud en las señales que conforman ambas interfaces. Esto es debido a que sus señales se interconectan mediante el bus, realizando así las transferencias requeridas.

Capítulo 3

Desarrollo práctico

Una vez detallado el funcionamiento tanto del microprocesador como de los diferentes módulos Avalon Memory Mapped, en este apartado se introducirá el desarrollo llevado a cabo para la integración del sistema de buses Avalon MM Master en el microprocesador segmentado, junto con la creación periféricos útiles para la monitorización del sistema.

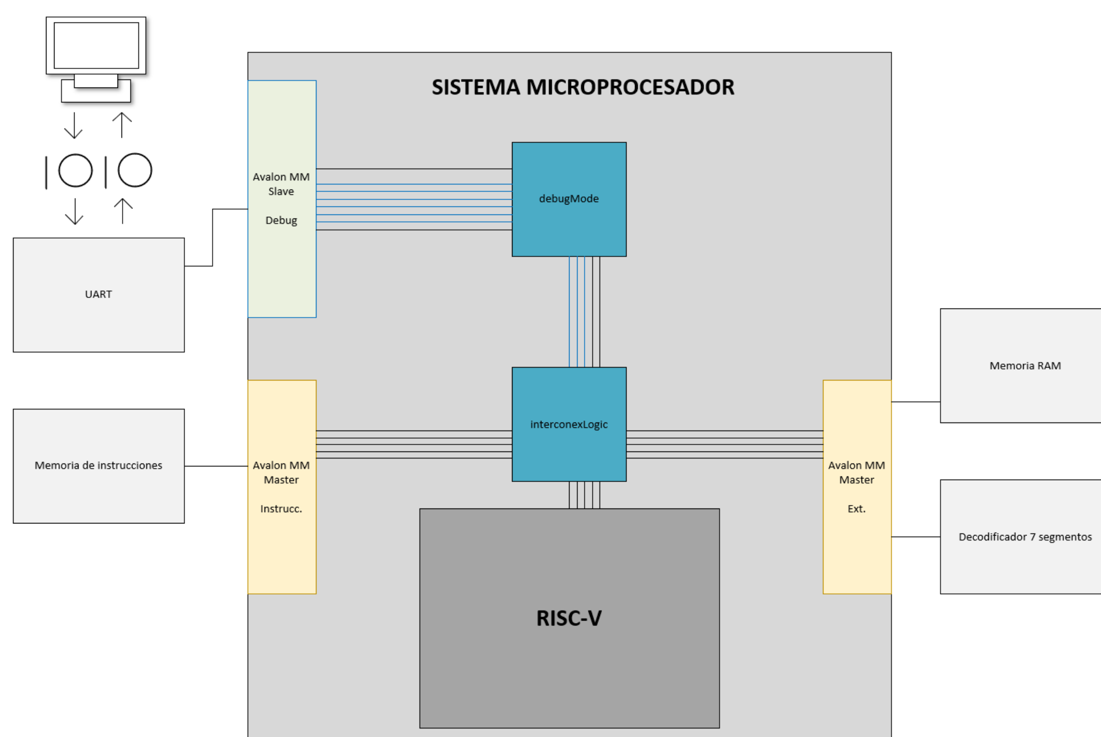

Figura 3.1: Esquema general del sistema a implementar

Como se puede ver en la *figura 3.1*, el trabajo se compone de un sistema central y diferentes sistemas que actúan como periféricos de este. A continuación se describirán los sistemas que

conformarán este trabajo.

- **El microprocesador segmentado RISC-V**, el cual forma el núcleo principal del sistema y es el encargado de procesar la información proveniente del exterior.
- Las **interfaces Avalon Memory Mapped**, que mejoran la versatilidad del sistema permitiendo adaptarse a cualquier tipo de módulo externo sin necesidad de realizar cambios en las estructuras que conforman el sistema principal. Su implementación se llevará a cabo a lo largo de los capítulos 4, 5 y 6.
- El **sistema de debug** integrado dentro del sistema del microprocesador, que permitirá modificar el comportamiento del microprocesador y acceder tanto a los dispositivos externos como a las memorias de instrucciones. Estos módulos se desarrollan en profundidad en el capítulo 5 de la memoria.
- Para complementar el sistema de debug integrado en el sistema, se desarrolla un **periférico de control UART**. Este será responsable mediante el uso del bus Avalon de transmitir y recibir datos desde el exterior, infiriendo en el sistema de debug del sistema del microprocesador, pudiendo así controlar su funcionamiento desde un dispositivo externo. El periférico se ha descrito en el capítulo 6 de la memoria.
- Para completar tanto el sistema de debug como el periférico UART, se ha desarrollado una **aplicación de control** en el lenguaje Python. Esta aplicación permite una total explotación de las funcionalidades integradas en el sistema de debug, aprovechando el periférico UART para establecer las conexiones entere el sistema y la aplicación a través del estándar de comunicaciones RS232 (desarrollada en el capítulo 6).
- Diferentes **periféricos** externos al sistema del microprocesador, que son conectados a este mediante el bus Avalon, permitiendo el uso de periféricos descritos por terceros sin realizar modificaciones en la descripción del hardware usado.

Para llevar a cabo estos objetivos se ha requerido del uso del siguiente software:

- **Quartus Prime 17.1** como compilador de código System Verilog junto con sus herramientas Platform Designer y Signal Tab Logic Analyzer para la desarrollar el sistema.
- **ModelSim STARTER EDITION 10.5b** para realizar las simulaciones del código generado y del conjunto del sistema.
- **Pycharm Community Edition** como compilador de **Python 3.8.8** para la programación de la aplicación de depuración del microprocesador.
- **GitHub Desktop** como gestor de versiones on-line para controlar los cambios realizados a lo largo del desarrollo y en paralelo obtener copias de estos.
- **Notepad++** como editor de texto para realizar modificaciones puntuales en el código.

Capítulo 4

Interfaz Avalon MM Master en RISC-V

Como uno de los objetivos principales a cubrir en este TFG, en este apartado se cubrirá la integración del microprocesador segmentado al estándar de buses Avalon Memory Mapped Master.

Uno de los principales cambios respecto a la implementación previa del microprocesador es que se sustituyen las memorias generadas mediante código Verilog por interfaces, de esta forma el microprocesador no tendrá acceso directo a periféricos externos a él. En primera instancia, mediante la integración de las interfaces en el sistema del microprocesador, desaparece la conexión directa entre las memorias, por lo que se pierde la capacidad de lectura asíncrona de estas. Sin embargo, una vez implementadas las interfaces aparece una nueva variable al sistema, como lo es el tiempo de espera para realizar transmisiones de lectura y escritura en los dispositivos externos al microprocesador.

Todas estas desventajas se ven eclipsadas debido a la gran versatilidad que gana el sistema, ya que como se ha explicado en la sección 2.1 de este documento el sistema acepta el uso de diferentes tipos de componentes; esto es debido a la generalidad de las interfaces implementadas. En este caso se han implementado interfaces del tipo Master, ya que estas proporcionan la capacidad de establecer conexiones con los periféricos externos al sistema, interviniendo así en su comportamiento.

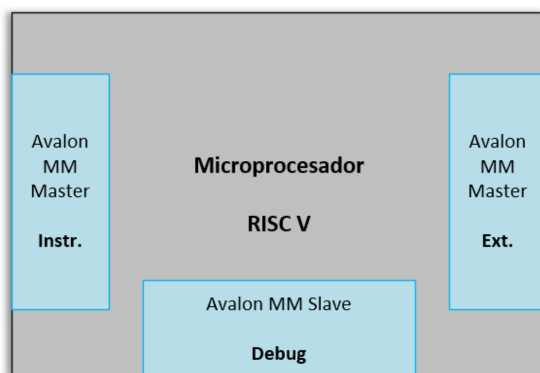

Figura 4.1: Esquema microprocesador

En la *figura 4.1* se observa a groso modo el esquema del sistema del microprocesador, integrando en este tres interfaces, las cuales se dividen en dos interfaces del tipo Avalon MM Master, para el acceso a periféricos; y una interfaz Avalon MM Slave, que se encarga de establecer la comunicación con el periférico UART. En este apartado nos centraremos en las interfaces tipo Avalon Memory Mapped Master.

- **Avalon MM Master de instrucciones:** A través de esta interfaz se gestionan los accesos a memoria de instrucciones del microprocesador, es decir, el bus es capaz de gestionar las diferentes direcciones dadas por el PC (Program Counter) en cada ciclo de ejecución del sistema, de forma que se accedan a los datos que se encuentran en el dispositivo establecido como memoria de instrucciones.

  De la misma forma, se permite a los diferentes módulos encargados del control de la sincronización de las diferentes partes del sistema que realicen un control del funcionamiento del PC, todo esto en función de los tiempos de espera requeridos por el dispositivo usado a modo de memoria de instrucciones. Pudiendo conectar cualquier tipo de dispositivo normalizado en el estándar del bus Avalon, y el microprocesador no debe apreciar ninguna diferencia significativa en su funcionamiento.

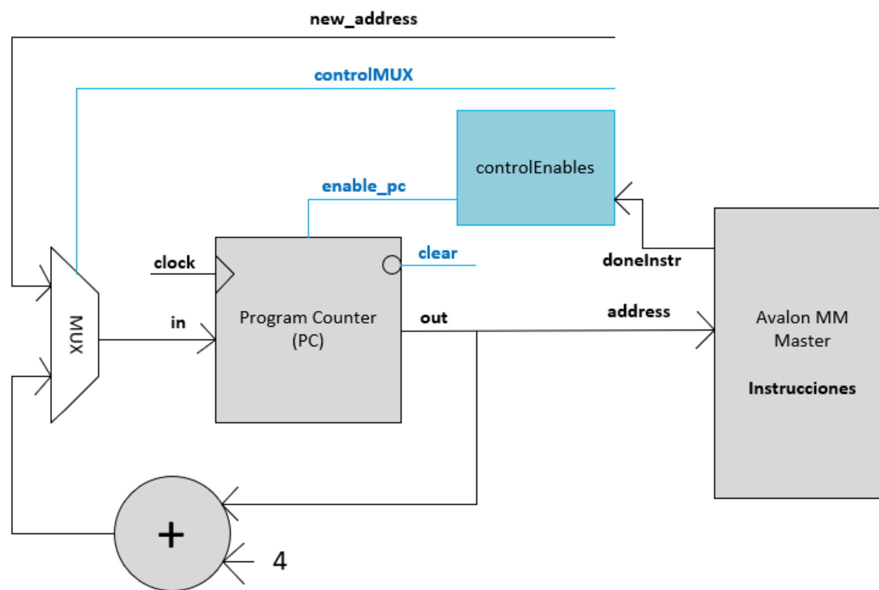

Figura 4.2: Módulo de control del Program Counter (PC)

En la *figura 4.2* se observa que el módulo de *controlEnables* funciona mediante el uso de la señal de control **doneInstr**, la cual determina cuando una operación de escritura o lectura se ha completado con éxito. Este módulo controla la ejecución del sistema mediante el control de las señales de habilitación, tanto de las etapas de segmentación del microprocesador, como del propio **PC**. Esto se traduce en que independientemente del dispositivo al que se esté accediendo el funcionamiento del microprocesador no sufrirá errores, ya que este no depende directamente de las especificaciones del dispositivo usado, por lo que el bus Avalon junto

con el módulo de *controlEnables* son los encargados de sincronizar el microprocesador con los elementos externos a él.

- **Avalon MM Master de memoria de datos:** De la misma forma que en el caso de la interfaz de datos de instrucciones, se ha utilizado una interfaz del tipo Master, de forma que se obtendrá el control por parte del microprocesador hacia los periféricos conectados como dispositivos externos. Esta interfaz seguirá el mismo comportamiento que la interfaz de datos de instrucciones, sin embargo, en el caso de la interfaz de instrucciones únicamente se permiten transmisiones de lectura por parte del microprocesador. Permitiendo en el caso de la interfaz de memoria de datos, tanto transmisiones de lectura como de escritura por parte del microprocesador. Esto se llevará a cabo siempre que se utilicen las instrucciones **sw** o **lw** integradas en el juego de instrucciones (*tabla 2.1*).

  De esta forma el esquema general del conjunto de componentes compuesto por el microprocesador segmentado y las diferentes interfaces mencionadas formaría el sistema de la *figura 4.3*.

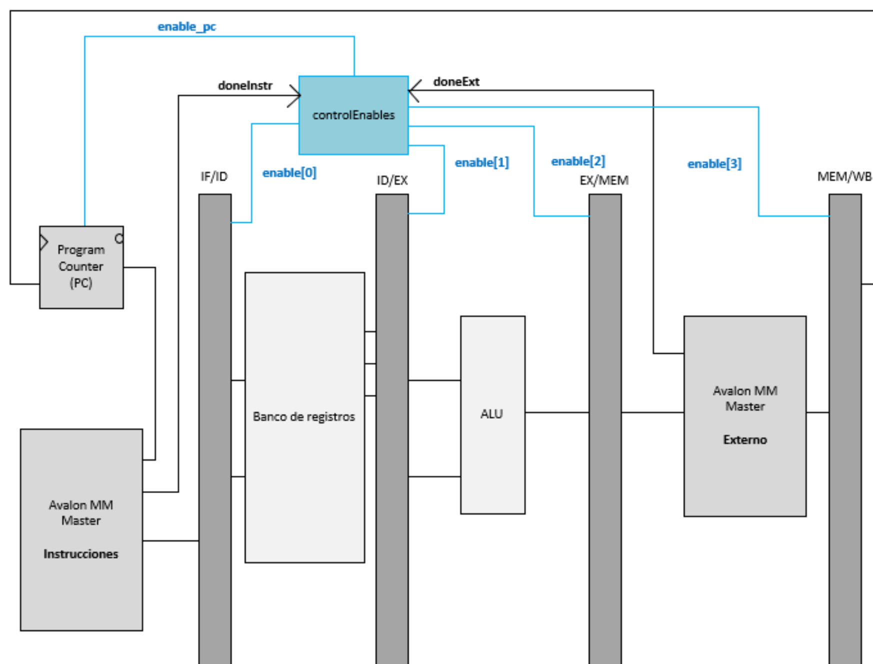

Figura 4.3: Esquema del microprocesador integrando los buses Avalon

Pudiendo apreciar a groso modo el funcionamiento del módulo descrito anteriormente, junto con la posición de los módulos que lo conforman dentro del esquema de segmentación del microprocesador.

En el caso de la interfaz de datos externos se encuentra en la penúltima etapa del sistema, por lo que no se encuentra sincronizada con la interfaz de instrucciones. Debido a esto el módulo de *controlEnables* debe de tener en en cuenta que pueden realizarse lecturas tanto

de instrucciones como de datos externos al mismo tiempo. Además el módulo de *controlEnables* ha de contemplar que los tiempos de espera para cada una de estas operaciones pueden ser diferentes, por lo que hasta que la última de estas operaciones no se complete no se reanudara la ejecución del microprocesador.

Ambos módulos comentados comparten el mismo funcionamiento, ya que únicamente se ha desarrollado un módulo Avalon MM Master en todo el proyecto y se han realizado diferentes instancias de este, de forma que es un módulo genérico con posibilidad de integración en cualquier tipo de dispositivo. El funcionamiento de este módulo integrado en el sistema es muy importante, ya que junto con el bus de interconexión Avalon actúan de árbitros para los datos; determinando el destino de estos y direccionándolos a través del bus hacia el dispositivo asociado a la dirección establecida.

Estas direcciones se establecen mediante la herramienta integrada en Quartus denominada **Pltaform Designer**, creando así un mapa de memoria donde se fijan los rangos de direcciones que ocupa cada dispositivo dentro del sistema. Disponiendo de un mapa de memoria independiente para cada interfaz, ya que cada una de estas conforma un sistema independiente a las demás.

4.1. Integración a Platform Designer

Para crear un componente en la herramienta Platform Desginer primero se han de instanciar las interfaces requeridas en el código Verilog que modela el comportamiento de este. En este caso las interfaces Master de instrucciones y de memoria de datos, se han instanciado en el módulo *top* del sistema, permitiendo así que todos los componentes encargados del control de las memorias tengan acceso a estas.

Figura 4.4: Señales Platform Designer

En la *figura 4.4* se muestra la creación de un nuevo sistema mediante la herramienta de Platform Designer [13], el cual se forma a partir de las señales pertenecientes a las interfaces que componen el sistema del microprocesador. En concreto la pestaña mostrada en la *figura 4.4* define el comportamiento de las señales que atacan directamente al bus Avalon.

En el caso del microprocesador segmentado, contiene las tres interfaces mencionadas anteriormente, dos de clase maestro y una de clase esclavo (para el módulo debug el cual se tratará posteriormente). En cada uno de estos se aprecian las señales descritas anteriormente en la *tabla 2.3* definiendo así su comportamiento de cara al bus que interconecta los diferentes componentes del sistema. Se puede observar en la *figura 4.4* como ambas interfaces Master comparten el mismo tipo de señales y sus comportamientos, reforzando así la generalidad de la implementación del módulo Avalon MM Master junto con la versatilidad que ofrece. Pudiendo fácilmente implementar en cualquier periférico o sistema un módulo Avalon MM e integrarlo al sistema sin ningún requerimiento ni modificación adicional.

Una vez integradas las interfaces Avalon MM Master en el sistema, la propia herramienta Platform Designer ofrece dos esquemas de temporización para entender el comportamiento temporal de la interfaz implementada sin necesidad de simulaciones externas. En este caso obtenemos los esquemas esperados tras diseñar el sistema.

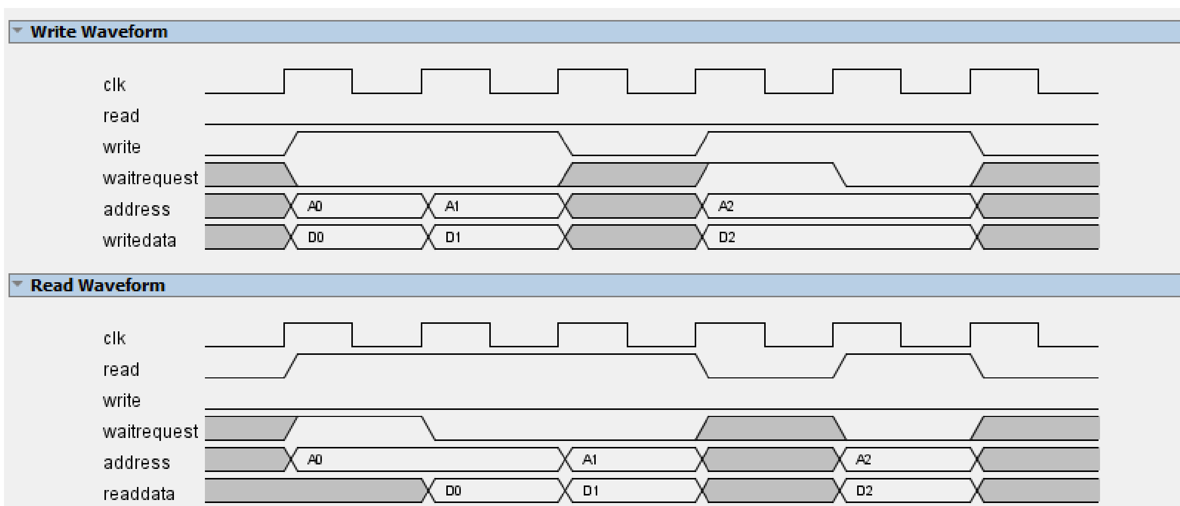

Figura 4.5: Temporización del módulo Avalon MM Master

Como se observa en la *figura 4.5*, únicamente cuando se activan las señales de lectura o escritura empieza la transmisión de datos por parte del componente con interfaz de clase Slave. Esto difiere dependiendo del componente, ya que algunos requieren más de un ciclo para completar las operaciones de lectura o escritura, por lo que para esto existe la señal *waitrequest* anteriormente mencionada. Esta señal ayuda a saber cuando los datos solicitados se encuentran disponibles tanto en el bus para leerlos, como en el dispositivo para escribirlos.

A modo de resumen de la implementación realizada en el sistema del microprocesador, se puede analizar la estructura obtenida a través de una de las ventanas disponibles en Platform Designer (mostrado en la *figura 4.6*).

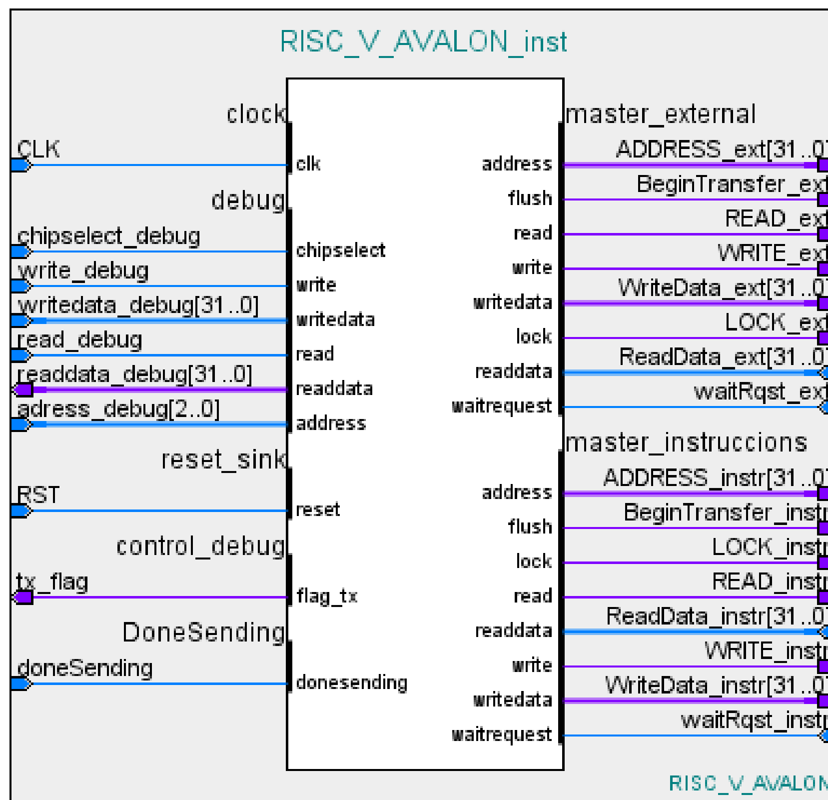

Figura 4.6: Módulo microprocesador en Platorm Designer

Donde se puede diferenciar entre las diversas interfaces implementadas y las señales pertenecientes a estas. Vislumbrando además diversas señales de tipo *Conduit*, las cuales son conexiones directas de datos entre varios sistemas sin necesidad de atravesar el bus ni establecer previamente una conexión entre los diferentes sistemas, por lo que en este tipo de conexiones no influyen para nada las interfaces Avalon MM. Este tipo de conexiones se utiliza generalmente para señales que interactúan con el medio exterior o para señales de control, ya que no requieren de una interfaz para esto y son más versátiles de forma independiente.

Para finalizar, una vez asignadas todas las señales, se genera el sistema que forma el microprocesador adaptado a las interfaces Avalon MM Master.

Para comprobar el funcionamiento del componente generado, se instancia el sistema generado RISC-V con diferentes componentes pertenecientes a librerías de **Platform Designer** con el fin de simular su comportamiento teniendo en cuenta los cambios acarreados por la integración de las interfaces Avalon.

En este caso se ha decidido complementar el sistema diseñado con dos memorias del tipo *On-Chip Memory (RAM or ROM)*, de forma que se comportan como memoria de instrucciones y memoria de datos. Ambas cuentan con interfaces Slave, por lo que se pueden conectar con el microprocesador mediante sus interfaces Master sin ningún problema (se aprecia en la *figura 4.7*).

El sistema se completa con un módulo de comunicación serie desarrollado para poder establecer una transmisión de datos entre el sistema del microprocesador y un elemento externo. Este debe ser capaz de aprovechar las funcionalidades que ofrece el módulo debug integrado en el microprocesador; este módulo se describirá con más detalle en el capítulo 6.

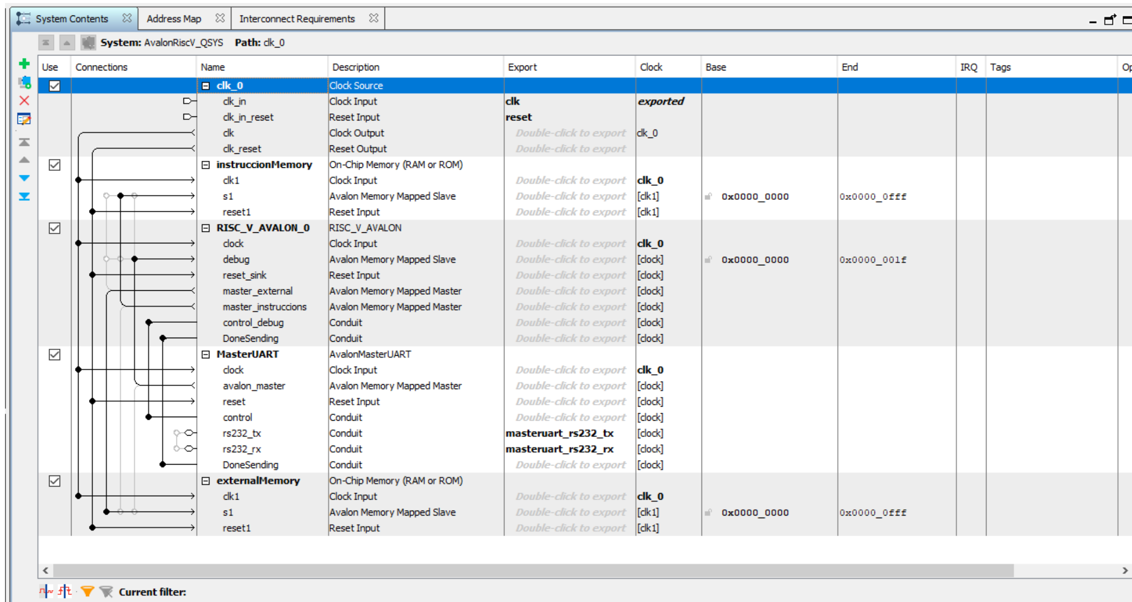

Figura 4.7: Sistema completo Platform Designer

Para simular el sistema de la *figura 4.7* se genera un banco de pruebas o ”Testbech”, este se genera mediante el uso de un menú integrado en Platform Designer. Usando la herramienta situada en ***Generate/Generate Testbench System*** se genera una carpeta en el directorio del proyecto con los archivos preparados para ejecutarse en el software **ModelSim** y así simular el conjunto del sistema. Sin embargo, se ha de establecer de forma manual los estímulos deseados para simular el sistema, ya que por defecto no son generados automáticamente. Todas las simulaciones se detallarán en el capítulo 7 de esta memoria.

Capítulo 5

Módulo de Debug

Para poder realizar una ejecución controlada del microprocesador y de sus componentes externos se ha desarrollado un módulo *debug*. En el presente capítulo se detallará su funcionamiento.

Este módulo tiene como principal finalidad el control del sistema mediante el uso de diferentes comandos provenientes del sistema externo UART (Transmisor-Receptor Asíncrono Universal), y su gestión a través de la interfaz Avlon MM Slave integrada en el microprocesador.

Todo esto resulta en la generación de un sistema compuesto principalmente por el microprocesador segmentado basado en la aquitectura RISC-V, los módulos pertenecientes al sistema de *debug* (**debugMode** e **interconexLogic**), y las diferentes interfaces que complementan estos dos sistemas haciendo así que convivan como uno solo.

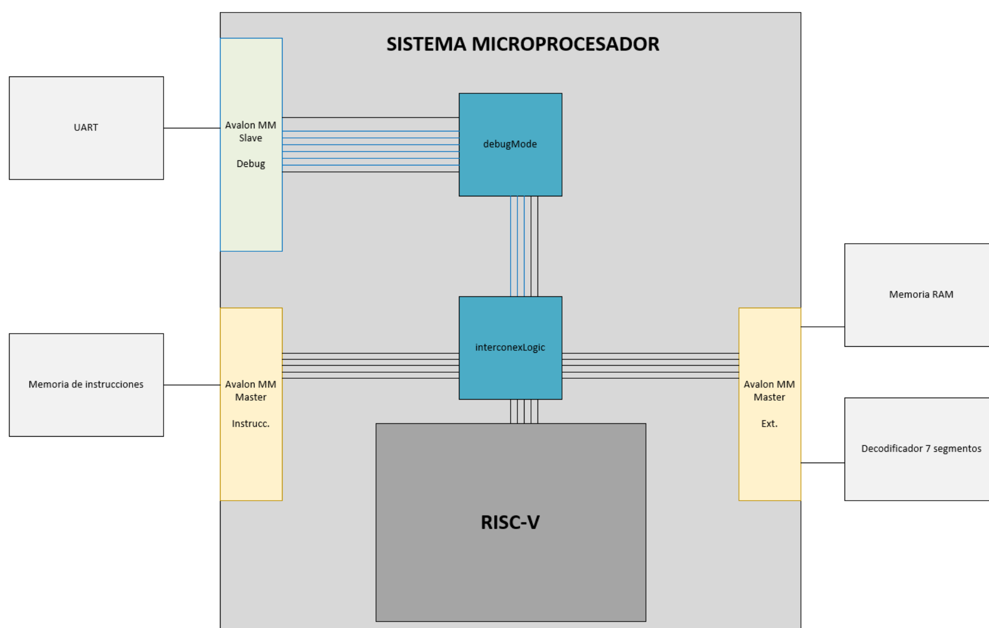

Figura 5.1: Esquema general del sistema final

El elemento principal en el sistema debug es la propia interfaz Slave, ya que en esta se almacenan los datos que posteriormente se procesan en el módulo **debugMode**. Asimismo este modifica el comportamiento del módulo **inteconexLogic** realizando un *bypass* entre las interfaces de datos del sistema, el microprocesador y el sistema debug, todo esto dependiendo de la instrucción de control recibida.

Es importante remarcar que la interfaz Slave del sistema cuenta con 3 registros de libre lectura y escritura, y un registro de solo lectura, de forma que se ha basado el comportamiento del **debugMode** en función de estas características.

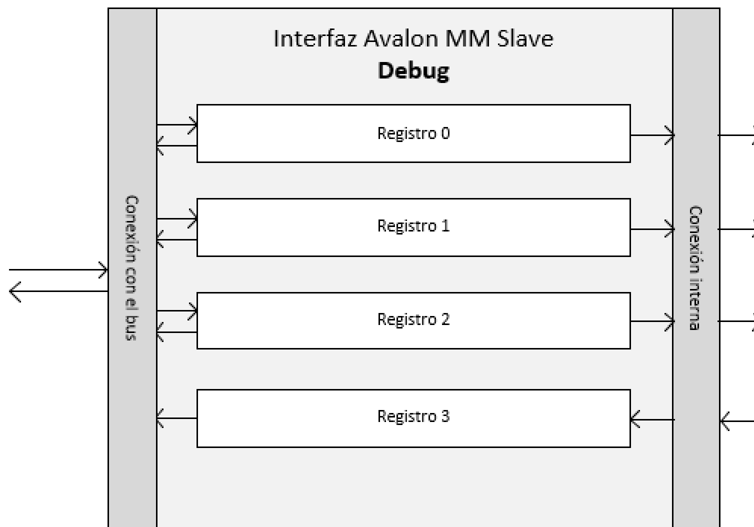

Figura 5.2: Esquema de bloques interfaz Avalon MM Slave debug

Para aprovechar al máximo sus características se ha optado por usar sus registros internos vistos en la *figura 5.2* de la siguiente forma:

- El **registro 0** se ha utilizado como contenedor para las instrucciones que se encargan de determinar el comportamiento del sistema, es decir, se ha elegido una determinada distribución de bits para realizar diferentes operaciones en el sistema desde el exterior y se ha almacenado la instrucción que contiene estos bits en este registro. La estructura diseñada para las instrucciones es la siguiente (*figura 5.3*).

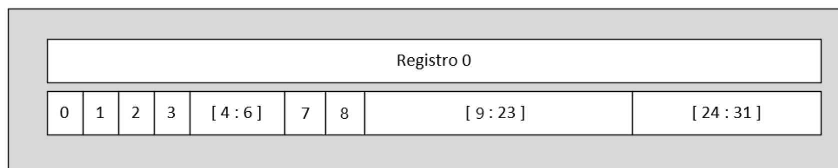

Figura 5.3: Distribución de bits del registro 0

Representando así cada bit o conjunto de estos una operación diferente dentro del sistema, pudiendo así interactuar con este a tiempo real.

- **Bit 0:** Cuando el bit se encuentra a nivel alto (su valor es 1) el sistema entra en modo debug, lo que ocasionará la parada de la ejecución libre del microprocesador y su entrada en un estado de espera, que se complementa con las posibles operaciones que describen el resto de bits que conforman esta instrucción.
- **Bit 1:** Bit reservado para posibles ampliaciones del sistema de debug para integrar nuevas funcionalidades.
- **Bit 2:** Cuando este bit se encuentra activo el microprocesador entra en estado de espera. Dependiendo del valor del bit 3, si se encuentra activo se realizará una función determinada o si no lo está únicamente mantendrá parada la ejecución del sistema. Sin embargo, cuando se encuentra desactivada el sistema se encuentra parado para poder realizar operaciones de lectura o escritura a través de las interfaces Avalon MM Master y el módulo **interconexLogic**.

  Dependiendo de los valores de los bits 4:6 del registro, el módulo **interconexLogic** modificará las conexiones entre las diferentes interfaces, el sistema de debug y el microprocesador, de forma que permitirá realizar diferentes operaciones a través del conjunto del sistema (vistas en la tabla 5.1).
- **Bit 3:** Cuando este bit se encuentra activo se habilita la ejecución por pasos del sistema. Esta permite parar la ejecución del sistema pudiendo ejecutar un número de ciclos de reloj determinados mediante los bits 24:31. Cuando se encuentra desactivada, únicamente mantiene el sistema parado (siempre que el bit anterior se encuentre activo).
- **Bits 4 al 6:** Este conjunto de bits proporcionan la identificación de la operación a llevar a cabo por el módulo **interconexLogic**, las combinaciones disponibles son las proporcionadas en la *tabla 5.1*.

Bits	Funcionamiento
000	Se realiza una conexión entre las interfaces de datos y el microprocesador.
001	Bypass entre la interfaz de datos externos y el sistema de debug para leer datos de elementos externos.
010	Bypass entre la interfaz de datos de instrucciones y el sistema de debug para leer instrucciones.
011	Bypass entre la interfaz de datos externos y el sistema de debug para escribir datos en elementos externos.
100	Bypass entre la interfaz de datos de instrucciones y el sistema de debug para escribir instrucciones.
101	Envío del valor del módulo PC a través del módulo debug.
110	Envío del valor a la entrada del módulo PC a través del módulo debug.
111	Reservado para posibles ampliaciones.

Tabla 5.1: Señales de control del módulo interconexLogic

- **Bit 7:** Este bit se encuentra reservado para posibles ampliaciones en el sistema de debug, más precisamente para añadir posibles nuevas funcionalidades relacionadas con el módulo **interconexLogic**.

- **Bit 8:** Determina a nivel alto si se realiza una activación de la señal *clear* a través de todo el microprocesador, limpiando los valores almacenados tanto en las etapas de segmentación como en el **PC**.
- **Bits 9 al 23:** Reservados para ampliaciones futuras u otro tipo de de implementación que requiera un alto número de bits de control.
- **Bits 24 al 31:** Número de interacciones a ejecutar en el modo ejecución por pasos. Se encuentra limitado por el número de bits establecido, sin embargo es fácilmente ampliable si se diera la necesidad.

- El **registro 1** determina la dirección a acceder cuando se realiza cualquier operación que haga uso de las interfaces de acceso de memoria, tanto de instrucciones como de datos. Ya que es necesaria la dirección a acceder para poder realizar cualquier tipo de transacción.

  Esta dirección utilizara el módulo **interconexLogic** para ser direccionada a la interfaz deseada, ya que el módulo tiene como única función direccionar los datos y direcciones a los sistemas que lo requieran. En el caso del módulo **debugMode**, se encarga de controlar el módulo **interconexLogic** dependiendo de la instrucción de control recibida en cada momento.

- El **registro 2** almacena los datos a escribir tanto en la memoria de instrucciones como en la memoria de datos, ya que estos se direccionan de la misma forma que las direcciones.

  En el caso de realizar operaciones de escritura se direccionará este registro hacia las interfaces junto con el registro 1, de forma que se escriban los datos contenidos en este registro a la dirección contenida en el registro 1. Usando el módulo **interconexLogic** para direccionar estos registros hacia las señales *dataToWrite* y *addressToAcces* respectivamente.

- En el caso del **registro 3**, al ser únicamente de lectura se utiliza para enviar datos desde el sistema hacia el exterior. Es decir, internamente se almacenan los datos recibidos tanto de la posible operación de lectura realizada desde cualquier interfaz, como la lectura del valor del PC actual o el siguiente.

  En él se almacenan los resultados obtenidos, de forma que cuando estos se encuentran disponibles, la interfaz Master del sistema (conformado por la UART y el sistema de debug) accede a este registro obteniendo sus datos.

Como se ha explicado previamente el funcionamiento del módulo **debugMode** depende totalmente de los valores establecidos en el registro 0 de la interfaz Avalon MM Slave, por lo que dependiendo de los valores enumerados anteriormente el sistema realizará diferentes funciones. Estas funciones se encuentran directamente vinculadas al módulo **interconexLogic** que depende directamente del módulo **debugMode**, por lo que su funcionamiento se encuentra directamente condicionado por los valores almacenados en el registro 0 citado previamente.

El módulo **interconexLogic** consta de un sistema de intercambio de señales, es decir, intercambia las señales de las interfaces del sistema con el microprocesador o la interfaz debug dependiendo del caso establecido.

Estas operaciones se realizan desconectando las señales requeridas del microprocesador realizando un bypass entre estas y la interfaz debug, trasladando así tanto los datos deseados desde el exterior del sistema hacia cualquier tipo de elemento conectado a este, como la lectura de distintos tipos de datos, ya sea del propio sistema o de elementos conectados a este.

En la *figura 5.4* se ilustran las señales de control utilizadas a través del sistema de debug para controlar el microprocesador y los diferentes periféricos vinculados a este. De forma que se muestran las señales que controlan la habilitación, tanto de las diferentes etapas de segmentación del microprocesador como del módulo **PC**, las señales de *clear* de las etapas y el **PC** para reiniciar el funcionamiento del microprocesador.

Adicionalmente, en la *figura 5.4* se muestran diferentes señales de control para el módulo **interconexLogic** que se encargará de realizar en conexionado necesario en cada tipo de operación recibida a través de la interfaz Avalon MM Slave.

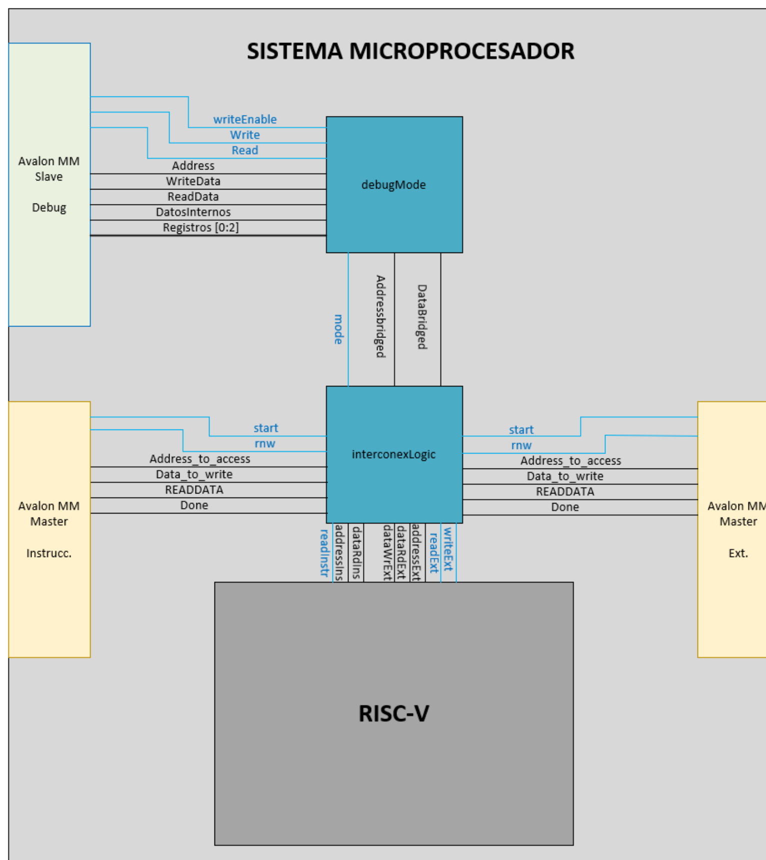

Figura 5.4: Señales pertenecientes al sistema de debug

Habiendo visto el funcionamiento del sistema de debug desarrollado. Sistema que permite la monitorización del sistema del microprocesador segmentado desde un dispositivo externo, el cual se introducirá en el siguiente apartado.

Capítulo 6

Periférico de comunicación serie UART

En el presente capítulo se describe el periférico UART. Este periférico se ha desarrollado de forma independiente al microprocesador, ya que en este caso el sistema UART se usa a modo de periférico y se conecta directamente al sistema de debug integrado en el microprocesador. Este periférico se encuentra conformado por los módulos lógicos encargados de la transmisión y recepción de datos, y la interfaz Avvalon MM Master, encargada de gestionar el envío y recepción de datos al sistema de debug integrado en el microprocesador (*figura 6.1*).

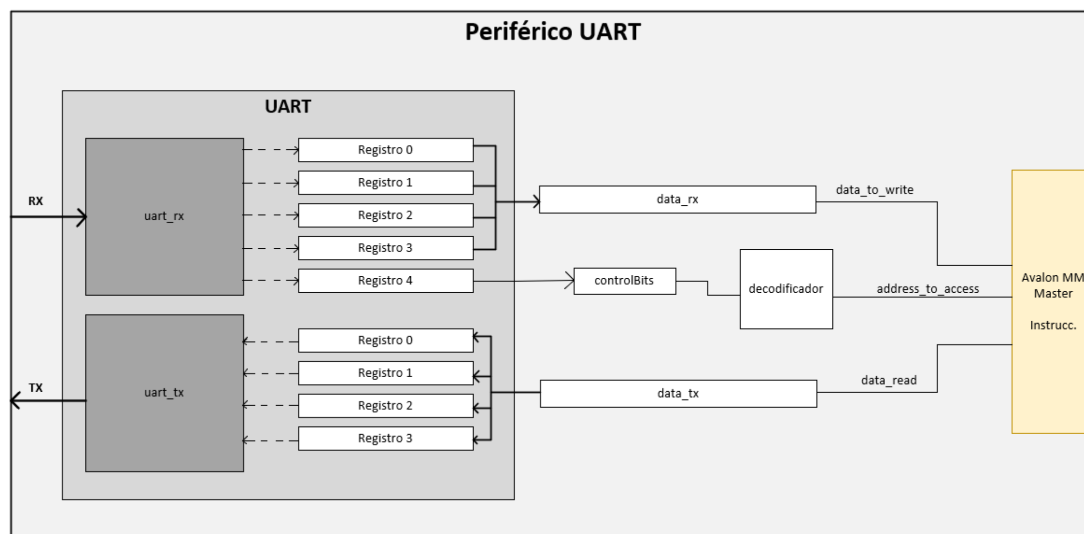

Figura 6.1: Esquemático del periférico UART

El sistema encargado de la transmisión y recepción de datos ha sido adaptado de un código abierto descargado desde la siguiente web *www.nandland.com* [14]. Este módulo ha sido modificado para adaptarse a las necesidades de este trabajo. Para eso se han instanciado los módulos encargados de la recepción de datos y del envío de estos (conocidos como **uart_rx** y **uart_tx** respectivamente) en un módulo cuya única finalidad es el control de la temporización de estos,

consiguiendo así que sen envíen y reciban los datos en el orden correcto.

Para realizar una transmisión asíncrona se debe de fijar una velocidad de transmisión conocida entre los dos dispositivos a interactuar. Para este trabajo se ha fijado una velocidad de 115200 baudios; esta velocidad de transmisión se ha fijado siguiendo una ecuación matemática que relaciona la velocidad del reloj usado por el módulo UART y la velocidad de transferencia de bits expresada en baudios, buscando así minimizar el número de ciclos de reloj a esperar para transmitir o recibir un dato.

$$CLKS_{perBit} = \frac{CLK_{SPEED}}{BAUD_RATE} = \frac{50MMz}{115200Bauds} \simeq 434 \tag{6.1}$$

De esta forma se necesitarán 434 ciclos de reloj para enviar cada bit. Sin embargo, al disponer de un reloj de 50 MHz esta operación únicamente tomará 8.68 μs por cada bit, de esta forma será posible realizar un gran numero de transmisiones en el menor tiempo posible, optimizando el uso del canal.

Así, el módulo conocido como **UART** fijará la cantidad de datos a recibir y a transmitir. En cada recepción de datos se reciben 40 bits (32 bits de datos y 8 bits de control), los bits de control son necesarios para direccionar los datos provenientes del periférico UART al registro deseado en la interfaz Slave del sistema de debug. Sin embargo, únicamente son necesarios dos bits, pero el sistema únicamente admite transmisiones de múltiplos de 8 bits por lo que es necesario transmitir datos extra.

En el caso de la transmisión, únicamente se envían 32 bits de datos, siendo estos los datos del registro de solo lectura conocido como el registro 3 (*figura 5.2*). Todo esto se ve condicionado por el tamaño de paquete de bits transmitidos o recibidos, ya que tienen que ser de 8 bits como viene en la norma (*figura 6.2*).

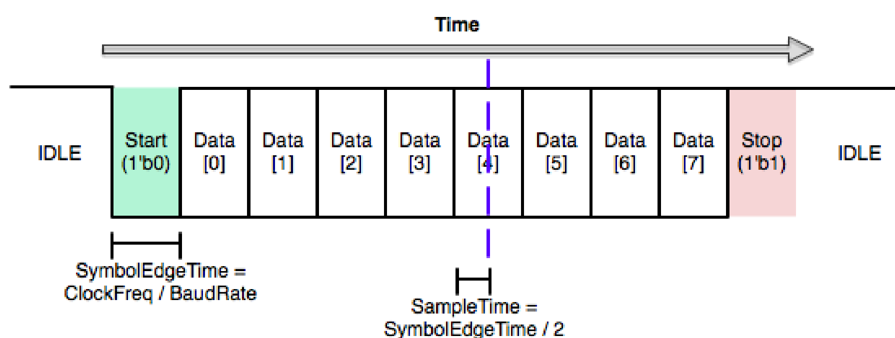

Figura 6.2: Esquema transmisión estándar de 8 bits

Como se muestra en la *figura 6.2*, para realizar cada transmisión se necesitan tener en cuenta diferentes factores, entre ellos, la utilización de un bit de inicio de transmisión y otro de finalización de esta, ya que al ser una transmisión serie únicamente contamos con un canal de transmisión sin indicadores adicionales a este. Los bits de control se utilizan para marcar el inicio y el fin de la transmisión.

En el caso indicar el inicio de la transmisión, se envía un bit a nivel bajo durante el tiempo de

bit establecido anteriormente. Cuando el tiempo de envío de este bit acaba, se toman las muestras de los datos hasta recibir el bit de finalización de la transmisión, el cual al contrario que el de inicio, se encuentra a nivel alto. En el momento que se acaba la transmisión de este bit el canal mantiene el nivel alto hasta la llegada de un nuevo bit de inicio, estableciendo el nivel alto como el nivel por defecto de la conexión.

Otro factor a tener en cuenta es el instante de muestreo de bit. Como al inicio de este pueden haber inestabilidades, normalmente se opta por realizar el muestreo justo a la mitad del tiempo de muestreo de cada bit, siendo en este caso $434/2$, lo que equivale a 217 ciclos de reloj desde el inicio del bit.

Una vez tenidos en cuenta estos factores se necesita estructurar un sistema que sea capaz de recibir datos en múltiplos de 8 bits de los módulos **uart_rx** y **uart_tx**, y almacenarlos de forma que se puedan concatenar. En el caso de la recepción de datos, se realiza en un registro de 32 bits y otro de 8 bits, y en el caso de la transmisión en un registro de 32 bits. Todo esto se muestra en la *figura 6.1*, encontrando una solución en una estructura que funciona de la siguiente manera dependiendo del tipo de transmisión requerida.

- En el caso de la **transmisión de datos**, al iniciarse la transmisión, primero se transmite el registro 0 que está conformado por los 8 bits de menor peso de los datos provistos por el sistema del microprocesador. Al activarse la señal que confirma la finalización del envío de estos datos se selecciona el siguiente registro que contiene los siguientes 8 bits, transmitiendo así los 32 bits recibidos mediante el uso de los registros auxiliares.

- Para la **recepción de datos** se sigue el mismo sistema usado para la transmisión aunque adaptándolo al uso de un registro adicional. Esto es debido a los bits extra necesarios que se han mencionado previamente. Por lo que primero se escriben los primeros 8 bits recibidos en el registro 0, y una vez finalizada esa transmisión se realiza la misma transacción con los registros restantes. Una vez se escriben los valores recibidos en el registro 4 (observado en la *figura 6.1*), únicamente se toman sus dos bits de menor peso, seleccionando estos últimos el registro de la interfaz del sistema debug con el que se va a interactuar (*figura 5.2*).

Bits	Dirección (Hex.)	Funcionalidad
00	00000000	Almacena el comando de control del sistema de debug en el registro 0.
01	00000004	Escribe la dirección a almacenar en el registro 1.
10	00000008	Escribe los datos a escribir en el registro 2.
11	00000000	Reservada para posibles ampliaciones.
–	0000000C	Realiza la lectura del registro 3 para la posterior transmisión de datos.

Tabla 6.1: Señales de control del periférico UART

Una de las principales limitaciones de este periférico es el propio bus utilizado, ya que únicamente puede realizar una transmisión en cada instante, por lo que hasta que esta no finalice no se pude realizar el envío o transmisión de otro dato diferente. Sin embargo, este factor limitante se atenúa mediante la limitación del uso del bus exclusivamente en el instante que se han recibido

todos los datos, o en el caso de la transmisión, cuando se ha recibido una petición para realizar una transferencia de envío. En estos casos puntuales el bus se encontrará ocupado realizando la transferencia de los datos requeridos y no podrá realizar ningún otro tipo de operación.

Cabe recalcar que el periférico puede llevar a cabo operaciones de envío y de recepción de forma paralela, siendo únicamente incompatibles en el momento de la interacción con el bus para realizar algún tipo de consulta. Ya que cuando el bus se encuentra ocupado realizando una transferencia es incapaz de realizar otra simultáneamente, por lo que únicamente se realizará la operación que haya bloqueado el bus.

Una vez finalizada la implementación de este periférico se han seguido los mismos pasos que en la *sección 4.1* de esta memoria, almacenando este componente como un periférico en una librería de uso propio para así poder reutilizar este periférico en cualquier sistema [15].

Para hacer uso de este periférico es necesario desarrollar un software externo que nos permita enviar y recibir comandos desde un PC. Por esto, a través del compilador PyCharm haciendo uso del lenguaje de programación Python, se ha implementado una aplicación la cual permite una monitorización y una modificación del sistema implementado en tiempo real. Pudiendo modificar el contenido de las memorias y programar el sistema desde un equipo externo con una interfaz gráfica.

6.1. Aplicación de control

Como se ha hecho mención previamente, se ha programado una aplicación que permite hacer uso de las distintas funcionalidades implementadas en el módulo de debug integrado en el sistema, usando las utilidades provistas por el periférico UART de control. Por lo que esta aplicación se centra en la explotación de estas funcionalidades con una interfaz gráfica.

Figura 6.3: Aplicación de control debug

La aplicación se ha programado con el lenguaje de programación Python 3.8.8 [16], siendo un lenguaje de programación de libre uso, por lo que no es necesaria una licencia para desarrollar sobre él. Esta implementación ha hecho uso de las librerías *tkinter* [17] y *serial* [18]. Estas librerías proporcionan soporte para el diseño gráfico de la propia aplicación y el uso de transmisiones serie.

La interfaz gráfica desarrollada puede verse en la *figura 6.3*. En esta interfaz se muestran las distintas opciones de funcionamiento programadas, detallándose a continuación sus características.

- Capacidad de **selección del puerto de comunicación serie** a utilizar entre los 256 disponibles en el propio sistema operativo, haciendo así que se pueda elegir cualquier puerto sin necesidad de cambios al código para establecer conexión.

- Una vez abierto el puerto se habilita la opción de **activar el sistema debug** del sistema, el cual deshabilita la ejecución de este hasta su desactivación. Este sistema habilita todas las características vistas en el capítulo 5 de esta memoria.

- Como característica más importante del trabajo se ha implementado la capacidad de **escribir en la memoria de instrucciones** del sistema, por lo que se puede sobre-escribir el programa que se encuentra en la memoria del sistema, reiniciando la ejecución de este sobre el nuevo programa cargado. El programa a cargar en memoria se deberá copiar en hexadecimal separado por saltos de línea, de forma que cada línea será interpretada como un dato de 32 bits a escribir en cada posición de memoria, por lo que una vez copiado todo el código para programar el microprocesador se realiza el envío de este a las memorias del sistema a través del sistema de debug y los módulos *debugMode* y *interconexLogic* vistos en la *figura 5.4*.

- **Escritura en dispositivos a través de interfaz de memoria de datos**, conociendo la dirección base del dispositivo se hace uso de las herramientas proporcionadas por el módulo debug (*figura 5.2*). Este módulo permite el uso de sus registros internos para poder escribir datos a los periféricos conectados, pudiendo modificar sus valores a tiempo real.

- Posibilidad de **leer los valores almacenados en la memoria de instrucciones y en los dispositivos mapeados en la memoria de datos**. Mediante la selección de estos, y conociendo la dirección a leer se pueden obtener los datos almacenados tanto en las memorias de instrucciones como en los elementos externos del sistema.

- De la misma forma que se pueden leer los valores de las memorias, se ha implementado la posibilidad de **acceder al valor del PC** en el momento que se ha parado el sistema, y del próximo valor de este.

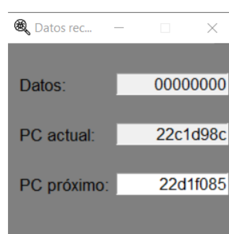

Figura 6.4: Ventana de recepción de datos

- Finalmente, mediante el control de las señales de habilitación de las etapas de segmentación del microprocesador y del módulo PC, se ha implementado un sistema que permite la **ejecución de hasta 10 pasos en el microprocesador** mientras se encuentra parado en modo debug. Observando así los cambios que puede realizar a través de las diferentes funcionalidades del sistema.

Una vez solicitados los datos de lectura tanto de las memorias de instrucciones, como de los periféricos externos o del registro PC, se abrirá una nueva ventana la cual contendrá los valores recibidos (*figura 6.4*). En el caso de encontrar algún tipo de error en cualquier transmisión, la propia aplicación notificará mediante una ventana emergente mostrando la causa del error.

Capítulo 7

Verificación

Una vez descrito el funcionamiento del los diferentes sistemas que desarrollados en el presente trabajo, junto con sus módulos internos más importantes, se ha de verificar su correcto funcionamiento. Para ello, previamente a la implementación física del sistema se ha verificado su correcto comportamiento mediante simulaciones. Gracias a estas simulaciones se han detectado diferentes errores que se han ido subsanado en el transcurso de la implementación del trabajo.

7.1. Simulaciones del sistema

En un sistema electrónico es importante verificar por separado el funcionamiento de cada uno de los módulos que lo componen, y posteriormente realizar la verificación del sistema completo.

Para realizar la verificación del sistema y de cada una de sus componente se han desarrollado diferentes simulaciones. Para ello se hace uso de una herramienta proporcionada por **Platform Designer**, cuya función es generar un testbench del sistema.

Para simular el sistema, primero se han de integrar los módulos deseados, de forma que mediante el uso del menú ***Generate***/***Generate Testbench System*** se generará un sistema listo para simular con el software **ModelSim**.

Sin embargo, antes de realizar las simulaciones se han de modificar diferentes parámetros. Los cambios requeridos se realizan en un archivo situado en la carpeta raíz donde se ha creado previamente el Testbench del sistema (en este caso el sistema creado se denomina AvalonRiscV_QSYS), siendo la ruta:

//AvalonRiscV_QSYS/testbench/AvalonRiscV_QSYS_tb/simulation

En esta ruta se encuentra el fichero Verilog que define los estímulos de la simulación, encontrándose por defecto únicamente con el sistema generado sin ningún tipo de estímulo. Modificar este archivo agrega complejidad a la simulación, y permite orientarla a la verificación de un determinado componente del sistema.

Para poder compilar el Testbench se necesita un archivo generado por **Platform Designer** denominado **msim_setup.tcl**, el cual se encuentra en la siguiente ruta:

//AvalonRiscV_QSYS/testbench/mentor

Este archivo se ha de copiar a la carpeta raíz donde se encuentra el fichero de estímulos. Posteriormente se ha de modificar uno de los parámetros internos de este archivo, por lo que se edita con el software **Notepad++** agregando en el parámetro **QSYS_SIMDIR** la expresión ”../..”(*figura 7.1*) determinando que el directorio de trabajo principal es el lugar donde se encuentra el archivo **msim_setup.tcl**.

```
100  # Initialize variables
101  if ![info exists SYSTEM_INSTANCE_NAME] {
102    set SYSTEM_INSTANCE_NAME ""
103  } elseif { ![ string match "" $SYSTEM_INSTANCE_NAME ] } {
104    set SYSTEM_INSTANCE_NAME "/$SYSTEM_INSTANCE_NAME"
105  }
106
107  if ![info exists TOP_LEVEL_NAME] {
108    set TOP_LEVEL_NAME "AvalonRiscV_QSYS_tb"
109  }
110
111  if ![info exists QSYS_SIMDIR] {
112    set QSYS_SIMDIR "../.."
113  }
```

Figura 7.1: Modificación del archivo msim_setup.tcl

Estas simulaciones se van a separar en tres, ya que se realizará un estudio de las respuestas del periférico UART, del sistema de debug integrado en el microprocesador y de la propia adaptación del microprocesador segmentado, en el cual se profundizará a continuación.

7.1.1. Adaptación del microprocesador segmentado

Para comprobar el correcto funcionamiento del microprocesador [19] primero se han centrado las simulaciones en el desempeño realizado por los módulos de sincronización (*figura 4.3*), ya que son la principal adición al microprocesador segmentado para adaptarlo a las interfaces Avalon.

Machine Code	Basic Code	Original Code
0x00100113	addi x2 x0 1	addi x2,x0,1
0x10000193	addi x3 x0 256	addi x3,x0,0x100
0x0021a023	sw x2 0(x3)	sw x2,0(x3)
0x00000013	addi x0 x0 0	nop
0x00000013	addi x0 x0 0	nop
0x0001a203	lw x4 0(x3)	lw x4,0(x3)

Figura 7.2: Código a ejecutar en primera instancia

Para comprobar este funcionamiento se han cargado previamente en la memoria de instrucciones un código de lectura de datos externos y escritura de estos mismos (*figura 7.2*), de forma que se realice la prueba de los posibles casos que alteran el funcionamiento del sistema mediante el módulo **controlEnables**.

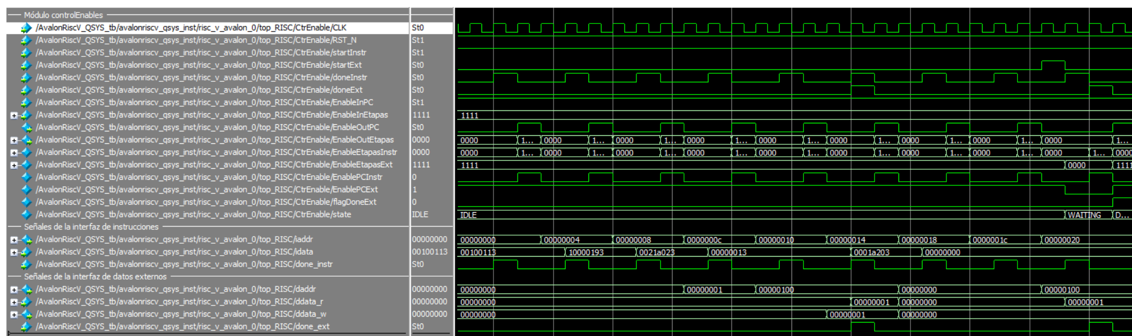

Figura 7.3: Resultado de la simulación del módulo controlEnables

Como se observa en la *figura 7.3*, la respuesta del sistema es la esperada, ya que mientras se está realizando la lectura en las memorias (ya sean de las de datos externos o de instrucciones) el microprocesador se encuentra detenido. Se aprecia como la señal de *Enable* que controla al microprocesador oscila en función de la señal de finalización de la transmisión realizada. Es decir, hasta que se recibe la confirmación de finalización de la transmisión el microprocesador permanece parado, activándose tras la activación de esta.

Adicionalmente se puede observar como los datos obtenidos a través de las señales de lectura y escritura de datos del propio microprocesador son los esperados, siendo estos el resultado del procesamiento del código cargado. Por lo que observando que todos los datos cargados se han procesado, se puede asumir que su comportamiento es el adecuado.

7.1.2. Periférico UART

En el caso del periférico UART se ha seguido un modelo de simulación diferente, ya que se ha requerido un modelo que generase el envío de datos por parte de una UART. Esta complejidad adicional se ha de complementar con un código que simule el comportamiento de una UART con capacidad para la transmisión de datos.

Para esto se ha inicializado por defecto el canal a nivel alto (valor estándar por defecto), y una vez se hace la llamada a este código se envía el bit de inicio con un valor de 0 durante el tiempo de bit definido previamente. Una vez se ha enviado este bit, se empiezan a enviar bit a bit (empezando por el de menor peso) todos los datos introducidos, durando la transmisión de cada bit el tiempo fijado en el periférico (*capítulo 6* de esta memoria). Una vez acabada la transmisión se envía un bit a nivel alto manteniéndolo hasta una nueva petición, siguiendo así el esquema observado en la *figura 6.2*.

Para poder interactuar con el periférico de transmisión serie UART, se ha modelado su comportamiento en el testbech (*figura 7.4*), de forma que este sea capaz de transmitir datos hacia el sistema de forma versátil.

A la hora de enviar los 40 bits de datos requeridos al periférico se han realizado 5 transmisiones seguidas. De forma que se concatenan los valores de las diferentes transmisiones, formando los datos a enviar al periférico.

Como prioridad a la hora de enviar los datos, se ha elegido primero enviar la dirección de

memoria donde se van a almacenar los datos. Posteriormente los propios datos a enviar, y para finalizar, la instrucción de control deseado. Siendo este orden necesario para que el sistema no actúe sobre sobre una dirección previamente cargada.

```
53  // Takes in input byte and serializes it
54    task UART_WRITE_BYTE;
55      input [7:0] i_Data;
56      integer     ii;
57      begin
58
59        // Send Start Bit
60        r_Rx_Serial <= 1'b0;
61        #(c_BIT_PERIOD);
62        //#1000;
63        repeat(c_CLKS_PER_BIT) @(posedge r_Clock);
64
65        // Send Data Byte
66
67            for (ii=0; ii<8; ii=ii+1)
68              begin
69                r_Rx_Serial <= i_Data[ii];
70                #(c_BIT_PERIOD);
71                repeat(c_CLKS_PER_BIT) @(posedge r_Clock);
72
73              end
74
75          // Send Stop Bit
76          r_Rx_Serial <= 1'b1;
77          #(c_BIT_PERIOD);
78          repeat(c_CLKS_PER_BIT) @(posedge r_Clock);
79
80
81
82       end
83    endtask // UART_WRITE_BYTE
```

Figura 7.4: Código que simula el comportamiento de una UART de transmisión

Una vez definidas las pautas a seguir para simular el sistema de recepción del periférico UART, se ha de simular el sistema de envío. Para esto se ha de insertar una instrucción de control para que el propio sistema del microprocesador envíe un dato al periférico UART, para este posteriormente transmitirlo, activando así las señales pertinentes para esta operación y completando la verificación de este periférico.

Figura 7.5: Resultado de la simulación del periférico UART

Viendo en la *figura 7.5* como los datos recibidos se corresponden con los enviados (los mencionados anteriormente en la *figura 7.2*), así como los datos solicitados de la memoria externa y de la memoria de instrucciones son los buscados. Confirmando además el correcto funcionamiento de una parte del sistema de debug en el que se profundizará a continuación.

7.1.3. Sistema de debug integrado en el microprocesador

Una vez el propio microprocesador ha sido adaptado para el funcionamiento con las interfaces Avalon MM y se ha verificado el correcto funcionamiento del periférico UART, las próximas características a simular para comprobar su funcionamiento son las que conforman el sistema de debug. Esto es, tanto el módulo **debugMode** como el módulo **interconexLogic** los cuales adicionalmente infieren tanto en el control del funcionamiento del microprocesador, como en el de sus interfaces Avalon MM Master. Para esto se ha utilizado el mismo código de prueba visto en la *figura 7.2*, sin embargo en este caso se han integrado diferentes comandos de control para probar todas las características implementadas en el sistema debug (detalladas en el *capítulo 5* de la memoria).

Primero se comprobará que funciona correctamente la escritura de datos, tanto en la memoria de instrucciones como en la memoria de datos, introduciendo el código mencionado anteriormente junto con un valor aleatorio a escribir en una memoria externa.

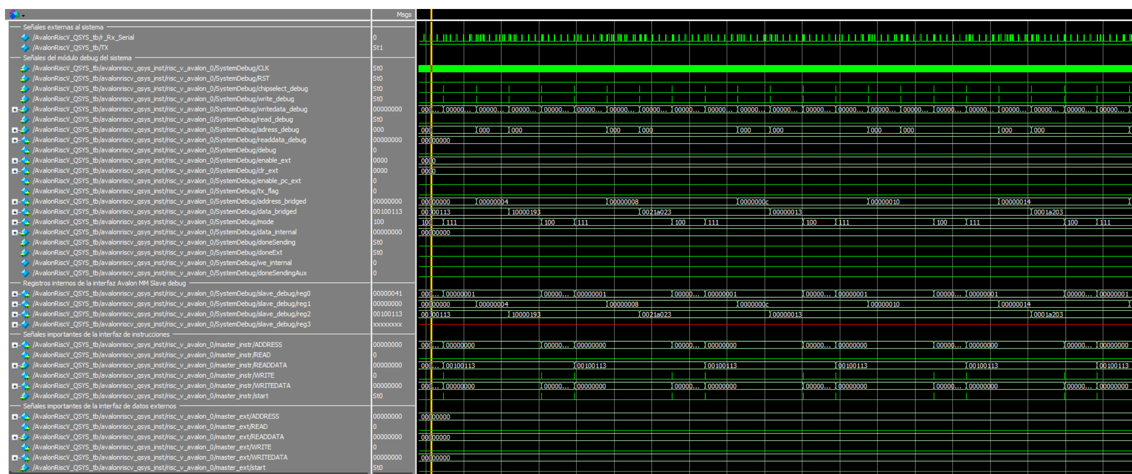

Figura 7.6: Resultado de la simulación de escritura de datos

Observando como se han realizado las operaciones requeridas; para confirmarlo se puede mostrar el contenido interno de las memorias situadas en el sistema simulado, observando así que los datos escritos son los correctos.

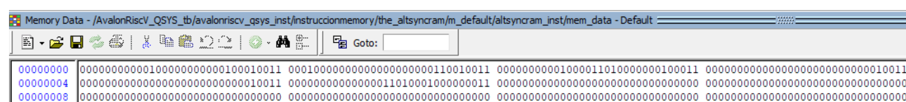

Figura 7.7: Contenido de las memorias en la simulación

Aprovechando los datos ya escritos en las memorias (*figura 7.7*) se modificará el fichero de estímulos únicamente añadiendo instrucciones de lectura, para así verificar la correcta lectura de los datos escritos previamente.

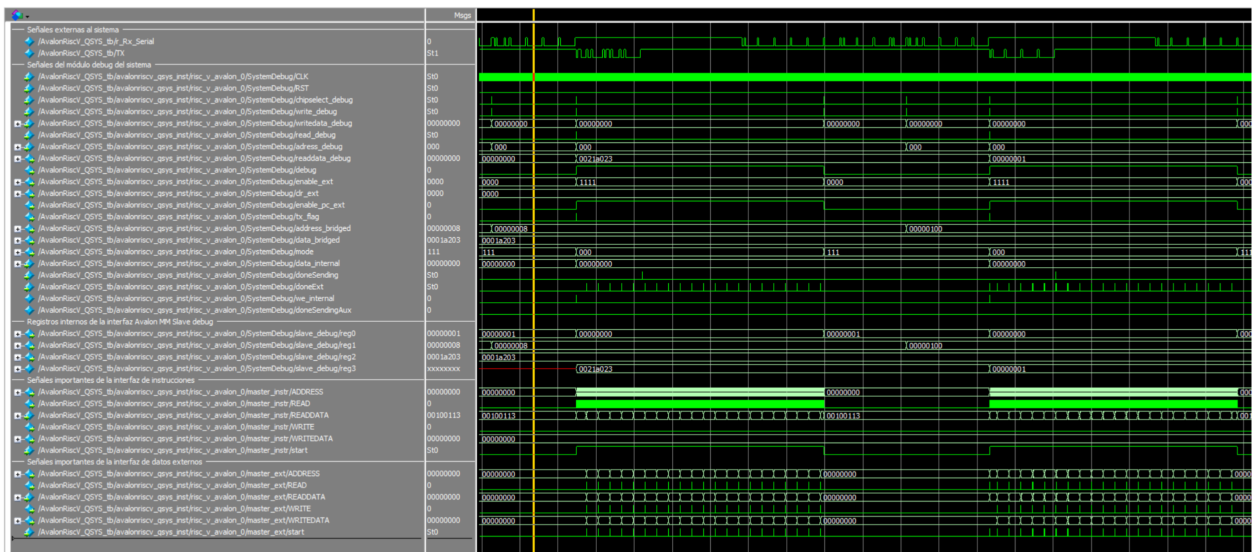

Figura 7.8: Resultado de la simulación de lectura de datos

Leyendo las direcciones que contienen datos de la ejecución anterior, en este caso se ha elegido la dirección 00000008 de la memoria de instrucciones, y el dirección 00000100 de dispositivos externos. Se aprecia como el dato a enviar transmitido hacia el periférico UART es el deseado, y posteriormente es el dato enviado mediante la comunicación serie como se ha visto previamente en la *figura 7.5*.

De la misma forma que se ha realizado la lectura de los datos deseados se puede leer el contenido actual del módulo PC así como el próximo valor calculado previamente a la detención del sistema para activar el modo debug.

Figura 7.9: Resultado de la simulación de lectura del PC actual y siguiente

Para finalizar las simulaciones se hará una prueba del control de la ejecución del sistema, es decir de la ejecución por pasos, añadiendo así un código a probar dentro del sistema. En este caso se añadirá un código más completo para verificar su comportamiento ante saltos de instrucciones.

Machine Code	Basic Code	Original Code
0x12300993	addi x19 x0 291	addi x19,x0,0x123
0x10000893	addi x17 x0 256	addi x17,x0,0x100
0x002ff937	lui x18 767	lui x18 0x2ff
0x0138a223	sw x19 4(x17)	sw x19,4(x17)
0x001a0a13	addi x20 x20 1	addi x20,x20,1
0x000000b3	add x1 x0 x0	add x1,x0,x0
0x0148a023	sw x20 0(x17)	sw x20,0(x17)
0x00108093	addi x1 x1 1	addi x1,x1,1
0xff2088e3	beq x1 x18 -16	beq x1,x18,loop2
0x00000013	addi x0 x0 0	nop
0xff209ae3	bne x1 x18 -12	bne x1,x18,loop

Figura 7.10: Código utilizado finalmente

Apreciando así en la simulación como se permite la ejecución del microprocesador en 95 ciclos, ya que se ha fijado este número en el código usado para simularlo (pudiéndose elegir cualquier valor entre 1 y 256). De forma que las señales Enable que controlan la ejecución del microprocesador se mantendrán a nivel alto el tiempo suficiente para realizar un ciclo dentro del sistema, es decir, que se realice la lectura de la instrucción correspondiente y se interprete, entendiendo esto como un ”paso”dentro del sistema.

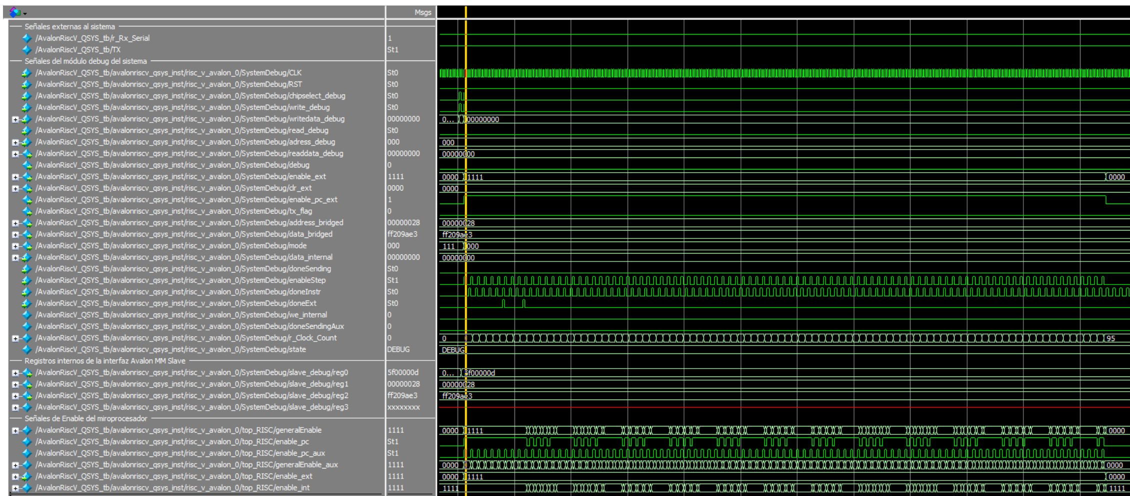

Figura 7.11: Resultado de la simulación de ejecución por pasos

En la *figura 7.11* se aprecia como las señales de Enable se mantienen activadas hasta realizar cada ciclo, sin embargo se activan únicamente 95 veces para ejecutar los ciclos descritos mediante el comando enviado al sistema.

Capítulo 8

Implementación Hardware

Una vez descrito y verificado el sistema en un entorno virtual, se procede a la implementación física de este para así verificar su funcionamiento en un entorno más realista y poder sacar partido a las características de este mediante el uso de hardware preparado para esto.

Para la realización de este trabajo se ha requerido de diferente hardware específico:

- **Cable adaptador USB - RS232**, necesario para conectar el PC con la placa de evaluación.
- Placa de evaluación **DE2-115 Cyclone 4 EP4CE115F29C7** donde se realizaran todas las pruebas de funcionamiento del sistema.
- **Un ordenador portátil genérico** con un entorno basado en Windows 10 Pro.

Para esto se utilizará la placa de evaluación **DE2-115 Cyclone 4 EP4CE115F29C7**, el cable adaptador de USB a RS232 y un USB tipo B para programar la placa. Una vez conectada la placa al ordenador donde se realizará la evaluación del sistema se compilará el conjunto de este, haciendo así que mediante la herramienta **Platform Designer** se genere un sistema que pueda ser instanciado mediante el propio Verilog.

Para esto se crea un nuevo proyecto de Quartus donde únicamente se encuentre un módulo, el que previamente será el módulo superior de toda la jerarquía, donde se instancie el sistema formado por los módulos descritos a lo largo de esta memoria.

De la misma forma que en la *figura 4.7* se instancian las partes que conformarán el sistema, en este caso los módulos generados previamente, como son el propio microprocesador con sus sistema de debug integrado, y el periférico UART. Además de estos se agrega un controlador de los módulos de 7 segmentos [11] encontrados en la placa de evaluación y dos memorias, una para el almacenamiento de instrucciones y otra para el almacenamiento externo (*figura 8.1*).

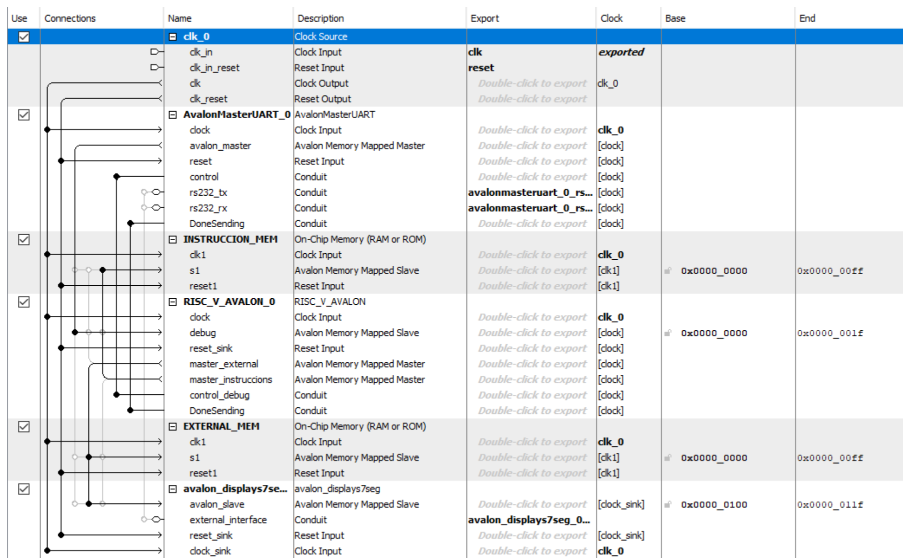

Figura 8.1: Sistema final

Una vez instanciados los módulos queridos, se generan los ficheros necesarios para la posterior compilación mediante Verilog para poder programar la placa de evaluación. Esto se realiza mediante la herramienta integrada en **Platform Designer**, seleccionando ***Generate/Generate HDL*** como podemos ver en la *figura 8.2* donde se habilita la posibilidad de exportar el sistema al lenguaje Verilog para su posterior síntesis.

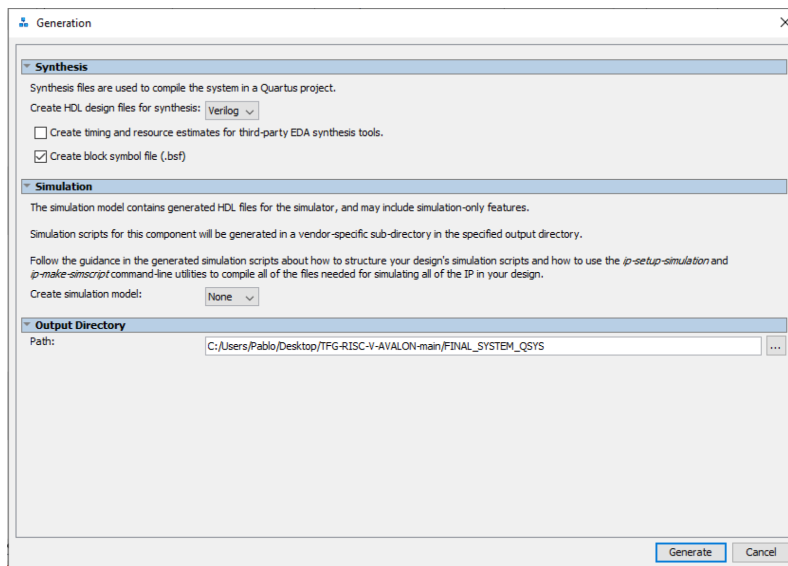

Figura 8.2: Generar el sistema para la compilación en Verilog

Una vez exportado el sistema, se ha de integrar en el proyecto de Quartus. Para esto se añade el archivo central del sistema a los archivos del proyecto, el cual interpreta el compilador una

vez se instancia. Para realizar la instanciación de las señales del sistema generado previamente se ha generado mediante otra herramienta integrada en **Platform Designer** localizada en el menú ***Generate/Show Instantiation Template***, de forma que se genera una plantilla personalizada para instanciar las señales del sistema (*figura 8.3*).

```
FINAL_SYSTEM_QSYS u0 (
    .avalon_displays7seg_0_external_interface_conduit1 (<connected-to-avalon_displays7seg_0_external_interface_conduit1>),
    .avalon_displays7seg_0_external_interface_conduit0 (<connected-to-avalon_displays7seg_0_external_interface_conduit0>),
    .avalon_displays7seg_0_external_interface_conduit2 (<connected-to-avalon_displays7seg_0_external_interface_conduit2>),
    .avalon_displays7seg_0_external_interface_conduit3 (<connected-to-avalon_displays7seg_0_external_interface_conduit3>),
    .avalon_displays7seg_0_external_interface_conduit4 (<connected-to-avalon_displays7seg_0_external_interface_conduit4>),
    .avalon_displays7seg_0_external_interface_conduit5 (<connected-to-avalon_displays7seg_0_external_interface_conduit5>),
    .avalon_displays7seg_0_external_interface_conduit6 (<connected-to-avalon_displays7seg_0_external_interface_conduit6>),
    .avalon_displays7seg_0_external_interface_conduit7 (<connected-to-avalon_displays7seg_0_external_interface_conduit7>),
    .avalonmasteruart_0_rs232_rx_rx                    (<connected-to-avalonmasteruart_0_rs232_rx_rx>),
    .avalonmasteruart_0_rs232_tx_tx                    (<connected-to-avalonmasteruart_0_rs232_tx_tx>),
    .clk_clk                                           (<connected-to-clk_clk>),
    .reset_reset_n                                     (<connected-to-reset_reset_n>)
);
```

Figura 8.3: Plantilla de instanciación proporcionada

Al instanciar el sistema y compilar el proyecto de Quartus, finalmente se han de asignar las señales de este a los pines físicos de la placa de evaluación usando la herramienta de Quartus llamada **Pin Planner**. Por lo que recurriendo al manual de instrucciones de la propia placa y a esta herramienta se asignan los pines a usar como se aprecia en la *figura 8.4*.

Figura 8.4: Herramienta Pin Planner

Finalmente, tras realizar la asignación de pines se compila el proyecto y se procede a usar la herramienta de programación denominada **Programmer**, la cual permite mediante un sistema integrado de JTAG programar los elementos lógicos de la placa para que se comporte de la forma que se ha programado (*figura 8.5*).

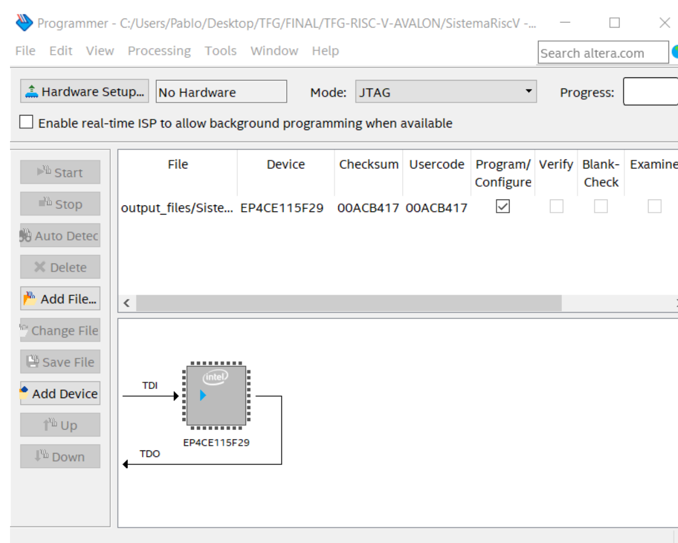

Figura 8.5: Programación de la placa de evaluación

Como adición a los datos proporcionados mediante la compilación del sistema, el propio compilador proporciona características del funcionamiento del código implementado respecto a la placa de evaluación usada. Obteniendo así las siguientes características:

Característica	Detalles
Elemento lógicos totales	**5409** / 114480 (5 %)
Registros usados	**3160**
Pines usados	**60** / 529 (11 %)
Bits de memoria usados	**3313664** / 3981312 (83 %)
Frecuencia máxima del sistema	**74.65 MHz** (Limitado por hardware a 50 MHz)

Tabla 8.1: Características del sistema final

Una vez programado el sistema y conocidas sus características principales, se hace uso de la herramienta debug diseñada para controlar el sistema. El propio sistema se encuentra diseñado de forma que una vez es programado, carga la información almacenada en la memoria de instrucciones, independientemente de si es válida o no. Esto se ha diseñado para el posible caso de integrarse en un hardware con instrucciones prefijadas, ejecutando el microprocesador directamente estas instrucciones sin necesidad de ninguna interacción externa con este.

Sin embargo, la placa de evaluación una vez es apagada pierde los datos almacenados en sus memorias, por lo que hay que programarla cada vez. Por lo cual se hará uso de la aplicación diseñada específicamente para esto, cargando los datos deseados y reiniciando la ejecución para tratar los datos cargados de la forma deseada.

Observando el comportamiento de la placa se ha comprobado como el sistema funciona de forma óptima en su implementación física. Por lo que se convierte en un sistema funcional física-

mente en cualquier tipo de FPGA, pudiéndose adaptar a estas una vez el sistema es recompilado mediante Quartus (encargado de adaptar el sistema) y los pines se reasignan (dependiendo de la FPGA utilizada) siguiendo el mismo sistema que en este trabajo.

Capítulo 9

Conclusiones

Como conclusión de este trabajo, los objetivos de este eran realistas, de forma que se han alcanzado la mayoría de ellos. A través del desarrollo del TFG se ha visto la importancia de los sistemas SoC en todo tipo de sistemas, por lo que un desarrollo como el llevado a cabo puede tener gran repercusión debido principalmente al uso del estándar de bus Avalon, permitiendo integrar cualquier tipo de periférico que se requiera sin realizar modificaciones internas. Además el uso de la ISA RISC-V hace de este trabajo una alternativa a los diferentes sistemas propietarios que se encuentran en el mercado en la actualidad.

A lo largo del trabajo han aparecido diferentes dificultades, entre ellas la des-sincronización de los diferentes módulos pertenecientes al microprocesador tras la adaptación al estándar de bus Avalon, solucionada como se ha detallado en el *capítulo 4* de la memoria.

Otra de las dificultades más remarcables del desarrollo del trabajo se encuentra en el módulo debug, ya que este no era capaz de temporizar correctamente el envío y recepción de datos entre las interfaces y el periférico UART, sin embargo con la ayuda de diversas simulaciones se logró corregir su funcionamiento.

Como es habitual en proyectos de diseño y verificación, gran parte del trabajo ha consistido en la corrección de errores que fueron aparecieron a través del proceso de implementación y validación de los diferentes sistemas y periféricos. Tras cumplir con las tareas programadas, se han llegado a completar los objetivos planteados al inicio del Trabajo Final de Grado:

- Adaptación de la implementación previa del microprocesador segmentado al estándar de buses Avalon MM Master y Slave, desarrollado en la secciones 4 y 5.
- Integración de un sistema de debug que pueda ser monitorizado externamente, y que de la misma manera altere el funcionamiento del sistema desde un elemento exterior a este.
- Implementación de un periférico cuya finalidad es recibir y transmitir la información necesaria por el sistema de debug, de forma que se establezca una canal de comunicaciones entre el microprocesador y el exterior.
- Complementando al anterior punto, y como se ha descrito en la sección 6.1, se ha desarrollado una aplicación para interactuar externamente con el sistema generado.

- Mediante las simulaciones y la posterior verificación física del sistema se ha confirmado la funcionalidad de este en un entorno real, de forma que se puede generar un sistema genérico con todo tipo de periféricos únicamente usando como núcleo los sistemas descritos a lo largo de la memoria, pudiendo usar en estos todo tipo de instrucciones suportadas por el microprocesador (tabla 2.1).

Capítulo 10

Lineas futuras

Como lineas futuras de este trabajo, se detallan las características que pueden ser explotadas en un futuro, de las cuales cabe destacar:

- Implementación de un módulo **prefetch** para precargar cierto rango de datos de las memorias, y así minimizar el impacto que supone la carga de estas en el tiempo de proceso.
- Ampliación de funciones de las interfaces Avalon MM, añadiendo así el modo de transmisión tipo **Burst**, el cual añadiría la funcionalidad de transmitir paquetes de datos entre sistemas sin realizar mas de una transmisión, optimizando así el funcionamiento de algunos periféricos que requieran de envíos de tramas de datos concatenados.
- Optimización del rendimiento de la segmentación del microprocesador, mejorando así la frecuencia de reloj del sistema.
- Añadir soporte para operaciones de multiplicación, ampliando así el set de instrucciones admitidas por el microprocesador.
- Agregar la característica de control de interrupciones, mejorando así la comunicación con diversos periféricos que requieren de esta para funcionar correctamente, mejorando así su desempeño en el sistema.
- Añadir nuevas funcionalidades al sistema de debug para lograr un mejor control en tiempo real y mejor monitorización de los recursos usados en la placa de evaluación.

El trabajo partía de una implementación simplificada y sólida de un microprocesador basado en la arquitectura RISC-V. La mayor parte de desarrollos hechos consisten en la mejora de la versión segmentada del mismo, y su adaptación para trabajar con un estándar en la comunicación con memorias. Partiendo de la filosofía modular de RISC-V, el sistema se ha implementado de forma que se pueda modificar y ampliar de forma sencilla, dejando muchas señales con posibilidad de ampliación sin modificar las conexiones entre los módulos. Esta característica garantiza que la solución pueda ser reutilizada y ampliada en líneas de trabajo futuro.

Bibliografía

[1] JEFF SHEPARD. "RISC-V for ultra-low power processing and AI on the edge". En: *microcontrollertips.com* (nov. de 2020). URL: https://www.microcontrollertips.com/risc-v-for-ultra-low-power-processing-and-ai-on-the-edge/.

[2] Breixo Gómez. URL: https://www.profesionalreview.com/2021/04/15/que-es-un-soc-caracteristicas/.

[3] David Patterson y Andrew Waterman. *Guía Práctica de RISC-V: El Atlas de una Arquitectura Abierta*. Primera edición. URL: http://riscvbook.com/spanish/guia-practica-de-risc-v-1.0.5.pdf.

[4] Comunidad de RISC-V. *RISC-V Specifications*. URL: https://riscv.org/technical/specifications/.

[5] Comunidad de Wikipedia. *RISC-V*. URL: https://es.wikipedia.org/wiki/RISC-V.

[6] Greg Novick Crystal Chen y Kirk Shimano. *How Pipelining Works on RISC-V*. URL: https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/pipelining/index.html.

[7] *Arquitecturas para procesado de datos (Tema 3 de Integración de Sistemas Digitales)*. URL: http://poliformat.upv.es/.

[8] Intel. *Avalon® Interface Specifications*. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf.

[9] Intel. URL: https://www.intel.com/content/dam/altera-www/global/en_US/documentation/iga1446487888057/iga1450181670900.jpg.

[10] Intel. URL: https://www.intel.co.jp/content/dam/altera-www/global/en_US/documentation/iga1446487888057/iga1450181688190.jpg.

[11] José M. Monzó Ferrer Ricardo José Colom Palero. *Creación de periféricos personalizados en Platform Designer (Práctica 2 de Codiseño Hardware Software)*. URL: http://poliformat.upv.es/.

[12] Ricardo José Colom Palero. *Diseño de sistemas integrados con Altera (Tema 1 de Codiseño Hardware Software)*. URL: http://poliformat.upv.es/.

[13] Intel. *Intel® Quartus® Prime Pro Edition User Guide, Platform Designer*. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-qpp-platform-designer.pdf.

[14] Nandland Community. *UART in VHDL and Verilog for an FPGA Implementation*. URL: https://www.nandland.com/vhdl/modules/module-uart-serial-port-rs232.html.

[15] José M. Monzó Ferrer. *Creación de periféricos personalizados en Platform Designer (Tema 2 de Codiseño Hardware Software)*. URL: `http://poliformat.upv.es/`.

[16] Juan Díaz. *Curso Python*. URL: `https://www.pildorasinformaticas.es/course/curso-python/`.

[17] Comunidad de Python. *Documentación paquete tkinter perteneciente a Python*. URL: `https://docs.python.org/es/3/library/tk.html`.

[18] Comunidad de Python. *Documentación paquete pySerial perteneciente a Python*. URL: `https://pythonhosted.org/pyserial/`.

[19] Izan Segarra Górriz. *CORE IP DE PROCESADOR RISC-V EN SYSTEM VERILOG*. 2019.