DISEÑO DE UN PROCESADOR CON ARQUITECTURA RISC-V PARA APLICACIONES DE DOMÓTICA.

T.G No. 1931.

EDWIN JAVIER BARBOSA MOLINA.

TRABAJO DE GRADO PARA OPTAR POR EL TÍTULO DE INGENIERO ELECTRÓNICO.

DIRIGIDO POR:

ING. FRANCISCO FERNANDO VIVEROS MORENO.

ING. LUISA FERNANDA GARCÍA VARGAS. M.SC

ING. JOSÉ LUIS URIBE. M. SC

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
BOGOTÁ D.C.
2020
PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

RECTOR DE LA UNIVERSIDAD:	**JORGE HUMBERTO PELÁEZ PIEDRAHITA S. J**
DECANO FACULTAD INGENIERÍA:	**LOPE HUGO BARRERO SOLANO, SCD.**
DIRECTORA DE CARRERA:	**ALEJANDRA MARÍA GONZÁLEZ CORREAL, PH.D.**
DIRECTOR DE PROYECTO:	**FRANCISCO FERNANDO VIVEROS MORENO.**
CODIRECTOR DE PROYECTO:	**LUISA FERNANDA GARCÍA VARGAS. M.SC**
CODIRECTOR DE PROYECTO:	**JOSÉ LUIS URIBE. M. SC**

ARTICULO 23 DE LA RESOLUCIÓN No. 13 DE JUNIO DE 1946.

"La universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Solo velará porque no se publique nada contrario al dogma y la moral católica y porque los trabajos no contengan ataques o polémicas puramente personales. Antes bien, que se vea en ellos el anhelo de buscar la verdad y la justicia".

AGRADECIMIENTOS.

"Gracias a la universidad por haberme permitido realizar mi formación en ella, gracias a todas las personas que estuvieron a mi alrededor durante este proceso, en especial a mi familia quienes siempre me apoyaron. Gracias a mis directores de trabajo de grado que siempre estuvieron apoyándome en la realización de este proyecto"

TABLA DE CONTENIDO.

LISTA DE FIGURAS.

FIGURAS DE LOS ANEXOS.

LISTA DE TABLAS.

Tablas de los ANEXOS.

LISTA DE ANEXOS.

1 INTRODUCCIÓN.

En un mundo que avanza cada vez más hacia las aplicaciones *open source*¸ las cuales permiten obtener mayores opciones para realizar las modificaciones que el usuario considere necesarias,[1] se pueden realizar aplicaciones de *hardware* libre para diseñar procesadores a la medida. Esto con el objetivo de tener mejoras en la aplicación que se realice, las cuales pueden ser de rendimiento, de costos, de seguridad, de facilidad en el manejo y programación de la aplicación entre otros.

Además, el mundo avanza en búsqueda de facilitar las tareas que realiza el ser humano por medio de dispositivos inteligentes. Una de estas aplicaciones es la domótica, la cual es una aplicación que ayuda al control de los sensores y actuadores del hogar para realizar una correcta gestión de estos y de esta forma brindar al ser humano una mayor comodidad.

Basados en lo anterior, lo que se busca con este trabajo de grado es diseñar e implementar un procesador el cual sea capaz de manejar múltiples periféricos para una aplicación especifica de domótica. Esto para dar una alternativa distinta a los procesadores que actualmente se utilizan en domótica, procesadores que son genéricos y que en la mayoría de los casos no dan una opción de personalización amplia.[2]

En este trabajo tiene como objetivo: implementar un procesador basado en arquitectura RISC-V (Conjunto de instrucciones reducido) para una aplicación de domótica (se implementa hasta pruebas con periféricos en un ambiente controlado). Este procesador será específicamente diseñado para una aplicación de domótica (Control y manejo de sensores y actuadores), en el cual se eligen instrucciones que se utilicen de forma constante en domótica dejando de lado instrucciones que no son usadas, como lo son operaciones en punto flotante y que conllevan una mayor complejidad del procesador, mayor cantidad de compuertas y por consiguiente un mayor consumo de potencia.

Por otro lado, la arquitectura RISC-V es arquitectura de uso libre, que permite una amplia gama de posibilidades de diseño y elección de instrucciones. Es una arquitectura empleada por varias empresas con varios proyectos, lo que la proyecta en un gran futuro y con posibilidades de convertirse en una arquitectura de uso global. Algunas empresas que apoyan actualmente esta arquitectura son, Google, Microsemi, NVIDIA, University of California, entre otras.[3]

El presente documento de trabajo de grado se subdivide en ocho secciones. En la sección dos se explican los objetivos iniciales y el alcance final del proyecto. En la sección tres, correspondiente al marco teórico, se explican las características de un procesador, de la domótica y las características de la arquitectura RISC-V que debe cumplir el procesador para poder considerarlo perteneciente a esta arquitectura. En la sección cuatro, se realiza el diseño del procesador, en el cual se incluyen diagramas de bloques, [1]AHPL, esquemáticos, codificación entre otros. En la sección 5 se especifica los protocolos de pruebas que se realizaron para verificar el correcto funcionamiento del procesador, y el manejo adecuado de periféricos y se realiza el análisis de resultados de las pruebas realizadas. Posteriormente en la sección 6 se exponen las conclusiones y mejoras que se pueden realizar al procesador. En la sección 7 se encuentran las referencias utilizadas en el libro y por último en la sección 8 se presentan el listado de anexos el enlace a ONEDRIVE donde se presentan.

[1] AHPL: (*A hardware -Programming language*), Es un lenguaje propuesto por F. Hill y G. Peterson ,el cual permite describir el comportamiento de un sistema digital, mediante un escrito.

2 OBJETIVO DEL PROYECTO

2.1 OBJETIVO GENERAL.

- Implementar un procesador basado en arquitectura RISC-V (Conjunto de instrucciones reducido) para una aplicación de domótica.

2.2 OBJETIVOS ESPECÍFICOS.

- Determinar los requerimientos de las aplicaciones de domótica para el diseño del procesador.
- Determinar las características de la arquitectura RISC-V necesarias para el diseño del procesador.
- Diseñar el procesador basado en arquitectura RISC-V para domótica.
- Implementar el procesador RISC-V en una FPGA.
- Realizar un protocolo de pruebas el cual permita poner a prueba el procesador para un contexto de domótica.

2.3 ALCANCE FINAL.

Como alcance final del proyecto se desarrolló un procesador basado en la arquitectura RISC-V, el cual posee las instrucciones necesarias para realizar aplicaciones de domótica. Este procesador es implementado en *hardware* por medio de la FPGA DE2-115, por lo que los resultados son analizados por medio del analizador lógico. Para este análisis se realizaron pruebas pequeñas donde interactuaron múltiples instrucciones del procesador, adicional a esto se realizó una prueba con dos sensores y un actuador, en la cual se adicionaron los controladores de los periféricos para recibir los datos de un infrarrojo y un ultrasonido (periféricos mapeados en memoria). Para finalmente probar el funcionamiento de los periféricos por medio de interrupciones y procesar los datos que son mostrados en una pantalla LCD 16X2, integrada en la tarjeta de desarrollo.

3 MARCO TEÓRICO.

En este apartado se presentan de forma resumida los conceptos básicos del trabajo de grado.

3.1 UNIDAD CENTRAL DE PROCESAMIENTO (PROCESADOR).

Un procesador es un dispositivo que procesa datos y toma decisiones en un computador, o en un sistema digital de mayor complejidad, es el cerebro. El procesador es capaz de leer (*Fetch*), decodificar (*Decode*) y ejecutar (*Execute*), para esto lee instrucciones que se ubican en una memoria, decodifica (traduce) estas instrucciones, y para posteriormente las ejecuta. El procesador a su vez se compone de diferentes bloques, algunos son: ALU (unidad aritmética lógica, registros), unidad de control, registros, y buses de comunicación, entre otros.

ALU (unidad aritmética lógica): Este bloque se encarga de realizar todas las operaciones de procesamiento de datos, operaciones lógicas y aritméticas con los datos binarios que le suministran los diferentes bloques/ entradas, memorias (dispositivos E/S), Registro y unidad de control.[4]

Registros: Proporcionan almacenamiento interno al procesador.[4]

Unidad de control: La unidad de control se encarga de controlar el funcionamiento del procesador esto significa que este bloque es capaz de identificar las instrucciones y direccionar los buses para que estas se ejecuten de manera óptima.[4]

Buses de comunicación: Son mecanismos que sirven para interconectar (comunicar) la ALU, los registros y la unidad de control.[4]

3.2 DOMÓTICA.

Domótica proveniente de (*domus* (latín)) = casa y tica (griego) = automática). La domótica es el conjunto de sistemas que sirve para controlar los servicios de una vivienda, por ejemplo, seguridad, energéticos, confort, comunicaciones, entre otros, etc. Este control se realiza por medio de sensores y actuadores que están siendo controlados por una unidad de procesamiento central, esta unidad se encarga de recibir las medidas de los sensores y a partir de esta modificar el estado de los actuadores. Ejemplo un sensor detector de luz, y el actuador es controlar la iluminación del lugar.[5]

3.3 ARQUITECTURA RISC-V.

La arquitectura RISC-V (Conjunto de instrucciones reducido) es una arquitectura nacida en la Universidad de California en Berkeley, cuya consigna es ser de uso libre, libre de uso, modificación y comercialización, lo que permite reducir costos de diseño al sistema que se esté realizando. Adicional a esto al ser una arquitectura de dominio público, se logra que haya una versión base la cual no va a sufrir modificaciones con el tiempo debido a que ya ha llegado a una versión estable donde los cambios que quedan son mínimos. Adicional a esto las futuras instrucciones se agregan como nuevas extensiones independientes dando la libertad de elegir se utilizan. La arquitectura RISC-V posee 3 posibilidades de ancho de palabra (32,64 y 128), y múltiples extensiones de instrucciones a elegir como lo son: para las instrucciones tipo I para operaciones aritméticas de enteros entre un valor inmediato y un registro, instrucciones tipo M para multiplicación y división de números enteros, instrucciones tipo F para precisión simple de punto frotante entre otros extensiones de instrucciones. Adicionalmente posee múltiples registros de propósito general (dependiendo del tamaño de palabra y de la aplicación se pueden poner más o menos). Todo arquitectura RISC-V debe tener registros de propósito general debido a que estos permite que siempre estén implementadas la extensión de las operaciones entre registros identificadas por la letra R. esta extensión es la base de la arquitectura.

Los procesadores RISC-V se nombran identificando que extensiones utilizan y el número de bits de palabra que tiene ejemplo: RV32I, RV: RISC-V, 32:32 bits de ancho de palabra, I operaciones inmediatas. Cabe aclarar que toda arquitectura RISC-V debe tener implementada operaciones entre registros (identificados por medio de la letra R) por lo que para especificar el nombre de un procesador esta letra (R) se omite. Para mayor referencia de la arquitectura RISC-V consultar las referencias [3] (Página oficial del proyecto),[6],[7],[8],[9], diversos textos donde se explica el contenido de la arquitectura, instrucciones, modos de privilegio, (acceso al procesador) registros de estados, etc.

4 DESARROLLO.

4.1 DISEÑO.

El procesador diseñado tiene un bus de datos de 32 bits y un bus de direcciones de 13 bits. Con 32 registros de propósito general, de los cuales el primer registro siempre tendrá el valor de 0. Además, se implementaron algunos de los registros de control y estados de la arquitectura, específicamente los de control de interrupciones, entre los que se encuentran: registros para habilitar/inhabilitar, guardar dirección de inicio de interrupción, guardar el contador de programa en interrupciones, y guardar causa de fallos del sistema, entre otros. Esto registros tienen la posibilidad de ser leídos y escritos por el usuario.

También se implementaron, por ejemplo, las instrucciones Tipo R, las cuales sirven para realizar operaciones de lógica y aritmética entre los registros. Instrucciones Tipo I: para realizar operaciones lógicas y aritméticas entre un registro y un valor inmediato de 12 bits con extensión de signo, y operaciones de carga de datos desde memoria. Operaciones Tipo S para cargar datos a memoria. El detalle de todas las instrucciones implementadas se encuentra en el apartado 4.1.4. Siendo estas de números enteros por lo que la arquitectura del procesador se define con RV32I.

Las operaciones que se implementan para este procesador son las más utilizadas en aplicaciones de domótica, que se caracteriza por realizar comparación de los estados de los sensores y dependiendo de dicho estado activar un actuador.

4.1.1 DESCRIPCIÓN ENTRADAS/SALIDAS.

En este apartado se realiza la explicación general de las entradas y salidas del procesador.

En la Figura 1 se presentan las entradas generales del procesador.

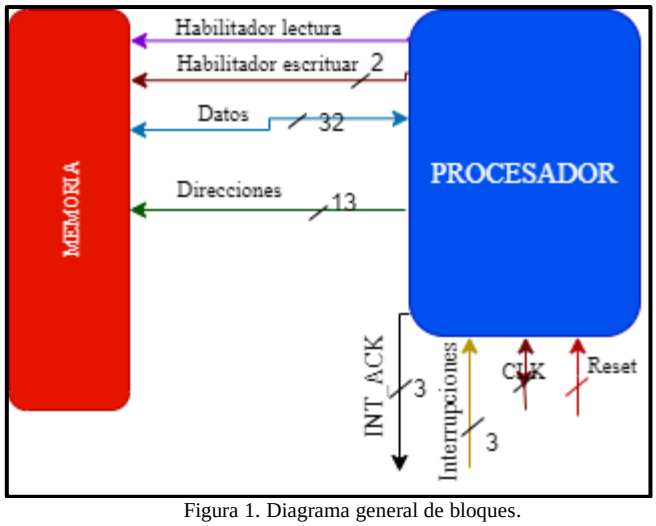


Figura 1. Diagrama general de bloques.

- Datos: Este bus es de 32 bits de ancho, sirve para realizar la comunicación entre memoria, periféricos y procesador, es un bus bidireccional. En este diseño específicamente se utilizan memorias RAM que esta embebidas en el chip, en la FPGA, estas memorias poseen un bit de lectura, un bit de escritura, un bus de direcciones, un bus de entrada de datos y un bus de salida de datos, por consiguiente, al tener dos buses de datos independientes no se maneja bus bidireccional para el bus de datos de la memoria.
- Direcciones: Este bus es de 13 bits con el cual se obtiene un máximo de tamaño de memoria de 8192(0-8191), para esta memoria se utilizará la memoria embebida en la FPGA. Es unidireccional, Tiene como destino todos los dispositivos externos como los son la memoria y los diferentes dispositivos de E/S (entrada salida).
- Habilitador de lectura. Esta señal es de un bit, indica que se está realizando una operación de lectura por parte del procesador.
- Habilitador de escritura. Esta señal es de dos bits, indica que se está realizando una operación de escritura por parte del procesador.
- Clk. Esta señal es de un bit. Esta señal es el reloj general del sistema, posee una frecuencia definida por la FPGA, para este trabajo se utilizará una tarjeta FPGA con frecuencia de 50 M Hz, con lo cual se obtiene un periodo de 20 ns.
- Reset. Esta señal es de un bit activa en alto, sirve para reiniciar el procesador externamente, es una señal asíncrona, lo que permite reiniciar en cualquier momento que el usuario lo desee.
- Interrupciones. Esta entrada externa al procesador sirve para informarle al procesador que hay un periférico que necesita atención, consta de una entrada de 3 bits. Las interrupciones se manejan activas en alto y decodificadas, por lo cual se obtienen hasta 7 niveles de interrupción.
- INT_ACK. Esta señal de salida sirve para indicarle a los periféricos que ya se esta ejecutando un nivel de interrupción, consta de 3 bits.

4.1.2 REGISTRO DE CONTROL Y ESTADOS.

Para definir los registros de estados que se van a utilizar es necesario, definir los modos de manejo que tiene la arquitectura.

Modos de privilegio

- Modo usuario: es el de menores privilegios, no controla interrupciones, y está limitado en funciones.
- Modo supervisor: da soporte a sistemas operativos.
- Modo máquina: permite acceso total al procesador, lo que permite habilitar y deshabilitar interrupciones entre otras funciones. Este modo esta implementado en todos los procesadores RISC-V y es en el que inicia el procesador después de un apagón o un reinicio.[7][6]

La codificación de los modos de usuario se muestra en la Tabla 1.

Nivel de acceso	código	Modo
0	00	Usuario
1	01	Supervisor
2	10	No aplica
3	11	Máquina.

Tabla 1. Codificación modos de usuario

Basado en los modos de privilegio se implementa únicamente el modo máquina debido a que permite controlar las interrupciones (activar, desactivar), que son de utilidad para aplicaciones de domótica donde un sensor debe ser capaz de ejecutar una rutina de tratamiento. Debido a esto no se implementa el modo usuario, y como el procesador no va a estar destinado a sistemas operativos el modo de supervisor se hace innecesario.

Dado que el modo que se va a implementar va a ser el de máquina, este se va a enfocar completamente a manejo de excepciones de interrupciones y algunas excepciones síncronas (Fallas en los códigos de operación). Por lo que permite agregar dos instrucciones más, las cuales son:

MRET: Sirve para retorna de una excepción (Interrupción) y WFI: Sirve para congelar el procesador a la espera de una interrupción, después de la interrupción regresa a la instrucción que sigue después de ejecutar WFI.

La arquitectura posee diferentes excepciones, de las cuales solo se implementaron las excepciones por instrucción ilegal y las excepciones por interrupción externa (Modo máquina)(Para consultar todas las posibles excepciones a implementar [6] y [9]).En este modo se define un banco de registros de control y estado el cual posee un bus de direcciones de 12 bits, con lo cual se puede direccionar 4096 posiciones de los registros, de las que solo se implementaron las que se mencionan a continuación.

MSTATUS: 32 bits. Tiene como dirección 0x300 (hexadecimal), es de lectura y escritura, posee los valores que se muestran en la Figura 2:

31		**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
SD		**WPRI**								TSR	TW	TVM	MXR	SUM	MPRV	XS[1]

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
XS[0]	FS[1:0]		MPP[1:0]		**WPRI**		SPP	MPIE	UBE	SPIE	**WPRI**	MIE	**WPRI**	SIE	**WPRI**

Figura 2. Registro de estado mstatus.

Los bits del registro son:

WPRI: Son campos reservados para futuras implementaciones de la arquitectura son ignorados o cableados a 0.

MIE Y SIE: Son habilitadores de interrupciones globales para modo máquina (MIE) y modo supervisor (SIE), en este diseño solo se utiliza MIE, el campo SIE es ignorado (No se implementa modo supervisor).

UBE: Este bit es cableado a 0 o ignorado debido a que este campo es para el modo usuario.

MPIE Y SPIE: Modo máquina (MPIE), modo supervisor (SPIE). Este valor mantiene guardado el valor de habilitación anterior a la excepción que haya ocurrido. (Solo se utiliza MPIE y guarda el valor anterior a MIE).

MPP Y SPP: Modo máquina (MPP), modo supervisor (SPP). Este valor mantiene guardado el valor del modo en el que se encontraba el procesador antes de la excepción que haya ocurrido, MPP es de dos bits debido a que puede ir a cualquiera de los otros modos de privilegios del procesador, SPP es de un bit debido a que solo puede ir a modo usuario y modo supervisor. Para este diseño el procesador se mantendrá siempre en modo máquina por consiguiente MPP=11 y SPP es conectado a 0.

FS Y XS: Se utiliza para guardar estados iniciales (Son de solo lectura) de punto flotante o de implementaciones especiales. En este caso toman valor de 0 debido a que no se implementa punto flotante ni implementaciones especiales que requieran XS.

MPRV: Esta señal esta cableada a 0 debido a que es para manejo de palabra de modo de usuario(*"endianness"*)

MXR: Sirve para la carga virtual de memoria en modo supervisor, este modo no se ejecuta por lo tanto se cablea a 0.

SUM: Sirve para habilitar el acceso de memoria de modo supervisor (Modifica el acceso de cómo el modo accede a la memoria virtual). Este modo no se ejecuta por lo tanto se cablea a 0.

TVM: Es un campo que sirve para la administración de memoria virtual en modo supervisor, este modo no se ejecuta por lo tanto se cablea a 0.

TW: Este campo sirve para detener la instrucción WFI (pausar el procesador a la espera de una interrupción, o por un tiempo predefinido), Como solo se ejecuta el modo m en el procesador este campo toma un valor de 0.

TSR: Esta instrucción sirve para excepciones en modo supervisor, como este modo no se implementa, la instrucción toma un valor de 0.

SD: Esta señal es dependiente de FS y XS. En este caso esta cableada a 0.

MTVEC: Ancho de 32 bits, dirección de acceso 0x305 (hexadecimal). Como se muestra en la Figura 3.

31-2	1-0
Valor Base	modo

Figura 3. Registro de estado mtvec.

Este registro contiene la dirección a la cual salta el procesador cuando ocurre una excepción, consta de dos campos el primer campo es el cambo de valor base este consta de 30 bits, este valor es la dirección de salto del procesador, tiene valor que es múltiplo de 4 (el procesador es direccionable por bytes se explica en la sección 4.1.5) adicional a esto tiene un segundo campo Modo el cual es para especificar el modo de operación de las interrupciones. Este dato lo puede cambiar el usuario dependiendo de sus necesidades.

Si el valor de modo es ubicado en cero el contador del programa toma el valor base si modo es ubicado en uno, el contador del programa toma base+4*causa de excepción; si el valor de modo es igual a 2 no se realiza nada. Por consiguiente, para el caso de interrupciones de nivel 1, si el modo se ubica en uno el contador del programa toma el valor de: base + (4*11) = base +44. (se explica mejor en la sección 4.1.5), si lo que ocurre es una excepción síncrona (instrucción ilegal), siempre se salta al valor de base.

MEPC: Ancho de 32 bits, lectura y escritura. Dirección de acceso 0x341(Hexadecimal). Como se muestra en la Figura 4.

31-0
MEPC

Figura 4. Registro de estado mepc.

Este registro sirve para guardar la dirección (contador de programa a la cual se debe regresar) cuando ocurre una interrupción.

MCAUSE: Ancho de 32 bits, lectura y escritura. Dirección de acceso 0x342(Hexadecimal). Como se muestra en la Figura 5.

31	30-0
Interrupción	Código de excepción

Figura 5. Registro de estado mcause.

INTERRUPCIÓN: Es un bit indica cuando la excepción fue causada por una excepción externa(interrupción).

CÓDIGO DE EXCEPCIÓN: Este valor indica exactamente qué ha ocurrido el sistema, para este diseño se utilizan los códigos que son mostrados en la Tabla 2, pero hay más códigos para los diferentes modos de privilegio que se implemente y/o excepciones. (para consultar los posibles códigos de excepción ver referencia[9]). Adicional a estos códigos como se manejan múltiples interrupciones decodificadas, a estas se les asignan un código de excepción especial, definido por el diseñador.

INTERRUPCIÓN	CÓDIGO EXCEPCIÓN	DESCRIPCIÓN.
1	40	Interrupción externa de nivel 7. (Modo máquina)
1	35	Interrupción externa de nivel 6. (Modo máquina)
1	30	Interrupción externa de nivel 5. (Modo máquina)
1	25	Interrupción externa de nivel 4. (Modo máquina)
1	20	Interrupción externa de nivel 3. (Modo máquina)
1	15	Interrupción externa de nivel 2. (Modo máquina)
1	11	Interrupción externa de nivel 1. (Modo máquina)
0	2	Instrucción ilegal (Códigos de operación, funct3 o funct7 no validos)

Tabla 2. Códigos de excepción.

MTVAL: Ancho de 32 bits, dirección de 0x343(hexadecimal). Como se muestra en la Figura 6.

31-0
Mtval

Figura 6. Registro de estado mtval.

Este registro sirve para guardar la instrucción que causo un fallo de instrucción ilegal para que el usuario la pueda conocer y modificar.

MIE Y MIP: Ancho de 32 bits cada uno, direcciones 0x304(hexadecimal) y 0x344(hexadecimal respectivamente). Como se muestra en la Figura 7.

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEIE	MEIE	0				MEIE	0	SEIE	0	MTIE	0	STIE	0	MESIE	0	SSIE	0

Figura 7. Registro de estado mie y mip.

Este registro toma los valores mostrados en la Figura 7. Los valores del 16 al 31 son de uso libre o conectados con 0 si no se utilizan, en este procesador se implementan dos bits más (16 y 17) para control y manejo de múltiples interrupciones. MIE es el habilitador de todas las interrupciones. MIP es el que indica cuando hay interrupciones pendientes dependiendo de las posibles interrupciones. Todos los que inician por M indican que son de modo máquina, Los que inician por S indican que son de modo supervisor (Este no se implementa en este diseño). En el registro MIP, se utiliza el bit en posición 31 para indicar que ocurrió un error de excepción síncrona, falla de código, este cambio lo realiza automáticamente el procesador.

MEIE Y SEIE: Manejo de excepciones externas(interrupciones) (habilitación y pendientes).

MTIE Y STIE: Manejo de excepciones de tiempo(interrupciones) (habilitación y pendientes), no implementadas.

MESIE Y SSIE Manejo de excepciones de software(interrupciones) (habilitación y pendientes), no implementadas.

MSCRATCH: 32 bits, dirección 0x340(hexadecimal). Como se muestra en la Figura 8.

31-0
Mscratch

Figura 8. Registro de estado mscratch.

Este registro contiene un puntero a memoria que se utiliza para guardar registros de excepciones, es dado por el usuario, y se intercambia por un registro (Registro 2 (Sp)) para realizar el manejador de interrupciones, se utiliza para guardar el punto de inicio del puntero de pila (stack pointer), en interrupciones o excepciones.

MSTATUSH: 32 bits de ancho, dirección de acceso 0x310(hexadecimal, solo lectura). Como se muestra en la Figura 9.

31-6	**5**	**4**	**3-0**
WPRI	MBE	SBE	**WPRI**

Figura 9. Registro de estado mstatush.

Adicional al mstatus está el mstatush, el cual indica en qué tipo de formato se reciben los datos *Big-endian* (bits más significativos en la parte más alta de la memoria) *o Little-endian* (Bit más significativos en la parte más baja de la memoria). Las instrucciones siempre se envían igual sin importar el formato del dato. (Para más información consultar [9])

MBE Y SBE: Modo máquina (MBE), modo supervisor (SBE). Estos bits indican en qué tipo de *endiannes* está operando el procesador. SBE es cableado a 0 debido a que no se implementa este modo, MBE, es configura como 1 para Big-endain o 0 para Little-edian. En este caso permanece en 0. Como este registro permanece en este caso con valor de 0 no se hace necesaria su implementación.

STACK-POINTER DEL PROCESADOR: 32 bits, dirección 0x380(hexadecimal). Como se muestra en la Figura 10.

31-0
¨STACKPOINTER¨

Figura 10. STACK-POINTER.

Este registro contiene un puntero a memoria que se utiliza para guardar los registros de estados MCAUSE, MIE, MEPC Y MTVAL, cuando ocurre una interrupción, este registro se hace necesario debido a que se necesitan guardar los registros de estados para que cuando llegue una segunda interrupción no se pierdan estos registros y el procesador sea capaz de retornar entre las diferentes interrupciones. Este registro siempre se inicializa con todos sus bits en 1.

4.1.3 DIAGRAMA DE BLOQUES.

En la Figura 11 se encuentra el diagrama de bloques del procesador (para ver en mejor resolución ir al ANEXO diagrama de bloques). A continuación, se presenta una descripción de cada uno de los bloques, su interacción y las señales que posee. Él procesador se encuentra encerrado en un bloque gris claro, con bordes en negro y externamente se muestra la memoria.

Descripción de los bloques y sus señales:

MEMORIA: Este dispositivo externo al procesador, representa la memoria total del sistema (fuera del procesador), en esta se guardarán las instrucciones que el procesador va a ejecutar, así como datos de resultados de estas operaciones. La memoria que se va a utilizar consta de un bus de direcciones de 13 bits, esto permite almacenar hasta 8192 (0-8191, 32764 bytes) datos, además posee un bus de datos de 32 bits, esto permite manejar valores desde −2147483648 hasta 2147483647 con signo y sin signo desde 0 hasta 4294967295, esta memoria es direccionable por bytes. Las direcciones deben estar alineadas en múltiplos de 4 para leer una instrucción de manera correcta, si la dirección esta desalineada, se va a aproximar a la dirección inferior.

Señales de interacción de la memoria.

Direcciones: Este bus de datos es de 13 bits, es de entrada. Sirve para indicar que posición de memoria se va a leer, también sirve para identificar con que periférico el procesador se desea comunicar. Es síncrono.
Datos: Este bus de direcciones sirve para cargar datos al procesador, o que el procesador le cargue datos a la memoria o a los distintos periféricos que este posea, es de 32 bits. Para esta memoria se utiliza una memoria integrada de la FPGA (memoria en el chip), está a diferencia del esquemático posee un bus de datos de salida de la memoria y otro bus de datos de entrada.
EnMLEC: Este es un bit, asíncrono, de salida del procesador. Sirve para habilitar la lectura de la memoria.
EnMESC: Este es dos bits, asíncrono, de salida del procesador. Sirve para habilitar la Escritura de la memoria, habilita por byte, dos bytes, o cuatro bytes.
CLK: Reloj del sistema, es un bit de entrada. Se omite escribirlo en el diagrama de bloques.

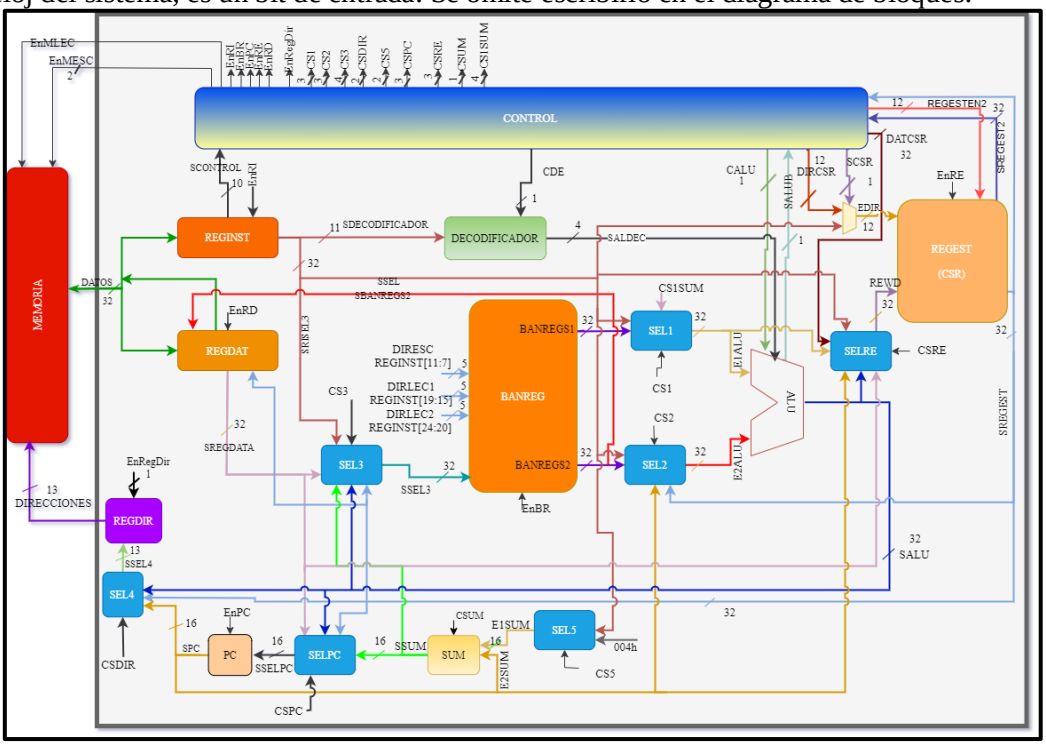


Figura 11. Diagrama de bloques.

REGISTRO DE INSTRUCCIONES (REGINST):
Este bloque se encarga de guardar la instrucción (de forma sincrónica) que se está ejecutando, es un único registro y posee un ancho de palabra de 32 bits, en este se encuentra la información de las operaciones, sus operandos, tipo de formato de instrucción, entre otros.

Señales de interacción del registro de instrucciones.
DATOS: Este bus es de 32 bits, viene de la memoria y sirve para traer las instrucciones desde la memoria, hasta el procesador, se sincroniza con el procesador para el guardado del mismo.
SCONTROL: Esta señal se envía a control, es de 11 bits, consta de tres partes del registro de instrucciones, la primera es el código de operación (Opcode, se explica en la sección 4.1.4) ubicado de las posiciones 6 a la 0(REGINST[6:0]) , la segunda parte es el funct3(se explica en la sección 4.1.4) ubicado en las posiciones 14 a la 12 (REGINST[14 :12]), la tercera partes es un bit que sirve para identificar las operaciones de espera de interrupción (WFI), y fin de interrupción, REGINST[29].
EnRI: Esta señal es de un bit, sirve para habilitar la escritura del registro, esta señal es enviada por control, es síncrona.
CLK: Reloj del sistema, es un bit de entrada. Se omite escribirlo en el diagrama de bloques.
SDECODIFICADOR: Esta señal es enviada al decodificador. Las partes del Registro de instrucciones que se envían son: REGINST([6:0] ,[14:12],[30]) que corresponden a El código de operación (se explica en la sección 4.1.4), el funct3 (se explica en la sección 4.1.4) y adicional mente se envía un bit del func7(se explica en la sección 4.1.4),este se ubica en la posición REGINST[30], en total esta señal posee 11 bits de tamaño y es asíncrona.
SRISEL3: Esta señal es utilizada para la operación LUI (se explica en la sección 4.1.4), en la cual el dato de entrada (valor inmediato, (se explica en la sección 4.1.4)), es puesto en los 20 bits más significativos y guardado en el banco de registros. Posee un tamaño de 32 bits.
SSEL: Salida a selector 1 (SEL1), a selector2 (SEL2) y a selector del registro de estados (SELRE): Esta señal se envía a los selectores de la ALU y del registro de estados, consta de todo el registro de instrucciones debido a las diferentes ubicaciones del inmediato según el formato de la instrucción para la ALU (se explica en la sección 4.1.4) y se necesita de forma completa para algunas funciones del registro de estado. Esta señal es síncrona de 32 bits.

REGDAT: REGISTRO DE DATOS:
Este bloque se encarga de guardar los datos (de forma sincrónica) que intercomunican el procesador con los [2]dispositivos externos, y viceversa, es un único registro y posee un ancho de palabra de 32 bits. En diagrama de bloques se omite un multiplexor que se necesita para la identificación del dato que se desea guardar en el registro.

Señales de interacción del registro de datos.
DATOS: Este bus es de 32 bits, contiene la información que se necesita almacenar en el registro de datos, y sirve para realizar los diferentes intercambios de información entre los dispositivos externos y la memoria, se sincroniza con el procesador para el guardado del mismo.
SBANREGS2: Esta señal sirve para guardar un dato del banco de registro en las operaciones que lo requiera. Es de 32 bits.
SREGEST: Esta señal sirve para almacenar un dato del banco de registros de control y estado, es de 32 bits.
ENRD: Esta señal es el habilitador del registro, proviene de control es de un bit y sirve para habilitar los momentos en los cuales se debe dejar escribir el registro.
CLK: Reloj del sistema, es un bit de entrada. Se omite escribirlo en el diagrama de bloques.
SREGDATA: Esta señal contiene el valor almacenado en el registro de datos, tiene como destino el selector 3, selector de registro de estado y el selector del contador del programa, dependiendo de la instrucción se define su destino final. Esta salida es de 32 bits, asíncrona.

[2] Entiéndase como dispositivo externo a cualquiera periférico que necesite realizar un intercambio de información con el procesador, estos son memorias, dispositivos de entrada y salida, periféricos en general.

REGISTRO DE DIRECCIONES (REGDIR):
Este bloque se encarga de guardar la dirección (de forma sincrónica) que se envía a la memoria o al periférico para ser leída posteriormente.

Señales de interacción del registro de direcciones.
SSEL4: Esta señal es de 13 bits de ancho, contiene la dirección próxima a ser guardada en el registro de direcciones.
ENREGDIR: Esta señal es el habilitador del registro, proviene de control es de un bit y sirve para habilitar los momentos en los cuales se debe dejar escribir el registro.
CLK: Reloj del sistema, es un bit de entrada. Se omite escribirlo en el diagrama de bloques.
DIRECCIONES: Esta señal con destino final la memoria, sirve para enviar la dirección de lectura o escritura de la memoria, tiene un ancho de 13 bits.

REGISTRO DEL CONTADOR DE PROGRAMA (PC):
Este registro sirve para guardar la dirección de la instrucción en la cual se está operando, es de 16 bits.

Señales de interacción del registro del contador de programa.
SSELPC: Esta señal es el valor que debe guardar el registro del contador de programa, es de 16 bits, síncrona.
EnPC: Esta señal es de un bit sirve para habilitar la escritura del contador del programa, viene de control.
SPC: Esta señal tiene el valor guardado en el contador de programa tiene como destino el registro de direcciones y el registro de control y estados, según la instrucción, es de 16 bits, asíncrona.

REGISTROS DE PROPÓSITO GENERAL O BANCO DE REGISTROS (BANREG):

Este bloque propio de la arquitectura consta de 32 registros de propósito general (para procesadores con otros requerimientos se pueden agregar más registros (caso de punto flotante)), todos los registros de la arquitectura tienen un ancho de palabra igual a la del procesador, en este caso 32 bits. De los 32 registros que ofrece la arquitectura 31 son utilizables para cualquier propósito estos son los registros del 1 al 31, el registro 0 siempre guarda el valor de 0 (Nunca se puede modificar), el registro del contador del programa y otros que se requieran para el funcionamiento no son tenidos en cuenta en los 32 registros de propósito general. [7]

Aunque no hay nombramiento oficial de los registros hay una convención llamada “estándar RISC-V”, que da nombre/usos a los diferentes registros (ver Tabla 3), y especifica como los compiladores deben utilizar los registros (La idea de esto es que la mayoría de los compiladores utilicen esta convención para hacerlos de uso genérico)[7]. (La referencia principal es la mencionada al final del texto para complementarla se pueden leer las referencias [6]y[8]).

Registros	Nombre	Función
0	x0/cero	Cero
1	x1/ra	Dirección de retorno
2	x2/sp[3]	Stack pointer
3	x3/gp	Global pointer
4	x4/tp	Thread pointer
5	x5/t0	Temporal
6	x6/t1	Temporal
7	x7/t2	Temporal
8	x8/s0/fp	Registro de guardado ,frame pointer
9	x9/s1	Registro de guardado
10	x10/a0	Argumento de función, valor retorno
11	x11/a1	Argumento de función, valor retorno
12	x12/a2	Argumento función
13	x13/a3	Argumento función
14	x14/a4	Argumento función
15	x15/a5	Argumento función
16	x16/a6	Argumento función
17	x17/a7	Argumento función
18	x18/s2	Registro de guardado
19	x19/s3	Registro de guardado
20	x20/s4	Registro de guardado
21	X21/s5	Registro de guardado
22	X22/s6	Registro de guardado
23	X23/s7	Registro de guardado
24	X24/s8	Registro de guardado
25	X25/s9	Registro de guardado
26	X26/s10	Registro de guardado
27	X27/s11	Registro de guardado
28	X28/t3	Temporal
29	X29/t4	Temporal
30	X30/t5	Temporal
31	X31/t6	Temporal

Tabla 3. Registros, nombre y función.

** Tabla basada en la figura 2.4 de la referencia [6]

Registros de guardado: Registros que preferiblemente no deben cambiar cuando se llama una interrupción o subrutina, esto se hace para hacer más rápidas las interrupciones.

Temporal. Estos registros son utilizados en los llamados de interrupciones y subrutinas para realizar las diferentes operaciones que se necesiten realizar en estos programas.

Argumento de función y retorno de función: Estos registros son los que almacenan los datos que se necesitan utilizar en una subrutina o que provienen de una subrutina para su utilización en el programa principal.

Dirección de retorno: Registro que almacena el contador de programa siguiente a la instrucción en la que se llamó a una subrutina. Para posteriormente regresar a esa dirección.

[3] Debido a la naturaleza del procesador este posee un registro de estados de stack-pointer (del procesador), el cual se utiliza para manejar las interrupciones múltiples y un registro de stack-pointer en el banco de registro (x2/sp) que sirve para realizar el manejador de interrupciones. (guardar datos del banco de registros en la memoria)

Stack,global y thread pointer: Stack pointer es un puntero superior de pila, global pointer, es un puntero a un espacio a memoria donde se almacenan todas las variables globales de los diferentes programas que se están ejecutando. *thread pointer,* este puntero se utiliza cuando se manejan multiprocesadores (Multi-nucleo, Multi-hilos de ejecución), en cada núcleo o hilo de ejecución tiene diferente valor y sirve para indicar una posición de memoria donde el núcleo o hilo de ejecución puede guardar sus propias variables sin estarlas compartiendo con los otros núcleos o hilos de ejecución.

En la Tabla 3 se muestran los registros y su posible utilización en lenguaje ensamblador, se muestran los registros que se preservan cuando ocurre una excepción o llamado a subrutina, los registros de guardado temporal, los punteros de stack, argumentos de función entre otros, todos estos registros son en base a la convención estandarizada de RISC-V, pero se recuerda que no son estrictamente obligatorios.

Señales de interacción del banco de registros.
DIRLEC1: Esta señal sirve para enviar la dirección del primer registro a leer del banco de registro. En todos los formatos de instrucción que se necesite leer uno o más registros, la dirección del registro uno siempre se ubica en los mismos bits, estos son en las posiciones REGINST[19:15].
DIRLEC2: Esta señal sirve para enviar la dirección del segundo registro a leer del banco de registro. En todos los formatos de instrucción que se necesite leer dos registros, la dirección del registro numero dos siempre se ubicara en los mismos bits, estos son en las posiciones REGINST[24:20].
DIRESC: Esta señal sirve para enviar la dirección del registro a escribir del banco de registro. En todos los formatos de instrucción que se necesite escribir datos en el banco de registros, la dirección del registro de escritura siempre se ubicara en los mismos bits, estos son en las posiciones REGINST[11:7].
SSEL3: Esta señal contiene el dato que se desea guardar en el banco de registros, proviene de un selector que es el que se encarga de monitorear en que momentos se deja pasar alguna señal para ser guardada., es de 32 bits de ancho.
ENBR: Esta señal es de un bit, es controlada por control y sirve para habilitar la escritura del banco de registros.
CLK: Reloj del sistema, es un bit de entrada. Se omite escribirlo en el diagrama de bloques.
BANREGS1 Y BANREGS2: Esta señal es de 32 bits, es el dato del primer/segundo registro leído, tiene como destino los selectores que las envían ya sea a la ALU o al registro de control y estados.

REGEST: REGISTROS DE CONTROL Y ESTADO(CSR).

Este bloque se encarga de guardar los diferentes registros de control y estado, registros que contienen habilitadores de interrupciones, guardar el contador de programa, entre otras funciones. El ancho de palabra de esos registros es de 32 bits y tiene un bus de direcciones de 12 bits. (definido por la arquitectura) (en la sección 4.1.2 se explica todo lo relacionado a los registros de control y estado CSR).

Señales de interacción de REGEST:
REWD (ENTRADA DE DATOS): Esta señal de 32 bits es el dato que se va a guardar en el registro de estados.
EDIR Y REGESTN2 (ENTRADA DE DIRECCIONES): Estas señales son de 12 bits permite direccionar hasta 4195 registros de control y estados (para este procesador no se utilizan todas las posiciones solo las necesarias para control y manejo de interrupciones (ver sección 4.1.2)).
CLK: Reloj del sistema, es un bit de entrada. Se omite escribirlo en el diagrama de bloques.
EnRE: Esta señal es de un bit, es controlada por control y sirve para habilitar la escritura del registro de estados.
SREGEST Y SREGEST2: Esta señal de 32 bits, contiene el dato proveniente del registro de estados. De la dirección 1 y 2 respectivamente.

ALU:

Este bloque se encarga de realizar diferentes operaciones lógicas, aritméticas, corrimientos y condicionales, tiene dos entradas de datos de 32 bits cada una, adicional a estas tiene dos entradas de manejo, una le indica que operación debe realizar, y la segunda funciona como habilitador. Posee dos salidas de datos una de 32 bits y una bandera de condiciones de un bit.

Señales de interacción de la ALU.
E1ALU: Esta señal es el primer operando de la a ALU, es de 32bits de ancho.
E2ALU: Esta señal es el segundo operando de la ALU, es de 32bits de ancho.
CALU: Esta señal funciona como habilitador a la ALU, cuando esta activa la ALU opera de lo contrario no realiza ninguna operación.
SALDEC: Esta señal indica que operación debe realizar la ALU (Aritmética, lógica, condicional, etc.), es de 4 bits de ancho, más adelante en la descripción del decodificador se encuentra la tabla resumen de salidas y operaciones.
SALUB: Esta señal contiene una bandera que sirve para los condicionales de salto, indica en qué momento se debe realizar un salto dependiendo de la condición de entrada (menor, mayor o igual, igual, etc.).
SALU: Esta señal contiene el resultado de las operaciones aritméticas, lógicas, condicionales, corrimientos, etc. Es de 32 bits, pero se subdivide en tamaño dependiendo del destino.

- SALIDA A SELECTOR 3 (SEL3): Esta señal tiene como destino final el banco de registros, su tamaño final es de 32 bits.
- SALIDA A SELECTOR 4 (SEL4): Esta señal tiene como destino final el bus de direcciones, por lo tanto, solo se usan los primeros 16 bits para este propósito.
- SALIDA A SELECTOR DE CONTADOR DEL PROGRAMA (SELPC): Esta señal tiene como destino final el registro del contador del programa (PC), por lo tanto, solo se usan los primeros 16 bits para este propósito. Este bus es usado en instrucciones como lo son JALR, donde se suma un registro más un inmediato y se modifica el PC.
- SALIDA A SELRE: Esta señal es utilizada para modificar los registros de control y estado, cuando se modifican por una operación procedente de la ALU.

CONTROL:

Este bloque es el encargado de manejar todo el sistema, contiene la máquina de estados del sistema, por medio de estas se sabe en qué momentos se deben habilitar/deshabilitar, leer, escribir los registros, los datos, la memoria; define que operaciones va a realizar, que estados debe seguir para realizar determinada operación con éxito, etc.

Señales de interacción del control.
ENTRADA DE REGISTRO DE INSTRUCCIONES (SCONTROL): Esta señal se envía a control para definir qué operación se va a realizar, es de 10 bits. (Previamente explicada)
ENTRADA DE ALU (SALUB): Esta bandera sirve para las condiciones de salto (previamente explicada).
SREGEST: Esta señal es de 32 bits de ancho, contiene el valor del registro de estados que se está leyendo, sirve para identificar que hay en el registro de estados, por ejemplo, si las interrupciones han sido desactivadas.
SALIDAS DE HABILITADORES: Estas señales sirven para habilitar la escritura y funcionamiento de los distintos bloques del sistema. A continuación, se listan los distintos habilitadores.

- EnMESC : Habilitar escritura en memoria, dos bits.
- ENMLEC: Habilitar lectura de memoria, un bit.
- EnRI: Habilitar escritura del registro de instrucciones, un bit.
- EnBR: Habilitar Escritura del banco de registros, un bit.
- EnPC: Habilitar Escritura del contador de programa, un bit.
- EnRE: Habilitar escritura del registro de control y estado, un bit.
- EnRegDir: Habilitar escritura del registro de direcciones.
- EnRE: Habilitar escritura del registro de control y estado, un bit.

- EnRD: Habilitar escritura del registro de datos, un bit.
- CDE: Habilitador de decodificación, un bit.
- CALU: Habilitador para realizar operaciones, un bit.

SALIDAS DE CONTROL (SELECTORES): Estas señales sirven para identificar que dato y en qué orden de bits, se debe dejar pasar determinado dato (dejar pasar un registro o un valor inmediato, extendido en signo, lo organiza porque el dato viene en diferentes partes, etc.). A continuación, se listan los distintos controladores.

- CS1: Controla que señal deja pasar en el selector 1, consta de 3 bits.
- CS1SUM: Esta señal ayuda a definir un valor constante para sumar al registro MTVEC, se utiliza para interrupciones vectorizadas. Y para stack-pointer del procesador.
- CS2: Controla que señal deja pasar en el selector 2, consta de 3 bits.
- CS3: Controla que señal deja pasar en el selector 3, consta de 4 bits.
- CSDIR: Controla que señal deja pasar en el selector 4, consta de 2 bits.
- CS5: Controla que señal deja pasar en el selector 5, consta de 2 bits.
- CSPC: Controla que señal deja pasar en el selector del contador de programa, consta de 3 bits.
- CSRE: Controla que señal deja pasar en el selector de registro de estado, consta de 3 bits.
- SCSR: Controla que señal deja pasar en el selector de direcciones del registro de estados (este funciona como un multiplexor), consta de 1 bits.

SALIDA DE DIRECCIONES PARA EL REGISTRO DE ESTADOS (DATCSR): Esta señal consta de 12 bits sirve para enviar una determinada dirección al registro de estados, esta dirección ya sea para leer o para escribir. Es usada por ejemplo para leer el registro de mcause, el cual indica que ha ocurrido una excepción,
SALIDA DE DATOS PARA EL REGISTRO DE ESTADOS (DATCSR): Esta señal consta de 32 bits sirve para enviar un determinado dato al registro de estados. Por ejemplo, para modificar la dirección de salto de interrupciones. (registro MTVEC)

DECODIFICADOR:

Este bloque recibe el código de operación, el funct3 y un bit del funct7 (se explican en la sección 4.1.4) y a partir de estos datos define qué operación va a realizar la ALU (Aritmética, lógica, condicional, etc.).

Señales de interacción del decodificador.
SDECODIFICADOR: Esta señal contiene el código de operación el func3 y el funct 7 (se explican en la sección 4.1.4) necesarios para identificar la operación deseada
ENTRADA DE CONTROL (CDE): Esta señal es de un bit. Viene de control y sirve para habilitar la decodificación.
SALIDA A ALU: Esta señal tiene como finalidad indicarle a la ALU que operación se debe realizar, es de 4 bits. A continuación, se lista la tabla de entrada y salida del decodificador.

DECODIFICADOR-ALU				
FUNC7*, **	**FUNC3***	**OPCODE***	**SALIDA DECODIFICADOR**	**OPERACIÓN**
0000000-No aplica-No aplica-No aplica-No aplica-No aplica	000-000-(000-001-010-100-101)-(000-001-010)-000-No aplica	0110011-0010011-0000011-0100011-1100111-0010111	0000	Suma
0100000	000	0110011	0001	Resta
0000000-No aplica	001-001	0110011-0010011	0010	Corrimiento lógico a la izquierda
0000000-No aplica	010-010	0110011-0010011	0011	Comparación menor (con signo)(32 bits)*

0000000-No aplica	011-011	0110011-0010011	0100	Comparación menor (sin signo)(32 bits)*
0000000-No aplica	100-100	0110011-0010011	0101	XOR
0000000-0000000	101-101	0110011-0010011	0110	Corrimiento lógico a la derecha
0100000-0100000	101-101	0110011-0010011	0111	Corrimiento aritmético a la derecha
0000000-No aplica-No aplica	110-110-(010-110)	0110011-0010011-1110011	1000	Or
0000000-No aplica-No aplica	111-111-(011-111)	0110011-0010011-1110011	1001	And
No aplica	000	1100011	1010	Igual
No aplica	001	1100011	1011	Diferente
No aplica	100	1100011	1100	Comparación menor (con signo)(1 bits)*
No aplica	101	1100011	1101	Mayor o igual con signo
No aplica	110	1100011	1110	Comparación menor (sin signo)(1 bits)*
No aplica	111	1100011	1111	Mayor o igual sin signo

Tabla 4. Decodificación de la ALU código de operación- operación.

* Se explica en la sección 4.1.4,** Funct7 Completo, para el decodificador solo se utiliza el 2 bit más significativo.

SUMADOR DEL CONTADOR DE PROGRAMA. (SUM):

Este bloque se encargar de realizar el conteo del contador del programa, es un bloque sumador, el cual es asíncrono, realiza sumas de máximo 8192 como resultado (número máximo de direcciones).

Señales de interacción del registro del contador de programa.
E1SUM: Esta señal es de 16 bits, asíncrona, tiene dos funciones habilitar un 4 para aumentar el contador del programa (contador del programa direccionado por bytes), y habilitar un valor inmediato para sumarle al contador del programa.
E2SUM: Esta señal es de 16 bits, asíncrona. Siempre está conectada a la salida del contador del programa.
SSUM: Esta señal es de 16 bits, asíncrona. Contiene el valor de la suma del contador del programa con un 4 o un valor inmediato para almacenar el valor siguiente del contador de programa,
CSUM:
Adicional a estas señales el bloque sum posee una señal extra que sirve para realizar la resta de 4 a los datos, esto se realiza para los saltos condicionales y la instrucción JALR (salto y registro), en los cuales las operaciones se realizan con respecto a la posición actual. Como el contador de programa almacena la posición siguiente, por eso se hace necesario esta señal.

SELECTOR 1 (SEL1):
Ese bloque sirve para seleccionar los datos que tiene como destino E1ALU, y SELRE, además de seleccionar los datos también sirve como extensor de signo dependiendo de los diferentes tipos de formato (se explican en la sección 4.1.4).

Señales de interacción del selector 1.
BANREGS1: Esta señal es un bus de datos proveniente del banco de registros, dependiendo de la operación se deja pasar o se niega y se deja pasar, o no se deja pasar.
ENTRADA DE REGISTRO DE INSTRUCCIONES (SSEL): Esta señal es un bus de 32 bits provenientes del registro de instrucciones, dependiendo de la operación se dejan pasar y sele extiende el signo, o se extiende con ceros, dependiendo del destino.
ENTRADA DE CONTROL (CS1): Esta señal controla en que forma y que dato se deja pasar determinado dato, es de 2 bits.
ENTRADA DE CONTROL (CS1SUM): Esta señal controla que constante se deja pasar hacia la ALU, estas constantes corresponden a la causa de la interrupción multiplicada por cuatro.
E1ALU: Bus de datos de tamaño 32 bit, con destino a ALU y selector del registro de estados, es el primer operador de la ALU.

SELECTOR 2(SEL2):
Ese bloque sirve para seleccionar los datos que tiene como destino E2ALU, además de seleccionar los datos también sirve como extensor de signo dependiendo de los diferentes tipos de formato (se explican en la sección 4.1.4).

Señales de interacción del selector 2.
BANREGS2: Esta señal es un bus de datos proveniente del banco de registros.
ENTRADA DE REGISTRO DE INSTRUCCIONES (SSEL): Esta señal es un bus de 32 bits provenientes del registro de instrucciones.
ENTRADA DE REGISTRO DE ESTADOS (REGEST): Esta señal es un bus de 32 bits provenientes del registro de estados, se utiliza para las instrucciones que modifican el registro de estados realizando un or o and entre el valor actual y un valor de ingreso.
SPC: Esta señal es un bus de 16 bits provenientes del registro del contador de programa, a la llegada a este selector se le resta un valor de 4 debido a que esta se utiliza para las operaciones suma al contador del programa (AUIPC)y estos se realizan con respecto a la posición actual, y el contador de programa almacena la posición siguiente.
ENTRADA DE CONTROL (CS2): Esta señal controla en que forma y que dato se deja pasar determinado dato, es de 3 bits
E2ALU: Bus de datos de tamaño 32 bit, con destino a ALU, es el segundo operador de la ALU.

SELECTOR 3 (SEL3):

Ese bloque sirve para seleccionar los datos que tiene como destino el banco de registros, además de seleccionar los datos también sirve como extensor de signo dependiendo de los diferentes tipos de formato (se explican en la sección 4.1.4).

Señales de interacción del selector 3.
ENTRADA REGISTRO DE INSTRUCCIONES (SRISEL3): Esta señal es utilizada para la operación LUI (se explica en la sección 4.1.4), en la cual el dato de entrada (valor inmediato, (se explica en la sección 4.1.4)), es puesto en los 20 bits más significativos y guardado en el banco de registros. Posee un tamaño de 32 bits.
SREGDATA: Esta señal contiene el valor almacenado en el registro de datos.
SSUM: Esta señal sirve para preservar almacenar el valor del contador de programa más cuatro o un valor inmediato dependiendo de la instrucción.
ENTRADA DE ALU (SALU): Esta señal tiene como destino final el banco de registros, su tamaño final es de 32 bits.

ENTRADA DE REGISTRO DE ESTADOS (SREGEST): Esta señal sirve para guardar el valor actual del registro de control y estados en el banco de registros.
CS3: Esta señal controla en que forma y que dato se deja pasar determinado dato, es de 3 bits.
REWD. Este es bus el cual contiene el dato que se desea dejar pasar para guardar en el banco de registros.

SELECTOR 4 (SEL4):

Ese bloque sirve para seleccionar los datos que tiene como destino el bus de direcciones de la memoria.

Señales de interacción del selector 4.
ENTRADA DE ALU (SALU): Esta señal tiene como destino final el bus de direcciones de la memoria, por lo tanto, solo se usan los primeros 16 bits para este propósito.
ENTRADA DE CONTADOR DE PROGRAMA (PC): Esta señal tiene como destino a el bus de direcciones de la memoria, es de 16 bits, síncrona.
CSDIR: Esta señal controla en que forma y que dato se deja pasar determinado dato, es de 2 bits.
SSEL4: Esta señal tiene como destino el bus de direcciones de la memoria, es de 13 bits, este selector se encarga de definir qué datos deja pasar y el dato de la salida lo divide entre 4 para de esta forma obtener 13 bits, en los cuales se encuentra la dirección de palabra que se va a leer en la memoria. (los datos de entrada vienen direccionados hacia un byte de memoria)

SELECTOR 5 (SEL5):

Ese bloque sirve para seleccionar los datos que tiene como destino el sumador del contador de programa (Sum), además de seleccionar los datos también sirve como extensor de signo dependiendo de los diferentes tipos de formato (se explican en la sección 4.1.4).

Señales de interacción del selector 5.
SSEL (ENTRADA DE REGISTRO DE INSTRUCCIONES): Esta señal proviene del registro de instrucciones, tiene 32 bits de ancho, el selector se encargad de organizar los 13 bits que pasan para realizar la suma con el contador de programa.
ENTRADA DE DATO FIJO 004h: Este es un valor fijo de 4, este valor es debido a que la arquitectura define que el acceso a memoria se realiza por byte, lo que significa que para obtener instrucciones alineadas cada 4 bytes hay una nueva instrucción.
CS5: Esta señal controla en que forma y que dato se deja pasar. es de 2 bit.
E1SUM: Esta señal es la salida tiene como destino el contador de programa, es de 16 bits y deja pasar el dato que se haya seleccionado.

SELECTOR DE CONTADOR DE PROGRAMA (SELPC):

Ese bloque sirve para seleccionar los datos que tiene como destino el registro del contador de programa, además de seleccionar los datos también sirve como extensor de signo dependiendo de los diferentes tipos de formato (se explican en la sección 4.1.4).

Señales de interacción del selector PC.
ENTRADA DE REGISTROS DE DATOS (SREGDAT): Esta señal con destino final el contador del programa es una señal que sirve para guardar un nuevo valor al PC.
ENTRADA DE ALU (SALU): Esta señal tiene como destino final el bus de direcciones de la memoria, por lo tanto, solo se usan los primeros 16 bits para este propósito.
ENTRADA DE SUMADOR DEL CONTADOR DE PROGRAMA (SSUM): Esta señal es de 16 bits. contiene el valor del programa sumada con 4 o con un valor inmediato dependiendo de la instrucción ejecutada.
ENTRADA DE REGISTROS DE ESTADO (SREGEST): Esta señal se utiliza para interrupciones, por medio del registro de control y estados, se accede a la dirección a la cual se debe saltar cuando ocurre una interrupción.
CSPC: Esta señal controla en que forma y que dato se deja pasar determinado dato, es de 2 bit

SALIDA A REGISTRO DEL CONTADOR DE PROGRAMA (SSELPC): Es la señal de salida del bloque, contiene un valor que se desea guardar en el registro del contador del programa.

SELECTOR REGISTROS DE CONTROL Y ESTADO (SELRE):

Ese bloque sirve para seleccionar los datos que tiene como destino el registro de estados, además de seleccionar los datos también sirve como extensor de signo dependiendo de los diferentes tipos de formato (se explican en la sección 4.1.4).

Señales de interacción del SELRE.
ENTRADA DE DATOS DE CONTROL (DATCSR): Esta señal consta de 32 bits sirve para enviar un determinado dato al registro de estados.
(E1ALU): Este bus de datos de 32 bits contiene el valor de un registro o de un valor inmediato que se desea guardar en el registro de control y estados.
ENTRADA DE CONTADOR DE PROGRAMA (SPC): Esta señal contiene el valor el valor del contador del programa, sirve para almacenarlo en caso de interrupciones.
ENTRADA DE ALU (SALU): Esta señal contiene el valor resultado de una operación AND o OR que se desea guardar en el registro de estados.
ENTRADA DE REGDATA (SREGDATA): Esta señal contiene de un dato proveniente de memoria para ser almacenado en un registro de estados.
CSRE: Esta señal controla en que forma y que dato se deja pasar determinado dato, es de 2 bit
SALIDA A REGISTRO DE CONTROL Y ESTADOS(REWD): Esta señal contiene el valor a guardar en el registro de estados, elige por medio de una señal de control entre las señales de entrada o una variación de estas.

4.1.4 INSTRUCCIONES.

Instrucciones RISC-V:

Las instrucciones de la arquitectura RISC-V se subdividen en diferentes direccionamientos, estos indican el tipo de instrucción general que se va a realizar, como lo son operaciones entre registros o con valores inmediatos (operaciones de suma/resta, lógica, etc.), operaciones de comparación y salto entre otros.

Los diferentes formatos de instrucciones se diferencian por medio del *opcode* (código de operación) el cual se ubica en los primeros siete bits de la palabra, el cual permite identificar qué formato de instrucción se está realizando, si es operación entre registros, operaciones con un valor inmediato, carga a memoria, etc.). Para identificar que instrucción se va a realizar del formato definido por el *opcode,*Se utilizan los valores de Func3 y Func7, los cuales indican que operación específicamente se va a realizar.(suma, resta, mayor, etc.) Todas las instrucciones, formatos y otras especificaciones de la arquitectura se pueden encontrar en las referencias de RISC-V.[6],[7] y [8].

Instrucciones elegidas para domótica.

Para la elección de las instrucciones se profundizó y se tuvieron en cuenta los aspectos que se utilizan en las aplicaciones de domótica como lo son sus requerimientos/necesidades al momento de realizar operaciones, manejo de sensores, actuadores, etc. Concluyendo con la elección de las siguientes instrucciones.

- ✓ Instrucciones aritméticas, lógicas y de corrimientos, entre registros y con valores inmediatos, esto se realiza para control de los actuadores debido a que en ocasiones se desea mostrar el valor de los sensores en los actuadores. Por ejemplo, una pantalla led, en la cual se necesita convertir a ASCCI el valor numérico para ser presentado en la pantalla LCD.

- ✓ Instrucciones de comparación y salto. Utilizadas para comparar los valores de diferentes sensores, con diferentes datos, saltar a una subrutina y ejecutar un actuador y regresar, por ejemplo, para medición de temperatura y mostrar el resultado en una pantalla LCD, si la temperatura cambia se ejecuta una subrutina en la cual se actualice el LCD con el nuevo valor de temperatura, y para esto se necesita operaciones de comparación para verificar la temperatura, corrimientos y aritméticas para convertir a BCD(decimal codificado en binario).
- ✓ Instrucciones de carga desde memoria. Las cuales no solo permiten leer la memoria, sino que también permiten leer cualquier periférico, conocer su estado y sus datos, se implementan lecturas de 8 bits, 16 bits y 32 bits, dando libertad para usar una amplia gama de sensores de lectura en paralelo o lectura en serie de valores altos.
- ✓ Carga de datos a memoria, las cuales se utilizan para enviar datos a memoria y a los diferentes periféricos (Mapeados en memoria), lo cual permite realizar una comunicación completa con los periféricos o sus controladores, lo que permite darles ordenes de activar o desactivar las diferentes funciones que poseen los sensores y actuadores.
- ✓ Operaciones de registros de estado y control, con estas instrucciones se le da libertad al programador de controlar los destinos a los cuales se salta cuando ocurre una interrupción, habilitación o des habilitación de interrupciones, útil para ocasiones que se esperen interrupciones de diferentes periféricos y haya periféricos con mayor prioridad que otros, entonces al interrumpir un procesador de menores privilegios este puede activar las interrupciones por medio de software(un manejador de interrupciones) para estar pendiente de la interrupción del periférico de mayor privilegio.

Formatos de instrucción, operaciones e instrucciones.

Los tipos de instrucciones son: Tipo R (Operaciones entre registros), Tipo I (Operaciones entre registro y valor inmediato), Tipo I (Carga desde memoria), Tipo B (Operaciones de comparación y salto), Tipo S (Operaciones de carga de datos desde memoria).

Código de operación.
A continuación, se detallan algunos nombres que se utilizan en los diferentes modos de direccionamiento, y por consiguiente en las diferentes instrucciones.
Opcode: Identifica que direccionamiento y que lógica se va a realizar por ejemplo se usa 0110011 para operaciones lógicas y aritméticas entre registros (Tipo R).
Funtct3: Sirve para identificar qué operación exactamente se va a realizar, ejemplo si se tiene 000 se sabe que es una operación de suma en el formato tipo R.
Funct7: Sirve para identificar qué operación exactamente se va a realizar en caso de que funct3 se quede pequeño, Ejemplo suma y resta tienen el mismo Funct3 000 pero diferente Funct7 suma=> 0000000 resta =>0000010

Registros.
Rd: Dirección del registro destino en el cual se va a guardar la operación que se realice. En este caso se utilizan 32 registros por lo cual este espacio tiene un tamaño de 5 bits direccionamiento.
Reg1: Dirección del registro del primer operando de la operación. En este caso se utilizan 32 registros por lo cual este espacio tiene un tamaño de 5 bits de direccionamiento.
Reg2: Dirección del registro del segundo operando de la operación. En este caso se utilizan 32 registros por lo cual este espacio tiene un tamaño de 5 bits de direccionamiento.
Inm: Valor inmediato.

A continuación, se explican las operaciones seleccionadas para cada tipo.

INSTRUCCIONES A IMPLEMENTA TIPO R: Este formato de instrucción sirve para identificar las operaciones entre registros. El formato se muestra en la Figura 12.

31:25	24:20	19:15	14:12	11:7	6:0
Funct7	Reg2	Reg1	Funct3	Rd:RegDestino	Opcode

Figura 12. Formato de direccionamiento tipo R.

A continuación, se enuncian las operaciones implementadas y su funcionamiento.

- Suma: Esta operación suma dos registros (del banco de registros internos del procesador), estos registros tienen como dirección los campos Reg1 y Reg2 de la instrucción. La operación no realiza verificación de desbordamiento de palabra ("*overflow*").

  Assembler: ADD Rd Reg1 Reg2 # Rd= Reg1+ Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0000000	Reg2	Reg1	000	Rd:RegDestino	0110011

- Corrimiento lógico a la izquierda: Esta operación corre el dato que contiene Reg1, Reg2 (Primeros 5 valores, máximo valor 32) veces a la izquierda. Los bits que quedan libres son puestos en 0, ejemplo Hexadecimal A=2345 B=0004 SAL=3450

  Assembler: SLL Rd Reg1 Reg2 # Rd= Reg1<< Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0000000	Reg2	Reg1	001	Rd:RegDestino	0110011

- Comparación menor con signo: Esta operación compara bit a bit el contenido de Reg1 y Reg2 y guarda el resultado en Rd. Guarda un 1 si Reg1 es menor a Reg2 y un 0 si Reg1 es mayor a Reg2. Esta operación se realiza teniendo en cuenta los bits de signo.

  Assembler: SLT Rd Reg1 Reg2 # Rd=1 Si Reg1< Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0000000	Reg2	Reg1	010	Rd:RegDestino	0110011

- Comparación menor sin signo: Esta operación compara bit a bit el contenido de Reg1 y Reg2 y guarda el resultado en Rd. Guarda un 1 si Reg1 es menor a Reg2 y un 0 si Reg1 es mayor a Reg2. Esta operación se realiza ignorando los bits de signo.

  Assembler: SLTU Rd Reg1 Reg2 # Rd=1 Si Reg1< Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0000000	Reg2	Reg1	011	Rd:RegDestino	0110011

- XOR: Esta operación realiza el XOR (Ver Tabla 5) entre los registros Reg1 y Reg2 (la operación se realiza bit a bit) y guarda el resultado en Rd.

  Assembler: XOR Rd Reg1 Reg2 # Rd= Reg1 XOR Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0000000	Reg2	Reg1	100	Rd:RegDestino	0110011

A	B	Sal
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 5 XOR.

Corrimiento lógico a la derecha: Esta operación corre el dato que contiene Reg1 en Reg2 (Primeros 5 valores, máximo valor 32) veces a la derecha. Los bits que quedan libres son puestos en 0, ejemplo Hexadecimal A=2345 B=0004 SAL=0234

Assembler: SRL Rd Reg1 Reg2 # Rd= Reg1>> Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0000000	Reg2	Reg1	101	Rd:RegDestino	0110011

OR: Esta operación realiza el OR (Ver Tabla 6) entre los registros Reg1 y Reg2 (la operación se realiza bit a bit) y guarda el resultado en Rd.

Assembler: OR Rd Reg1 Reg2 # Rd= Reg1 OR Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0000000	Reg2	Reg1	110	Rd:RegDestino	0110011

A	B	Sal
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 6. OR

AND: Esta operación realiza el AND (Ver Tabla 7) entre los registros Reg1 y Reg2 (la operación se realiza bit a bit) y guarda el resultado en Rd.

Assembler: AND Rd Reg1 Reg2 # Rd= Reg1 AND Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0000000	Reg2	Reg1	111	Rd:RegDestino	0110011

A	B	Sal
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 7. AND.

- Corrimiento aritmético a la derecha: Esta operación corre el dato que contiene Reg1, Reg2 veces a la derecha. Los bits que quedan libres son puestos con el bit más significativo de Reg1, ejemplo Hexadecimal A=F345 B=0004 SAL=FF34

  Assembler: SRA Rd Reg1 Reg2 # Rd= Reg1>> Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0100000	Reg2	Reg1	101	Rd:RegDestino	0110011

- Resta: Esta operación resta dos registros (del banco de registros internos del procesador), estos registros tienen como dirección los campos Reg1 y reg2 de la instrucción, el resultado es guardado en un tercer registro identificado por la dirección puesta en Rd. En esta operación no realiza verificación de desbordamiento de palabra("*overflow*")

  Assembler: SUB Rd Reg1 Reg2 # Rd= Reg1- Reg2

31:25	24:20	19:15	14:12	11:7	6:0
0100000	Reg2	Reg1	000	Rd:RegDestino	0110011

INSTRUCCIONES TIPO I: se van a implementar instrucciones de lógica (and, or, xor), aritméticas (Suma y resta) y corrimientos (Izquierda, derecha, etc.). Estas operaciones son operaciones de un registro contra un valor inmediato extendido en signo, este valor se encuentra en la instrucción. El formato se muestra en la Figura 13.

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	Funct3	Rd:RegDestino	Opcode

Figura 13. Formato de direccionamiento Tipo I.

Inm. Inmediato: este es un valor para realizar operaciones con un dato introducido directamente en la instrucción, consta de 12 bits. Estos bits se encuentran ordenados lo que significa que el menos significativo se encuentra en la posición 20 de la palabra y el más significativo en la posición 32. Al tener 12 bits para el valor inmediato permite tener números desde -2048 hasta +2047 con signo.

Este formato de instrucción no cuenta con Funct7, salvo en los corrimientos lógicos y aritméticos donde se ubica en los 7 bits más significativos de la palabra y debe tomar un valor de 0000000.

A continuación, se enuncian las operaciones y su funcionamiento.

- Suma: Esta operación suman el registro Reg1 con el valor inmediato (Este valor toma en cuenta el signo), La operación no realiza verificación de desbordamiento de palabra("*overflow*").
  Assembler: ADDI Rd Reg1 Inm # Rd= Reg1+ Inm

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	000	Rd:RegDestino	0010011

- Comparación menor con signo: Esta operación compara bit a bit el contenido de Reg1 y de Inm (Con extensión signo) y guarda el resultado en Rd, guarda un 1 si Reg1 es menor a Inm y un 0 si Reg1 es mayor a Inm. Esta operación se realiza teniendo en cuenta los bits de signo.
  Assembler: SLTI Rd Reg1 Inm # Rd=1 Si Reg1< Inm

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	010	Rd:RegDestino	0010011

- Comparación menor sin signo: Esta operación compara bit a bit el contenido de Reg1 y de Inm (Con extensión signo) y guarda el resultado en Rd, guarda un 1 si Reg1 es menor a Inm y un 0 si Reg1 es mayor a Inm. Esta operación se realiza ignorando los bits de signo.
  Assembler: SLTIU Rd Reg1 Inm # Rd=1 Si Reg1< Inm

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	011	Rd:RegDestino	0010011

- XORI: Esta operación realiza el XOR entre los registros Reg1 y Inm (Con extensión signo), la operación se realiza bit a bit y guarda el resultado en Rd. (ver operación XOR en Tabla 5 XOR.)
  Assembler: XORI Rd Reg1 Inm # Rd= Reg1 XOR Inm

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	100	Rd:RegDestino	0010011

- ORI: Esta operación realiza el OR entre los registros Reg1 y Inm (Con extensión signo) (la operación se realiza bit a bit) y guarda el resultado en Rd. (ver operación OR en Tabla 6)
  Assembler: OR Rd Reg1 Inm # Rd= Reg1 OR Inm

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	110	Rd:RegDestino	0010011

- ANDI: Esta operación realiza el AND entre los registros Reg1 y Inm (Con extensión signo), la operación se realiza bit a bit y guarda el resultado en Rd. (ver operación AND en Tabla 7)
  Assembler: AND Rd Reg1 Inm # Rd= Reg1 AND Inm

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	111	Rd:RegDestino	0010011

- Corrimiento lógico a la izquierda: Esta operación corre el dato que contiene Reg1, a Inm veces a la izquierda. Los bits que quedan libres son puestos en 0. Ejemplo Hexadecimal A=2345 B=0004 SAL=3450
  Assembler: SLLI Rd Reg1 Inm # Rd= Reg1<< Inm [4:0]

31:25	25:20	19:15	14:12	11:7	6:0
0000000	Inm	Reg1	001	Rd:RegDestino	0010011

- Corrimiento aritmético a la derecha: Esta operación corre el dato que contiene Reg1, en Inm veces a la derecha. Los bits que quedan libres son reemplazados por el bit más significativo de Reg1. Ejemplo Hexadecimal A=2345 B=0004 SAL=0234
  Func3 = 101, Inm[11:5]=0000000;
  Assembler: SRAI Rd Reg1 Inm # Rd= Reg1>> Inm[4:0]

31:25	25:20	19:15	14:12	11:7	6:0
0000000	Inm	Reg1	101	Rd:RegDestino	0010011

- Corrimiento lógico a la derecha: Esta operación corre el dato que contiene Reg1, Inm veces a la derecha. Los bits que quedan libres son puestos en 0, ejemplo Hexadecimal A=2345 B=0004 SAL=F234.
  Assembler: SRLI Rd Reg1 Inm # Rd= Reg1>> Inm[4:0]

31:25	25:20	19:15	14:12	11:7	6:0
0100000	Inm	Reg1	101	Rd:RegDestino	0010011

Tipo I Load.

Adicional a las operaciones lógicas, aritméticas, etc. El formato de instrucción TIPO I también sirve para las operaciones carga de datos de registros a memoria, lo que se hace es variar es el contenido del Opcode.

- Carga de un byte al banco de registros: Esta operación carga el primer byte de una palabra de la memoria en el registro Rd. Recibe como datos el registro 1(Reg1) y el Inm (Con extensión signo), estos dos son sumados y esta suma da la dirección de memoria donde se debe ir a buscar el byte que se debe guardar en el Registro Rd (Este guardado se realiza manteniendo el signo)
  Assembler: LB Rd Reg1 Inm # Rd=Memoria [Reg1+Inm]

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	000	Rd:RegDestino	0000011

- Carga de dos bytes al banco de registros: Esta operación carga los dos primeros bytes de una palabra de la memoria en el registro Rd. Recibe como datos el registro 1 (Reg1) y el Inm (Con extensión signo), estos dos son sumados y esta suma da la dirección de memoria donde se debe ir a buscar los dos primeros bytes (*Half word*) que se debe guardar en el Registro Rd (Este guardado se realiza manteniendo el signo)
  Assembler: LH Rd Reg1 Inm # Rd=Memoria [Reg1+Inm]

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	001	Rd:RegDestino	0000011

- Carga de palabra (4 bytes) al banco de registros: Esta operación carga una palabra de la memoria en el registro Rd. Recibe como datos el registro 1 y el Inm (Con extensión signo), estos dos son sumados y esta suma da la dirección de memoria donde se debe ir a buscar la palabra que se debe guardar en el Registro Rd.
  Assembler: LW Rd Reg1 Inm # Rd=Memoria [Reg1+Inm]

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	010	Rd:RegDestino	0000011

- Carga de un byte sin signo al banco de registros: Esta operación carga el primer byte de una palabra de la memoria en el registro Rd. Recibe como datos el registro 1 (Reg1) y el Inm (Con extensión signo), estos dos son sumados y esta suma da la dirección de memoria donde se debe ir a buscar el byte que se debe guardar en el Registro Rd (Este guardado se realiza completando con 0 los campos restantes.)
  Assembler: LBU Rd Reg1 Inm # Rd=Memoria [Reg1+Inm]

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	100	Rd:RegDestino	0000011

- Carga de dos bytes sin signo al banco de registros: Esta operación carga los dos primeros bytes de una palabra de la memoria en el registro Rd. Recibe como datos el registro 1 (Reg1) y el Inm (Con extensión signo), estos dos son sumados y esta suma da la dirección de memoria donde se debe ir a buscar los dos primeros bytes que se debe guardar en el Registro Rd (Este guardado se realiza completando con 0 los campos restantes.))
  Assembler: LHU Rd Reg1 Inm # Rd=Memoria [Reg1+Inm]

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	101	Rd:RegDestino	0000011

Tipo I subrutina

El formato de instrucción TIPO I también sirve para direccionar una de las dos instrucciones de subrutina.

- Salto, enlace y registro: Esta instrucción guarda en Rd el contador de programa siguiente a la instrucción y modifica el contador de programa por la dirección comprendida por la suma del contenido del registro Reg1y el valor inmediato (Con extensión signo), poniendo en cero el último bit de la dirección.
  Assembler: JALR Rd Reg1 Inm # Rd=PC+4; # PC=(Reg1+INM) (Ultimo bit en 0)

31:20	19:15	14:12	11:7	6:0
Inm	Reg1	000	Rd:RegDestino	1100111

Tipo I registros de control y estado.

Instrucciones tipo I (Para leer y escribir registros de control y estados): sirven para conocer o controlar banderas de estado, habilitadores de interrupción, etc. El formato se muestra en la Figura 14.

31:20	19:15	14:12	11:7	6:0
CSR	Reg1/Inm	Funct3	Rd: RegDestino	opcode

Figura 14. Variación del formato de direccionamiento Tipo I para registro de estados.

CSR: Es la dirección del registro de estados a modificar.
Reg1/Inm: Dependiendo de la instrucción es el valor contenido en el registro 1 o el valor inmediato.

- Leer y escribir: Guarda en el registro Rd el valor que se encuentra en el registro de estados en la dirección CSR y además en el registro de estados en la dirección CSR guarda el valor de Reg1.
  Assembler: CSRRW Rd Reg1 csr # Rd=CSR CSR=Reg1.

31:20	19:15	14:12	11:7	6:0
CSR	Reg1/Inm	001	Rd: RegDestino	1110011

- Leer y cambiar bits: Guarda en el registro Rd el valor que se encuentra en el registro de estados en la dirección CSR y además en el registro de estados en la dirección CSR guarda el valor de Reg1 OR CSR(Anterior).
  Assembler: CSRRS Rd Reg1 csr # Rd=CSR CSR=Reg1 OR CSR(Anterior)

31:20	19:15	14:12	11:7	6:0
CSR	Reg1/Inm	010	Rd: RegDestino	1110011

- Leer y limpiar bits: Guarda en el registro Rd el valor que se encuentra en el registro de estados en la dirección CSR y además en el registro de estados en la dirección CSR guarda el valor del complemento a uno del Reg1 AND CSR(Anterior).
  Assembler: CSRRC Rd Reg1 csr# Rd=CSR CSR=~Reg1 AND CSR (Anterior).

31:20	19:15	14:12	11:7	6:0
CSR	Reg1/Inm	011	Rd: RegDestino	1110011

- Leer y escribir valor inmediato: Guarda en el registro Rd el valor que se encuentra en el registro de estados en la dirección CSR y además en el registro de estados en la dirección CSR guarda el valor de Inm(sin extensión signo).
  Assembler: CSRRWI Rd Inm csr #Rd= CSR. CSR=Inm.

31:20	19:15	14:12	11:7	6:0
CSR	Reg1/Inm	101	Rd: RegDestino	1110011

- Leer y cambiar bits, valor inmediato: Guarda en el registro Rd el valor que se encuentra en el registro de estados en la dirección CSR y además en el registro de estados en la dirección CSR guarda el valor de Inm(sin signo) OR CSR(Anterior).
  Assembler CSRRSI Rd Inm csr # Rd=CSR CSR=Inm OR CSR(Anterior)

31:20	19:15	14:12	11:7	6:0
CSR	Reg1/Inm	110	Rd: RegDestino	1110011

- Leer y limpiar bits, valor inmediato: Guarda en el registro Rd el valor que se encuentra en el registro de estados en la dirección CSR y además en el registro de estados en la dirección CSR guarda el valor del complemento a uno del Inm(sin signo) AND CSR(Anterior).
  Assembler: CSRRCI Rd Inm csr # Rd=CSR CSR=~Inm AND CSR (Anterior).

31:20	19:15	14:12	11:7	6:0
CSR	Reg1/Inm	111	Rd: RegDestino	1110011

- Fin de interrupción: Esta operación se encarga de devolver los valores que estaban antes de la interrupción, guarda en el contador del programa el valor del PC que se encuentra en el registro de estados MPEC. (ver sección 4.1.2 para más información).
  Assembler: MRET #Fin interrupción.

31:25	24:20	19:15	14:12	11:7	6:0
0011000	00010	00000	000	00000	1110011

- Espera de interrupción: Esta operación se encarga de esperar una interrupción externa, es un bucle, hasta que no halla interrupción externa no se continua con la siguiente instrucción.
  Assembler: WFI #Esperando interrupción.

31:25	24:20	19:15	14:12	11:7	6:0
0001000	00101	00000	000	00000	1110011

INSTRUCCIONES TIPO S: Estas instrucciones sirven para cargar datos a la memoria. El formato se muestra en la Figura 15.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [11:5]	Reg2	Reg1	Funct3	Inm [4:0]	Opcode

Figura 15. Formato de direccionamiento tipo S.

Inm : Inmediato: este es un valor para realizar operaciones con un dato introducido directamente en la instrucción, consta de 12 bits, lo que permite tener números desde 0 hasta 4095 sin signo o desde -2048 hasta +2047 con signo. En este formato de instrucción, el apartado inmediato se presenta se parado en dos partes, La primera los bits del 4 al 0 se ubican donde anteriormente se ubicaba Rd, y la segunda parte bits del 11 al 5 se ubican donde anterior mente se ubicaba Funct7. Se ubica de esta manera para preservar la mayor cantidad de similitudes con los distintos formatos de instrucción (opcode en las mismas posiciones, funct 3 en las mismas posiciones Reg1 y Reg2 si son necesarios en las mismas posiciones, etc.).

- Carga de un byte a memoria: Esta operación carga el primer byte de una palabra del registro Reg2 en memoria en la dirección comprendida por la suma del contenido del registro Reg1 y el Inm (Con signo).
  Assembler: SB Reg2 Inm (Reg1) # Memoria [Reg1+Inm] = Reg2.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [11:5]	Reg2	Reg1	000	Inm [4:0]	0100011

- Carga de dos bytes a memoria: Esta operación carga los dos primeros bytes de una palabra del registro Reg2 en memoria en la dirección comprendida por la suma del contenido del registro Reg1 y el Inm (Con signo).
  Assembler: SH Reg2 Inm (Reg1) # Memoria [Reg1+Inm] = Reg2.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [11:5]	Reg2	Reg1	001	Inm [4:0]	0100011

- Carga de palabra a memoria: Esta operación carga una palabra del registro Reg2 en memoria en la dirección comprendida por la suma del contenido del registro Reg1 y el Inm (Con signo).
  Assembler: SW Reg2 Inm (Reg1) # Memoria [Reg1+Inm] = Reg2.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [11:5]	Reg2	Reg1	010	Inm [4:0]	0100011

INSTRUCCIONES TIPO B: Estas instrucciones sirven para comparar (saber si es mayor, menor igual, etc.) y saltar modificando el contador de programa. El formato se muestra en la Figura 16.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [12][10:5]	Reg2	Reg1	Funct3	Inm [4:1][11]	Opcode

Figura 16. Formato de direccionamiento tipo B.

Inm[0]=0;
Inm : Inmediato. En este formato de instrucción funciona de manera similar al Tipo S. La diferencia radica en que el valor del inmediato esta multiplicado por 2, lo que significa corrido un bit a la izquierda, conservando el bit del signo. Para evitar que el bit más significativo del inmediato se pierda, este es puesto en la posición 7 del registro de instrucciones, de esta manera se reemplaza por el bit menos significativo del inmediato el cual siempre se considera en 0. Al realizar esto conlleva a aumentar el valor del inmediato hasta -4096 hasta +4094,[8].

- Igual: Esta operación compara el Reg1 y El Reg2 (tomando en cuenta el signo) y determina si son iguales, si no son iguales continua con la siguiente instrucción, si son iguales modifica el contador de programa (PC) sumándole el dato inmediato (Con extensión signo) más el contador de programa en el que iban.
  Assembler: BEQ Reg1 Reg2 Inm #Si es verdad PC=PC+Inm.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [12][10:5]	Reg2	Reg1	000	Inm [4:1][11]	1100011

- Diferente: Esta operación compara el Reg1 y El Reg2 (tomando en cuenta el signo) y determina si son diferentes, si son iguales continua con la siguiente instrucción, si son diferentes modifica el contador de programa (PC), sumándole el dato inmediato (Con extensión signo) más el contador de programa en el que iban.
  Assembler: BNE Reg1 Reg2 Inm #Si es verdad PC=PC+Inm.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [12][10:5]	Reg2	Reg1	001	Inm [4:1][11]	1100011

- Menor: Esta operación compara el Reg1 y El Reg2 (tomando en cuenta el signo) y determina si Reg1 es menor a Reg2. Si Reg1 es mayor o igual continua con la siguiente instrucción, si es menor

modifica el contador de programa (PC) sumando el dato inmediato (Con extensión signo) más el contador de programa en el que iban.

Assembler: BLT Reg1 Reg2 Inm #Si es verdad PC=PC+Inm.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [12][10:5]	Reg2	Reg1	100	Inm [4:1][11]	1100011

- Mayor o igual: Esta operación compara el Reg1 y El Reg2 (tomando en cuenta el signo) y determina si Reg1 es mayor o igual a Reg2. Si Reg1 es menor continua con la siguiente instrucción, si es mayor o igual modifica el contador de programa (PC) sumando el dato inmediato (Con extensión signo) más el contador de programa en el que iban.

  Assembler: BGE Reg1 Reg2 Inm #Si es verdad PC=PC+Inm.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [12][10:5]	Reg2	Reg1	101	Inm [4:1][11]	1100011

- Menor sin signo: Esta operación compara el Reg1 y El Reg2 (sin tomar en cuenta el signo) y determina si Reg1 es menor a Reg2. Si Reg1 es mayor o igual continua con la siguiente instrucción, si es menor modifica el contador de programa (PC) sumando el dato inmediato (Con signo) más el contador de programa en el que iban.

  Assembler: BLTU Reg1 Reg2 Inm #Si es verdad PC=PC+Inm.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [12][10:5]	Reg2	Reg1	110	Inm [4:1][11]	1100011

- Mayor o igual sin signo: Esta operación compara el Reg1 y El Reg2 (tomando en cuenta el signo) y determina si Reg1 es mayor a Reg2. Si Reg1 es menor continua con la siguiente instrucción, si es mayor o igual modifica el contador de programa (PC) sumando el dato inmediato (Con signo) más el contador de programa en el que iban.

  Func3 = 111; Assembler: BGEU Reg1 Reg2 Inm #Si es verdad PC=PC+Inm.

31:25	24:20	19:15	14:12	11:7	6:0
Inm [12][10:5]	Reg2	Reg1	Funct3	Inm [4:1][11]	1100011

INSTRUCCIONES TIPO U: Estas instrucciones sirven para cargar datos de manera inmediata en los registros del procesador. El formato se muestra en la Figura 17.

[31:12]	[11:7]	[6:0]
Inm[19:0]	Rd	Opcode

Figura 17. Formato de direccionamiento tipo U.

Inm: Este valor en este formato de instrucción toma 20 bits, con valores desde -524288 hasta +524286.

- Carga superior inmediata: Esta operación carga en el registro destino Rd el valor inmediato (Con signo) corrido 12 posiciones a la izquierda (Esta es la definición del tipo U), completando el resto con ceros. Ejemplo si Inm= Hexadecimal 00099988 => Rd=99988000

  Assembler: LUI Rd Inm #.

[31:12]	[11:7]	[6:0]
Inm[19:0]	Rd	0110111

- Suma superior inmediata al contador de programa: Esta operación carga en el registro destino Rd el valor inmediato (Con signo) corrido 12 posiciones a la izquierda (completando el resto con ceros) más el valor del contador de programa.

  Assembler: AUIPC Rd Inm #. RD= PC+INM

[31:12]	[11:7]	[6:0]
Inm[19:0]	Rd	0010111

INSTRUCCIONES TIPO J: Estas instrucciones sirven para realizar saltos en las instrucciones. El formato se muestra en la Figura 18.

[31:12]	[11:7]	[6:0]
Inm[20|10:1|11|19:12]	Rd	Opcode

Figura 18. Formato de direccionamiento tipo J.

Inm[0]=0;

Este tipo de instrucción es similar a la Tipo U. La diferencia radica en que el valor del inmediato esta multiplicado por 2, lo que significa corrido un bit a la izquierda, conservando el bit del signo. Esto conlleva a aumentar el valor del inmediato hasta -1048576 hasta +1048574. El valor inmediato se ubica en el direccionamiento de tal manera que: en la posición 31 siempre este el bit correspondiente a signo. En las posiciones 19 a 12, se ubican los bits más significativos del valor inmediato. En la posición 20 se ubica el bit siguiente del valor inmediato y en las posiciones 31 a la 21, se ubican los 10 bits menos significativos del valor inmediato, ignorando la posición 0 del inmediato, está siempre se reemplaza por 0. Esta asignación de bits del valor inmediato se hace para lograr mantener la mayor similitud posible con el valor inmediato del formato Tipo U.

- Salto y enlace: Esta instrucción guarda en Rd el contador de programa siguiente a la instrucción y modifica el contador de programa por la dirección comprendida por el valor inmediato (Con signo) más el contador de programa siguiente a JAL. Esta instrucción es usualmente usada para realizar subrutinas.
  Func3 = 000; Assembler: JAL Rd Reg1 Inm # Rd=PC+4; PC=PC+INM

[31:12]	[11:7]	[6:0]
Inm[20|10:1|11|19:12]	Rd	1101111

RESUMEN FORMATOS DE INSTRUCCIÓN.

En este resumen se muestra una breve tabla con las características de los diferentes modos de direccionamiento (ver Tabla 8), y se presenta una tabla final con todas las operaciones implementadas en el procesador. (Ver Tabla 9).

Opcode: Código de operación (define el tipo de operación y el formato de operación a realizar
Funct3: Especifica que operación se va a realizar, se puede repetir para distintas operaciones, siempre y cuando estén en diferente formato de instrucción.
Funct7: Funciona como complemento de func3, en caso de que este último se quede pequeño para todas las operaciones.
Reg1 y Reg2: Dirección del banco de registro, utilizada para leer un dato.
Rd: Dirección del banco de registro, utilizada para guardar un dato.
CSR: Dirección de los registros de estados, utilizada para leer/modificar un dato del registro de control y estado.

Tipo R.(Operaciones entre registros)					
31:25	24:20	19:15	14:12	11:7	6:0
Funct7	Reg2	Reg1	Funct3	Rd: RegDestino	Opcode

Tipo I(Operaciones, entre registro y valor inmediato, carga de datos desde memoria, Control de registro de estados, Salto y registro (Subrutina))				
31:20	19:15	14:12	11:7	6:0
Inm/CSR	Reg1	Funct3	Rd: RegDestino	Opcode

Tipo S(Carga de datos a memoria)					
31:25	24:20	19:15	14:12	11:7	6:0
Inm [11:5]	Reg2	Reg1	Funct3	Inm [4:0]	Opcode

Tipo B (Comparación y salto)					
31:25	24:20	19:15	14:12	11:7	6:0

Inm [12][10:5]	Reg2	Reg1	Funct3	Inm [4:1][11]	Opcode
Inm[0]=0;					
Tipo U (Suma con corrimiento (12 bits))					
[31:12]		[11:7]		[6:0]	
Inm		Rd		Opcode	
Tipo J (Salto y enlace (Subrutina))					
[31:12]		[11:7]		[6:0]	
Inm[20|10:1|11|19:12]		Rd		Opcode	

Tabla 8. Resumen de modos de direccionamiento.

RESUMEN INSTRUCCIONES.

En la siguiente tabla se muestra un resumen de las 44 instrucciones según el formato de instrucción:

Reg1: Dirección del registro 1(Banco de registros).
Reg2: Dirección del registro 2(Banco de registros).
Inm : Valor inmediato.

Instrucciones: Aritméticas, lógicas, corrimiento y de comparación(sin salto) Todas estas operaciones se guardan en Rd				
Instrucción	Operadores	Assembler	Formato instrucción	Ciclos de reloj*
Aritmética **Suma**	Reg1, Reg2	ADD	TIPO R	9
	Reg1, Inm (12 bits)	ADDI	TIPO I	9
Aritmética **Resta**	Reg1, Reg2	SUB	TIPO R	9
Aritmética **Carga superior inmediata**	Inm (20 bits)	LUI	TIPO U	9
Aritmética **Suma superior inmediata al contador de programa**	Inm(20 bits)	AUIPC	TIPO U	9
Lógica **XOR**	Reg1, Reg2	XOR	TIPO R	9
	Reg1, Inm (12 bits)	XORI	TIPO I	9
Lógica **OR**	Reg1, Reg2	OR	TIPO R	9
	Reg1, Inm (12 bits)	ORI	TIPO I	9
Lógica **AND**	Reg1, Reg2	AND	TIPO R	9
	Reg1, Inm (12 bits)	ANDI	TIPO I	9
Comparación **Comparación menor con signo**	Reg1, Reg2	SLT	TIPO R	9
	Reg1, Inm (12 bits)	SLTI	TIPO I	9
Comparación	Reg1, Reg2	SLTU	TIPO R	9

Comparación menor sin signo	Reg1, Inm (12 bits)	SLTIU	TIPO I	9
Corrimiento **Corrimiento lógico a la izquierda**	Reg1, Reg2	SLL	TIPO R	9
	Reg1, Inm (12 bits)	SLLI	TIPO I	9
Corrimiento **Corrimiento lógico a la derecha**	Reg1, Reg2	SRL	TIPO R	9
	Reg1, Inm (12 bits)	SRLI	TIPO I	9
Corrimiento **Corrimiento aritmético a la izquierda**	Reg1, Reg2	SRA	TIPO R	9
	Reg1, Inm (12 bits)	SRAI	TIPO I	9
Instrucciones de comparación y salto				
Igual	Reg1, Reg2, Inm(12 bits)	BEQ	TIPO B	10
Diferente	Reg1, Reg2, Inm(12 bits)	BNE	TIPO B	10
Menor	Reg1, Reg2, Inm(12 bits)	BLT	TIPO B	10
Mayor o igual	Reg1, Reg2, Inm(12 bits)	BGE	TIPO B	10
Menor sin signo	Reg1, Reg2, Inm(12 bits)	BLTU	TIPO B	10
Mayor o igual sin signo	Reg1, Reg2, Inm(12 bits)	BGEU	TIPO B	10
Instrucciones de salto y registro (Sub rutina).				
Salto y registro	Inm (20 bits)	JAL	TIPO J	10
Salto y registro2	Reg1, Inm (12 bits)	JALR	TIPO I	10
Instrucciones de registros de estados y control.				
Leer y escribir	Reg1, CSR (12 bits)	CSRRW	TIPO I	8
Leer y cambiar bits	Reg1, CSR (12 bits)	CSRRS	TIPO I	9
Leer y limpiar bits	Reg1, CSR (12 bits)	CSRRC	TIPO I	9
Leer y escribir valor inmediato	Inm(5 bits), CSR (12 bits),	CSRRWI	TIPO I	8
Leer y cambiar bits, valor inmediato	Inm(5 bits), CSR (12 bits),	CSRRSI	TIPO I	9
Leer y limpiar bits, valor inmediato	Inm(5 bits), CSR (12 bits),	CSRRCI	TIPO I	9
Fin interrupción	_____	MRET	TIPO R	9
Carga de datos (Desde y hacia memoria).				
Carga de un byte al banco de registros	Reg1, Inm (12 bits)	LB	TIPO I	12

Carga de dos bytes al banco de registros	Reg1, Inm (12 bits)	LH	TIPO I	12
Carga de un byte sin signo al banco de registros	Reg1, Inm (12 bits)	LBU	TIPO I	12
Carga de dos bytes sin signo al banco de registros	Reg1, Inm (12 bits)	LHU	TIPO I	12
Carga de palabra (4 bytes) al banco de registros	Reg1, Inm (12 bits)	LW	TIPO I	12
Carga de byte a memoria	Reg1, Reg2, Inm(12 bits)	SB	TIPO S	11
Carga de dos bytes a memoria	Reg1, Reg2, Inm(12 bits)	SH	TIPO S	11
Carga de palabra a memoria	Reg1, Reg2, Inm(12 bits)	SW	TIPO S	11

Tabla 9 Resumen de operaciones.

*Son los ciclos totales de ejecución de la instrucción incluyendo estados de *fetch*, *decode*, *execute*, y el estado en el que se verifica si ha ocurrido una interrupción.

Nota: En los compiladores de assembler RISC-V, también se suelen implementar pseudoinstrucciones, estas son una composición de instrucciones, lo que significa que se pueden escribir de forma acortada y el compilador se encarga de convertirlas a un formato completo (una instrucción o más). Algunos compiladores de RISC-V son el RIPES, (ver referencia [10]) y el RARS (ver referencia [11]), estos compiladores dan la posibilidad de convertir el assembler a binario y fueron utilizados para realizar las pruebas.

Un ejemplo de estas pseudo instrucciones son:

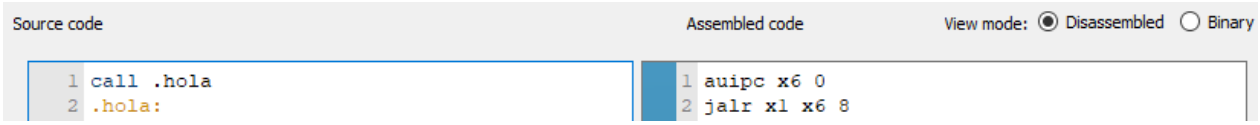


Figura 19. Pseudoinstrucción call., hecha con [10]

En la Figura 19 se muestra la pseudoinstrucción call, la cual se utiliza para llamado de funciones, a esta el compilador se encarga de realizar la conversión a instrucciones de RISC-V, la instrucción auipc, se encarga de reconocer el contador de programa de la instrucción actual y guardarlo en el registro temporal x6, jalr se encarga de realizar el salto respecto al contador de programa almacenado en x6 y se encarga de guardar la dirección de retorno en el registro x1.

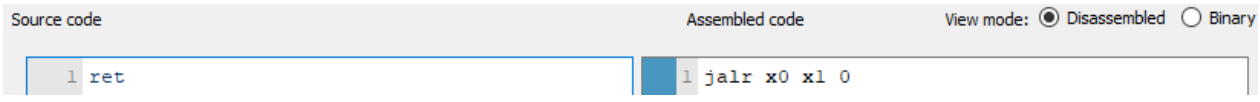


Figura 20. Pseudoinstrucción ret., hecha con [10]

En la Figura 20 se presenta la pseudoinstrucción ret, la cual es remplazada por la instrucción jalr, saltando al registro x1, En este registro se almacena la dirección de retorno cuando se llaman funciones en la arquitectura.

En la Tabla 10 se presentan algunas pseudoinstrucción y su versión de instrucciones en la arquitectura RISC-V. Para una mayor referencia se puede consultar la referencia [6]

PSEUDOINSTRUCCIÓN	INSTRUCCIÓN	SIGNIFICADO
bnez x ValorDeSalto	bne x x0 ValorDeSalto	Compara para saber si el registro x es diferente a 0.
bltz x ValorDeSalto	blt x x0 ValorDeSalto	Compara para saber si el registro x es menor que 0.
J ValorDeSalto	Jal x0, ValorDeSalto	Se salta a el valor de salto sin almacenar contador de programa.
li x 200 li x 20000	Addi x x0 200 lui x 5 addi x -480	Sirve para almacenar un valor inmediato en el registro x
nop	Addi x0 0	No se realiza operación
Mv xd x	Addi xd x 0	Guarda el registro x en el registro xd

Tabla 10. Pseudo instrucciones.

4.1.5 CARACTERÍSTICAS FINALES DE USO.

MEMORIA.

La memoria en la arquitectura está definida para que sea direccionable por bytes, eso significa que la palabra 1 de la memoria está ubicada en la dirección 4. Para obtener 8191 palabras en memoria significa direccionar a 32764 bytes, esto hace que los bloques de registro de direcciones y el registro de guardado del contador de programa tengan 15 bits. En la Figura 21 se muestran los direccionamientos de memoria por palabra (Lado izquierdo de la Figura 21) y por bytes (Lado derecho de la Figura 21), Se reconoce que cuando la memoria es direccionable por palabras al mandar la dirección 1 va tomar los 32 bits correspondientes a una palabra, pero cuando es direccionable por bytes se deben poner la dirección 4 que corresponde a la lectura de la palabra 1 de la memoria la cual corresponde a los bytes 4, 5, 6, 7 de la memoria.

MEMORIA DIRECCIONADA POR PALABRAS.	
	[31:0]
0	
1	
2	
3	
4	

MEMORIA DIRECCIONADA POR BYTES.				
	[31:25]	[24:16]	[15:8]	[7:0]
0	0	1	2	3
4	4	5	6	7
8				
12				
16				

Figura 21. Direccionamientos de memoria.

La memoria que se implementa es en realidad 4 memorias en paralelo de 8 bits de palabra esto con el fin de poder realizar guardado de byte, dos bytes (media palabra), 4 bytes (palabra), esta memoria utiliza una lógica que permite habilitar la escritura en las memorias por medio de 2 bits, esto se muestra en la Tabla 8. La

memoria como se dijo anteriormente consta de cuatro memorias, los archivos de inicialización de cada una de las memorias se nombrar de la siguiente forma [RAMD, RAMC, RAMB, RAMA], donde cada archivo de inicialización contiene un byte de la palabra donde RAMD contiene el valor más alto de la palabra[31:25] RAMC contiene el valor siguiente[24:16], RAMB contiene el valor siguiente[15:8] y RAMA el valor de byte más bajo de la palabra [7:0]. dando así que para completar una palabra se necesitan los cuatro archivos.

Memoria	
00	No escribe
01	Escribe en el primer Byte de la dirección
10	Escribe en los dos Bytes iniciales de la dirección
11	Escribe en toda la dirección (Palabra)

Tabla 11. Manejo de memoria.

Adicionalmente, la memoria es direccionable por palabra, esto significa que tiene 8191 posiciones de palabra, como la arquitectura está definida para ser direccionable por bytes (ver Figura 23), se necesita una lógica que realice la conversión de cambio de direccionamiento por byte a palabra. Esto se consigue realizando una división entre 4, en caso de que el número no sea divisible por cuatro, se aproximará a la dirección inferior del resultado. Dividir entre cuatro en binario es desplazar dos bits del número binario a la derecha. (Ver Figura 22)

Ejemplo división entre 4

Numero	Resultado
16 10000	4 00100
15 01111	3 00011

Figura 22. Ejemplo de división entre 4 en binario.

Ejemplo de direccionamiento por byte

Direccionamiento por byte	Direccionamiento por palabra
32764	8191
12000	3000

Figura 23. Ejemplo de direccionamiento por byte.

CONTROL DE INTERRUPCIONES.

Para que una interrupción sea habilitada debe pasar por varios procesos, primero se debe enviar una señal externa de tres bits activa en alto que indica que se le está pidiendo autorización al procesador para ejecutar una rutina de tratamiento para cierta interrupción, esta señal es guardada en el registro MIP en las posiciones [17,16,11] el cual contiene las interrupciones pendientes. (explicado en 4.1.2). Posteriormente el procesador verificar si las interrupciones globales del procesador están habilitadas en MSTATUS[3] (explicado en 4.1.2), esta habilitación global significa que pueden realizan interrupciones de distintos tipos como lo son interrupciones de tiempo (no implementadas), interrupciones de software (no implementadas), o interrupciones externas (implementadas), posterior a esto se revisa el registro MIE (explicado en 4.1.2), el cual contiene bits específicos para cada una de las interrupciones en los diferentes modos de manejo del procesador en este caso solo se revisa el MIE [17,16,11] debido a que este contiene los habilitadores de interrupciones externas. (Las únicas implementadas), el valor de MIE[17,16,11] debe ser menor al valor de MIP[17,16,11] para que la interrupción sea ejecutada de lo contrario el procesador la ignora hasta que se acabe la interrupción actual.

Para realizar el correcto control de interrupciones el usuario debe habilitar los registros de control y estado para manejo de interrupciones. Esto significa que hay que modificar el registro MSATUS[3], para habilitar interrupciones, y asignar a MTVEC[31:2] el valor de salto de inicio de la rutina de interrupción, estas modificaciones el usuario las realiza por medio de las instrucciones de control y estado. Inicialmente todos los registros de control y estados están inicializados en 0 por lo cual las interrupciones inician deshabilitadas. Como el procesador permite interrupciones múltiples el valor de MIP[17,16,11] es almacenado en cada ejecución de interrupción en el registro MIE[17,16,11], es decir si llega una interrupción nivel 3 y mie tiene un valor inferior a 3, entonces el procesador toma la interrupciones y guarda el valor de la nueva interrupción, por consiguiente, guarda un 3, y al finalizar las interrupciones es recuperado el valor que el registro MIE[17,16,11] tenía antes de la interrupción nivel 3, solo se recibe una interrupción por nivel pero el usuario puede modificar el valor del registro MIE de esta manera podría recibir múltiples dispositivos de un mismo nivel de interrupciones y leer quien causo la interrupción de dicho nivel de forma externa.

La arquitectura RISC-V, soporta interrupciones vectorizadas y auto vectorizadas, esto lo realiza por medio del registro MTVEC[1:0] (estas modificaciones el usuario las realiza por medio de las instrucciones de control y estado), donde el bit menos significativo MTVEC[0], indica si se tratan las interrupciones en forma vectorizadas o auto vectorizadas. Cuando MTVEC[0], está en 1 (Interrupciones auto vectorizadas), el procesador se encarga de decodificar para cada nivel de interrupción un lugar de memoria para saltar, esto lo logra dando un valor a cada nivel de interrupción llamado causa (mismo valor que se guarda en el registro MCAUSE.(explicado en 4.1.2)), este valor llamado causa se multiplica por 4 y se le suma el valor base del registro MTVEC (MTVEC[31:2]), y esta será la dirección de salto de la interrupción. En la Tabla 12 se presentan los valores de causa según nivel de interrupción.

CÓDIGO	VALOR DE CAUSA	DIRECCIÓN(CAUSA*4+MTVEC(31:2))
000	0	No interrumpe
001	11	44+MTVEC(31:2)
010	15	60+MTVEC(31:2)
011	20	80+MTVEC(31:2)
100	25	100+MTVEC(31:2)
101	30	120+MTVEC(31:2)
110	35	140+MTVEC(31:2)
111	40	160+MTVEC(31:2)

Tabla 12. Interrupciones, salto en modo auto vectorizado.

*Nota: cuando ocurre un error de código, el procesador siempre salta a la dirección MTVEC[31:2].

Cuando MTVEC[0], está en 0 las interrupciones se manejan de forma vectorizada, esto significa que todas saltan a la misma dirección comprendido por el valor base de MTVEC(MTVEC[31:2]), y allí se deben leer un valor externo o el registro MCAUSE por el cual se le indique la dirección donde se encuentra la rutina de tratamiento de la interrupción en memoria. El valor de MCAUSE contiene el valor de causa (Tabla 12) de las posiciones 30 a la 0 y en la posición 31 se pone en 1 cuando es una interrupción y en 0 cuando es una excepción.

Para activar las interrupciones el usuario lo realiza por medio de los registros de estado modificando el registro MSTATUS[3].

Proceso de manejo de interrupciones/excepciones.

Manejo de interrupción por el procesador.

1. Se levanta la entrada externa que identifica que hay una interrupción, esta se almacena en MPI (registro de interrupciones pendientes).
2. Se termina de ejecutar la instrucción que había en el momento que se activó la interrupción.
3. Posteriormente se verifica que el nivel de interrupción que llegó sea de mayor nivel de interrupción actual, esto se realiza verificando que el valor de MIP[17,16,11] sea mayor a el valor de

MIE[17.16.11], y después el procesador verifica si las interrupciones están habilitadas globalmente. (MSTATUS[3] en 1 (habilitador global de interrupciones)

4. Seguido a esto se procede a guardar los registros MCAUSE, MEPC, MTVAL, MIE en memoria, esto se realiza utilizando el registro interno del procesador stack-pointer el cual indica la posición en la que se va a almacenar, este registro siempre se inicializa con todos sus bits en 1 y decrece para almacenar.
5. Seguido de esto se guarda el contador de programa en el registro de control y estado MEPC (se guarda el valor siguiente a la instrucción en la que se generó la interrupción).
6. Luego, el procesador escribe en el registro de control y estado MCAUSE el valor de la excepción. Si es interrupción el bit más alto del registro es puesto en 1.
7. A continuación, se guarda la dirección de la interrupción en el contador del programa. Esta dirección debe estar almacenada previamente en MTVEC. Esta dirección se guarda respecto a lo que se explicó al inicio de esta sección (CONTROL DE INTERRUPCIONES).
8. Posterior a esto se modifica el registro MTVAL, cuando es interrupción almacena un 0, cuando es falla de código almacena la instrucción que causo el fallo.
9. Se ejecuta la rutina de tratamiento de la interrupción/excepción.

Proceso de manejo de excepción por instrucción ilegal por el procesador.

- Se verifica la instrucción, código de operación y complementos (funct3 y funct7), si no son válidos se salta a proceso de error.
- El procesador pone en uno el bit más alto de MIP y se realizan los pasos del 4 al 9 del manejador de interrupciones.

Fin interrupción

- El Contador del programa toma el registro MEPC.
- Se procede a recuperar los registros MEPC, MTVAL, MCAUSE y MIE de memoria aumentado el stack pointer

MANEJADOR DE EXCEPCIONES Y INTERRUPCIONES POR EL USUARIO.

En algunas ocasiones se hace necesario que el usuario guarde datos del banco de registro en memoria para que estos no se pierdan cuando ocurre una interrupción o fallo de código. Esta es una de las finalidades de una manejador de excepciones por software, adicional a esto en interrupciones vectorizadas se hace necesario para poder identificar cuál periférico requiere atención.

Un manejador de excepciones es un programa que permite identificar que ha ocurrido en el procesador y saltar a una rutina de tratamiento de esa excepción que ocurrió. Este se hace necesario porque cuando ocurre una excepción y una interrupción el programa salta a la misma dirección (en el caso vectorizado), haciendo necesario que el mismo programa realice una verificación de lo ocurrió y realice un manejo adecuado. Adicionalmente el manejador de excepciones también sirve para realizar el guardado de registros en un parte de la memoria (*Stack pointer de usuario*) en casos que se requiera, hay que recordar que la arquitectura RISC-V provee de una gran cantidad de registros capaces de mantener los datos sin ser modificados durante una interrupción, esto para que las escrituras y lecturas de datos en memoria no sean necesarios.

A continuación, se presentará un manejador de excepciones en el cual se guardan datos a memoria en una posición dada (*stack pointer).*

```
1 addi x29 x0 1600
2 CSRRW x0 0x305 x29 #MTVEC
3 CSRRWI x0 0x300 8   #MSTATUS
4 addi sp x0 2000   #stackpointer de usuario.
```

Figura 24. Activador de interrupciones

```
1  ##interrupcion
2  sw x10 0 x2                    # Almacenar registros que tenia previos
3  csrrw x27 0x342 x27            # Leer causa de excepcion
4  bge x27 x0 .LBB0_6F            # Verificar si es interupcion o si fue error de
5                                 # opcode (causa > 0;error de opcode)
6
7  addi x10 x5 -2040              # Operaciones de la interrupcion
8  slli x10 x5 3
9  slti x10 x5 -2040
10 sltiu x10 x5 -2040
11 xori x10 x5 20
12 srli x10 x5 2
13 srai x10 x5 3
14 ori x10 x5 -21
15 andi x10 x15 -21
16
17 lw x10 0(x2)                   #regresar los registros que estaban
18                                #antes de la interrupcion
19 mret                           # regreso de interrupcion
20
21 .LBB0_6F:
22 csrrw x27 0x342 x27            #manejo de error de opcode
23 mret                           #regreso de opcode
```

Figura 25. Manejo de interrupciones vectorizadas, Programa realizado con [10]

En la Figura 24. se muestra un ejemplo del activador de interrupciones por software, en donde la línea 1 es un auxiliar para realizar operaciones con los registros de estado.

- La línea 1 contiene un valor que será guardado en mtvec, las posiciones de la [31:2] valen 400 que es la dirección en bytes a donde saltan las interrupciones y excepciones en este caso.
- Las líneas de la 2 y 3 realizan la modificación de los registros de control y estado, donde x305 o x300, es la dirección del registro de estado en hexadecimal y el último valor es el registro o el valor que se desea guardar.
    - En la línea 2 se desea almacenar en MTVEC (dirección x305 en hexadecimal) el valor de 1600. MTVEC[31:2]=400 MTVEC[31:2]=0. Recordemos que el procesador es direccionable por bytes, esto significa que el valor de MTVEC[31:2]=400 es a byte, en palabras seria la dirección 100 (400/4).
    - En la línea 3 se guarda un 8 en el registro MSTATUS esto para modificar el bit MSTATUS[3] a 1, el resto de los bits son ignorados, la línea 4 contiene el valor del puntero de pila que se almacena en el registro 2(x2/sp) , este valor es de referencia para realizar el guardado de datos en memoria.

En la Figura 25 se muestra un manejador de interrupciones/excepciones sencillo.

- En la línea 2 se muestra que cuando se interrumpe el procesador lo primero que ejecuta en el manejador de excepciones de software es el almacenamiento los registros que el usuario desee (la recomendación de registros temporales, de stack, etc. se presenta en la Tabla 3), por medio de las instrucciones tipo S (sw,sb,sl). En este caso se guardó un registro (línea 2).
- Seguido a esto se lee y almacena el registro MCAUSE del registro de estados (este registro guarda la causa de la excepción (si es interrupción o excepción)), esta lectura se realiza por medio de las instrucciones tipo I de registros de estado CSRRW (línea 3) en donde el contenido de MCAUSE se almacena en el registro 27.
- Posteriormente, se verifica si se desea realizar una interrupción o una excepción. Como las interrupciones siempre están negadas (el bit más alto del registro MCAUSE es puesto en 1) para

interrupciones, para realizar esta verificación, lo que se realiza es una comparación del contenido del registro MCAUSE, con 0, de esta manera se sabe si fue una interrupción o una excepción, esta operación se muestra en la línea 4 donde se compara el dato previamente leído del registro de estados MCAUSE y se compara con 0. Después de esta comparación se salta a la etiqueta .LBBO_6F: si es excepción (falla de instrucción), o se continua si es interrupción, en ambos casos se ejecutan programas independientes para tomar alguna decisión sobre lo ocurrido (operaciones para manejo de periféricos en interrupción, y operaciones para conocer y decidir que se hacer cuando hay excepción) y posterior mente de ambos estados (excepción e interrupción), se regresa al programa principal con la instrucción MRET.

Nota: para el manejo de subrutinas se realiza un manejador similar para el almacenamiento de los diferentes registros que se necesiten recuperar después de la subrutina. Esto se realiza si es necesario, pero vale la pena considerar que la arquitectura posee una gran cantidad de registros para hacer que el almacenamiento en la pila (*stack pointer*) no sea necesario.

DIRECCIÓN DE INICIO

El procesador está diseñado para que el usuario le indique por medio de la posición 0 de la memoria la dirección de arranque de las instrucciones, esto significa que cada vez que se reinicie el procesador este siempre va a ir a busca a la dirección 0 de la memoria la dirección de inicio del programa.

4.1.6 AHPL

En el ANEXO A se muestra el AHPL (*A Hardware Programing Language*) que es un lenguaje de transferencia de registros que describe el comportamiento de un sistema digital mediante un lenguaje de descripción de hardware. En este caso se describe el funcionamiento del procesador lo que contempla el comportamiento de los diferentes selectores, del banco de registros y de la máquina de control del procesador.

4.1.7 ESQUEMÁTICOS

En este apartado se muestran los esquemáticos de todos los bloques. Se presentan los esquemáticos de los selectores, la ALU, etc. Ver ANEXO B.

4.1.8 VHDL

En el ANEXO C se presenta la descripción del hardware del procesador por medio de VHDL[4], para su desarrollo e implementación (codificación análisis y síntesis) se utilizó la herramienta Quartus II, esto con el fin de verificar que todo el proyecto funcionara sin fallos para evitar posibles bucles o errores en las instrucciones.

Adicional a esta herramienta también se utiliza *Waveform* y *Model Sim Starter -edition* para realizar las diferentes simulaciones.

Ver ANEXO C donde esta el proyecto realizado en Quartus II.

5 PROTOCOLO DE PRUEBAS Y ANÁLISIS DE RESULTADOS

El protocolo de pruebas se diseñó para realizarlos tanto en simulación como en la implementación en hardware. Para realizar el trabajo se utilizó la FPGA DE2-115 Cyclone IV. El reloj de la tarjeta tiene una frecuencia 50 M Hz.(Las pruebas que se realizan de implementación son en un ambiente de pruebas controlado, por lo cual no es critica la elección de la tarjeta). Esta tarjeta posee 40 pines para poder realizar medidas con analizador lógico y realizar conexiones de periféricos.

[4] El lenguaje VHDL es un lenguaje de descripción de *hardware*, creado en el año de 1983, este lenguaje de descripción es estandarizado gracias a la IEEE.

5.1 Simulación.

La simulación se realiza en 3 fases. La primera está comprendida por pruebas a los diferentes bloques del procesador, la segunda está relacionada con pruebas generales al procesador donde se ejecutan algunos programas y se verifican los resultados, los cambios de direcciones, el correcto manejo de las interrupciones, etc. Y la tercera fase está dirigida a pruebas con un programa ejecuta interrupciones.

- En la primera fase se realizaron pruebas de cada uno de los bloques para verificar su funcionamiento y definir los tiempos de retardo de los bloques, y así ajustar los estados en el bloque de control. Estas simulaciones se realizan por medio de la herramienta *waveform* de quartus II.
  Se realizaron pruebas de los selectores, donde se midió para las señales de control el retardo de la salida del selector para la señal seleccionada. Además, se calcularon los retardos de las diferentes operaciones de la ALU: los retardos del banco de registro y del banco de registros de estado. Asimismo, se verificó el correcto cambio de estados en la máquina de control, cambio en las diferentes operaciones, activación de las diferentes señales, habilitadores, estados de manejo de interrupciones entre otros. Todas estas pruebas se presentan en el ANEXO D.
- En la segunda fase se realizó la simulación de todos los bloques en conjunto, la memoria se inicializó por medio de archivos .mif y las señales que se observaron son, entre otras: la salida de la ALU, las direcciones de memoria, los datos que van hacia memoria. Ver ANEXO D, para ver las pruebas que se realizaron. Para realizar esta simulación se utiliza la herramienta Model sim Starter Edition y se hizo necesario el uso de un testbench (archivo de pruebas) donde se indicó en que momentos se hacen cambios de reset, interrupciones entre otros en el procesador.
  Las pruebas que se realizaron tienen una gran cantidad de instrucciones y usaron para verificar los resultados de cada una de estas. Posteriormente se hicieron pruebas para verificar las interrupciones y excepciones, verificar la capacidad de realizar múltiples interrupciones, ejecutarlas de forma correcta y retorno de los parámetros adecuados. Para finalizar, se probó con pequeños códigos en C++ convertidos a RISC-V, esto con ayuda de la herramienta de COMPILER EXPLORER; en estas pruebas se utilizó un código en c++ para comparar dos números, y se verificó que el resultado de esta comparación sea correcto, además se desarrollaron pruebas con subrutinas.
- En la tercera fase se diseñó una pequeña aplicación de domótica, la cual consiste en leer un infrarrojo y cuando este detecte una presencia genera una interrupción para activar un ultrasonido. Posteriormente cuando el ultrasonido a identificado la distancia interrumpe al procesador para que este lea la distancia, la divida entre 58, después convierta el resultado a BCD, para luego convertirlo en ASCII, y visualizar el resultado en una pantalla LCD, donde se mostrará dicha distancia con un texto.

Para la implementación de la aplicación con periféricos se utilizó un sensor de ultrasonido H-SRF05/ H-SRF04, un sensor infrarrojo FC51 y la pantalla LCD integrada en la FPGA DE2-115.

Controladores.

Infrarrojo: Este dispositivo posee 3 terminales, las cuales son VCC, GND y OUT. Se alimenta con 5 V DC. Es activo en bajo, cuando se interrumpe la señal receptora del infrarrojo manda un cero lógico, y cuando no se interrumpe, se envía un uno lógico. El controlador de este dispositivo consta de un estado de espera, un estado de envió de interrupción y espera de interrupción recibida por parte del procesador, y otro estado que sirve para verificar que ya se halla quitado el obstáculo, para no repetir mediciones, en la Figura 26, se presenta el diagrama de flujo del proceso que realiza el controlador.

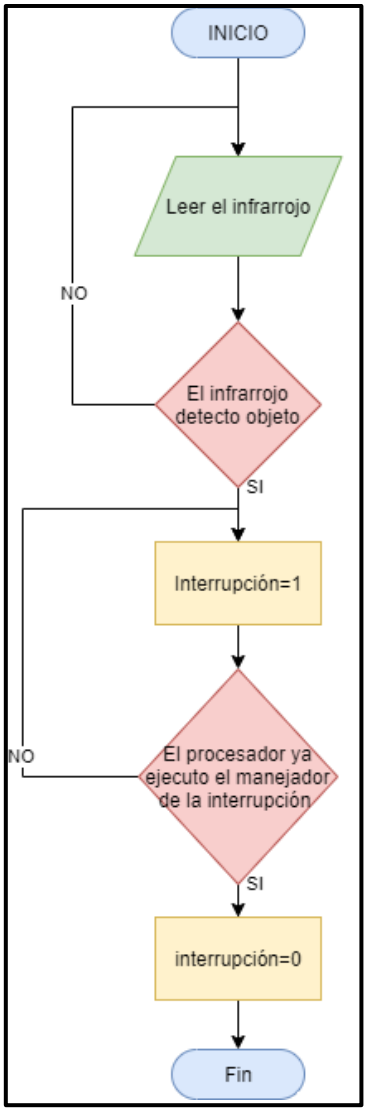


Figura 26. Controlador de infrarrojo.

Ultrasonido. Este dispositivo posee 4 terminales las cuales son VCC, GND, OUT/ECO Y TRIGGER. Se alimenta con 5 V DC. Para que este dispositivo funcione se envía una señal en alto por al menos 10 µs en la terminal TRIGGER. Después el ultrasonido pone en alto la señal de ECO durante el tiempo que se demora en ir rebotar y retornar la señal del ultrasonido, este tiempo que permanece en alto es equivalente a la distancia del objeto. La distancia donde se encuentra el objeto se calcula del tiempo que tarda en ir y regresar la señal del ultrasonido, esta distancia es dependiente de la velocidad del sonido que es de 0.0343 cm/µS, a partir de esta velocidad se calcula el tiempo que se demora en recorrer 1 cm dando como resultado t= 1 [cm] / 0.0343 [cm/µS] t = 29.15 µS como la señal es de ida y vuelta, por lo tanto se demora 58.30 µS por cada centímetro (58.30 µS/cm).[12] A partir de esto se obtiene la ecuación del cálculo de la distancia:

$$Distancia = \frac{TIEMPO\ DE\ LA\ SEÑAL\ ECO\ [\mu S]}{58\ [\mu S/cm]} = \frac{TIEMPO\ DE\ LA\ SEÑAL\ ECO}{58}[cm]\ (1)$$

Debido a estas consideraciones el controlador de este sensor consta de recibir una señal externa que le indica cuando se debe enviar la señal de lectura (TRIGGER), posterior se realiza la espera y el conteo de la señal OUT/ECO, seguido a esto se provoca una interrupción en el procesador y se espera que el procesador lea el resultado del conteo de eco. El proceso que realiza este controlador se muestra en el diagrama de flujo en la Figura 27.

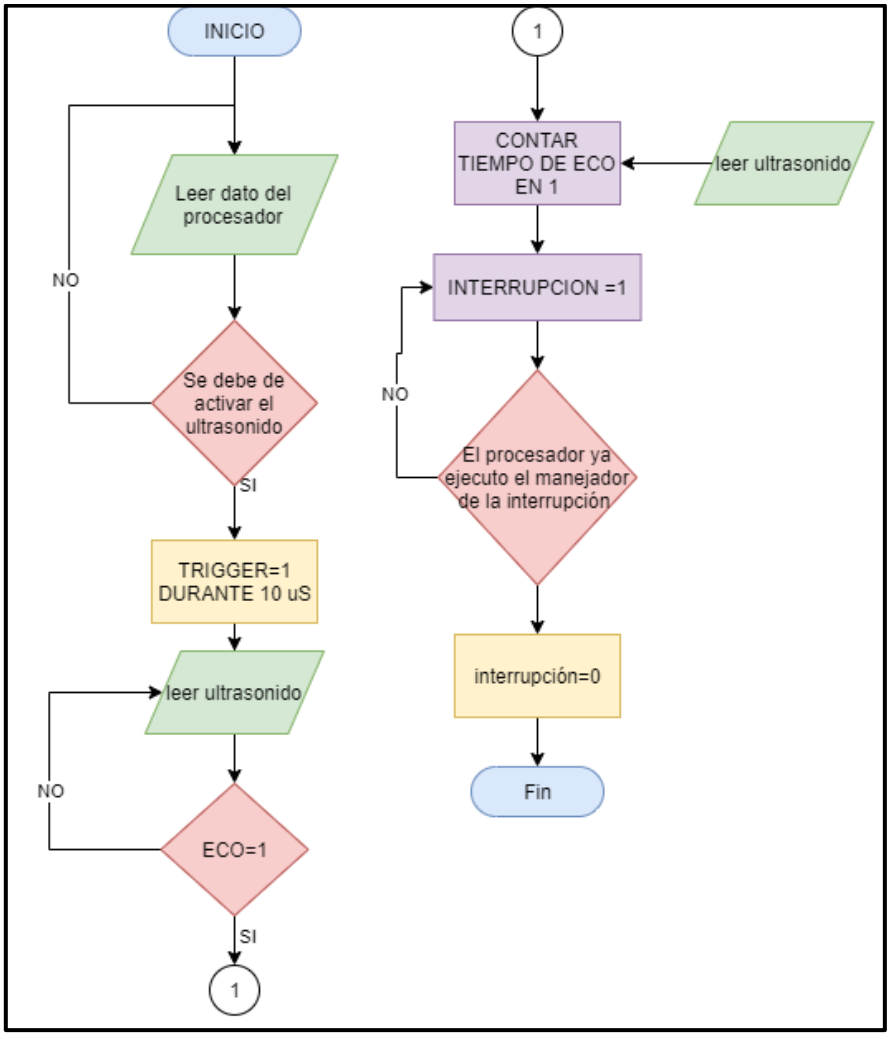


Figura 27. Controlador de ultrasonido.

LCD 16x2: Este dispositivo viene integrado en la FPGA, posee una señal de E (enable), una de escritura/lectura (RW) y una de seleccionar si es dato o un registro (RS). Adicionalmente a estas posee 8 entradas para identificar el dato o dirección a modificar. Es controlado por un microprocesador HD44780U al cual se le deben enviar diferentes instrucciones, para que se active y escriba de datos en la pantalla. Estas instrucciones las indica el fabricante de la pantalla y se muestran en el datasheet del microprocesador [13]. En el ANEXO E se presentan específicamente las instrucciones utilizadas y otras consideraciones que se tuvieron en cuenta para su uso en la aplicación.

El controlador de este dispositivo debe de mantener los tiempos de las señales el tiempo necesario para que estas sean leídas de forma correcta por el microprocesador propio de la pantalla.

Controlador de interrupciones: Este bloque se encarga de controlar las interrupciones, priorizar y organizar un vector de interrupciones para ser leído por el procesador (interrupciones vectorizadas) y de retornar la respuesta del procesador al periférico que la causó. El proceso que realiza este controlador se muestra en el diagrama de flujo en la Figura 28.

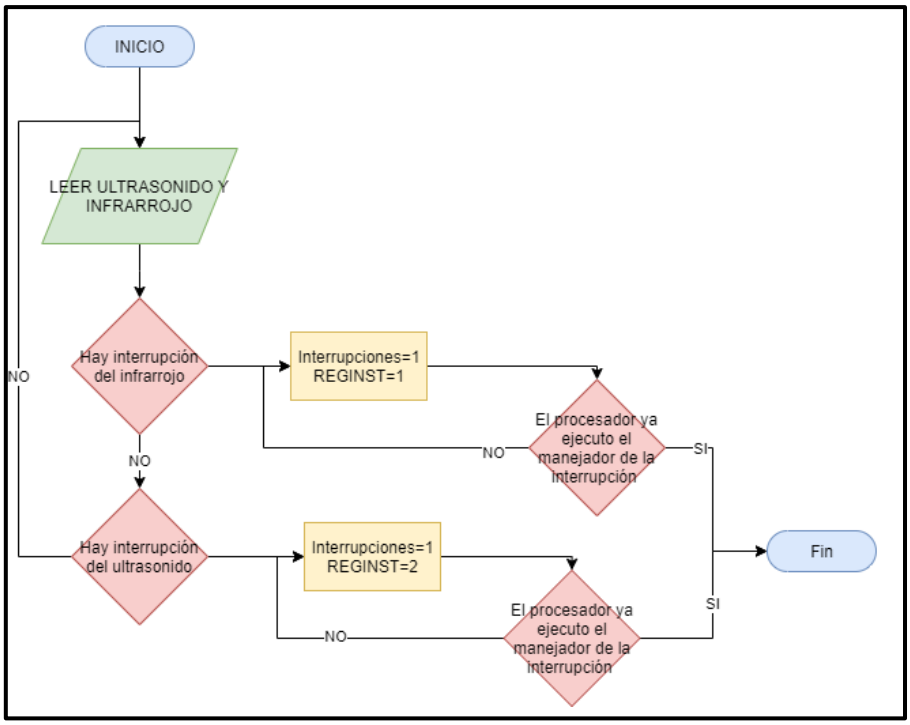


Figura 28 Controlador del control de interrupciones.

De este bloque hay una segunda versión el cual consta de manejar las interrupciones auto vectorizadas, en este el bloque se encarga de activar un nivel de interrupción diferente para los bloques y de regresar la respuesta de la interrupción a los periféricos.

Mapeado de memoria: Este bloque se encarga de distribuir las direcciones para la memoria que se utiliza para el programa y las direcciones que se utilizan para los periféricos. Dependiendo de las direcciones este bloque define hacia o de donde viene la información, y habilita ese dispositivo para leer o escribir información. Esta información se presenta en la Tabla 13.

MAPEO DE MEMORIA		
DIRECCIONES	USO	DISPOSITIVO
0-1023	PROGRAMA	Memoria del programa.
1024	Quien hizo la interrupción	REGISTRO DE PERIFÉRICOS
1025	ULTRASONIDO	ULTRASONIDO
1026	LCD	LCD

Tabla 13. Mapeo de memoria.

Programa:

El programa consta de dos partes la primera el programa principal y la segunda el manejador de interrupciones, el diagrama de flujo de estos se muestra en la Figura 29.

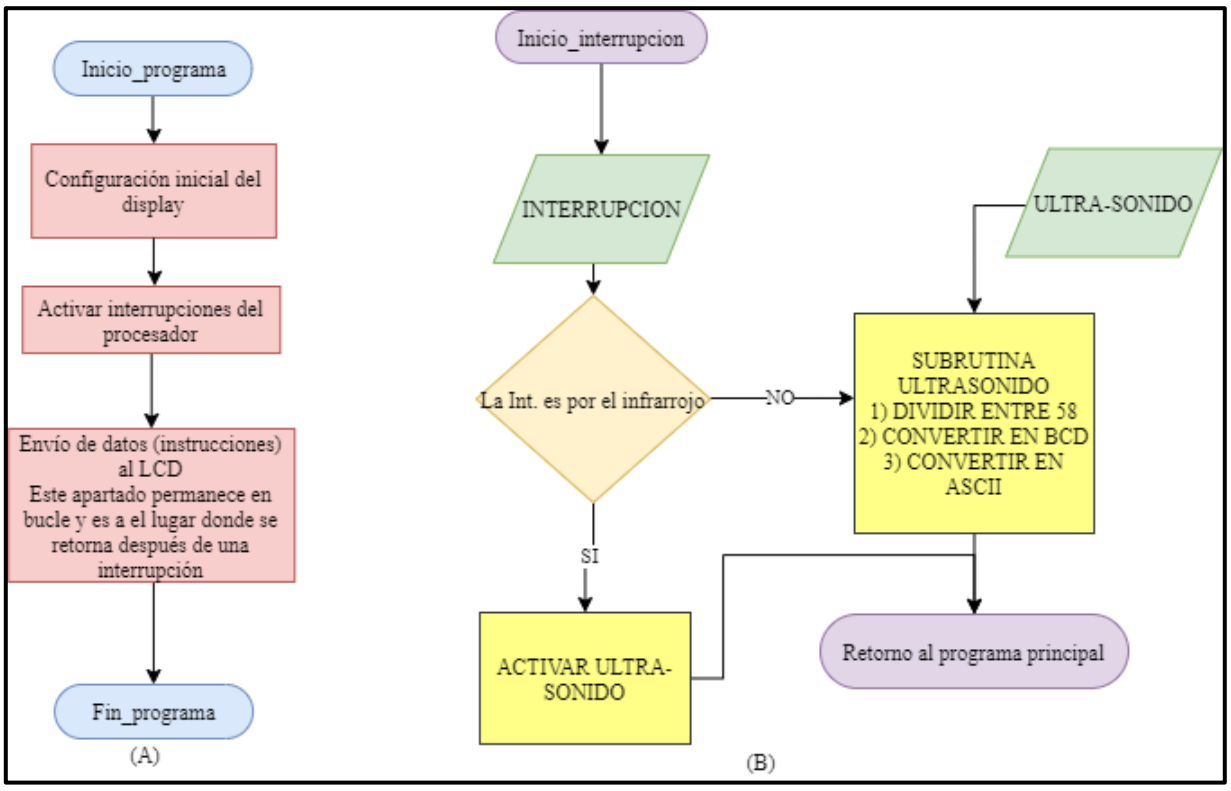


Figura 29 Diagrama de flujo de la fase 3 de simulaciones.

En el apartado (A) de la Figura 29 se presenta el programa principal, el cual consta de una configuración inicial del display, encenderlo, limpiar datos anteriores, etc, posteriormente se activan las interrupciones (esto se realiza como se explicó en la sección 4.1.5) y finalmente se escriben los datos en el LCD, se escribe la palabra DISTANCIA: D1 D2 D3 cm, donde D1,D2 y D3 son el valor de la lectura del ultrasonido procesados para ser mostrados en la pantalla LCD, estos datos se encuentran almacenados en 3 registros del banco de registros del procesador (usados exclusivamente para esta tarea). Este proceso de escritura de datos es de manera cíclica, lo que significa que esta parte del programa se mantiene en constante ejecución y es en la que ocurren y retornan las interrupciones.

En el apartado (B) de la Figura 29 se presenta el programa de la interrupción en el caso vectorizado, la cual consta de:

- Leer la interrupción.
- Verificar quien causó la interrupción y realizar la ejecución del manejador de interrupción de dicho sensor. Si la interrupción fue generada por el infrarrojo se procede a enviar la señal de activación a él ultrasonido. Si la interrupción fue generada por el ultrasonido, el procesador lee el valor del ultrasonido, este valor lo divide entre 58 (por software), posteriormente lo convierte en BCD y le suma 48 para convertirlo en ASCII este resultado lo deja almacenado en 3 registros del banco de registros del procesador, y posteriormente regresa a el programa principal de la aplicación.

El programa implementado y las respectivas simulaciones se presentan en el ANEXO E, adicionalmente se muestran los códigos correspondientes a el manejo del controlador de interrupciones, mapeo de memoria y controladores de forma vectorizada y auto vectorizada en los ANEXOS E.1 meddistancia (vectorizado) y ANEXO E.2 meddistancia (auto vectorizados)

5.2 Implementación.

- Por medio del analizador lógico se verifican las señales en hardware de la implementación del procesador y de los módulos de la aplicación con periféricos, donde se realizaron las mismas pruebas hechas en simulación. A continuación, se presentan sus resultados. En el ANEXO F se muestran en mayor detalle las pruebas completas.
  Algunas pruebas realizadas son:

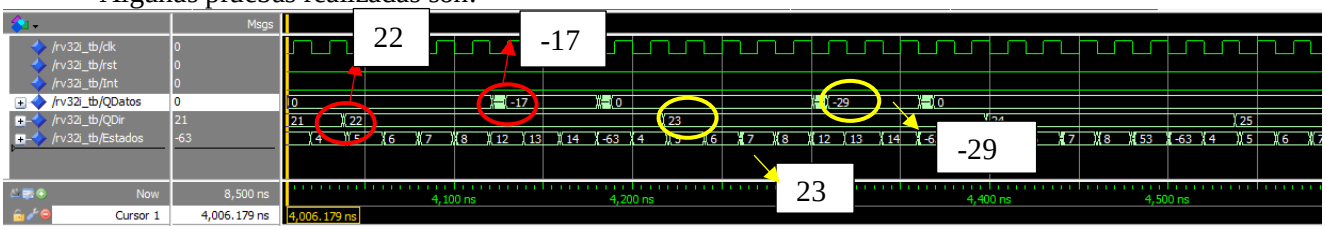


Figura 30. Resultado de simulación instrucciones ORI y ANDI.

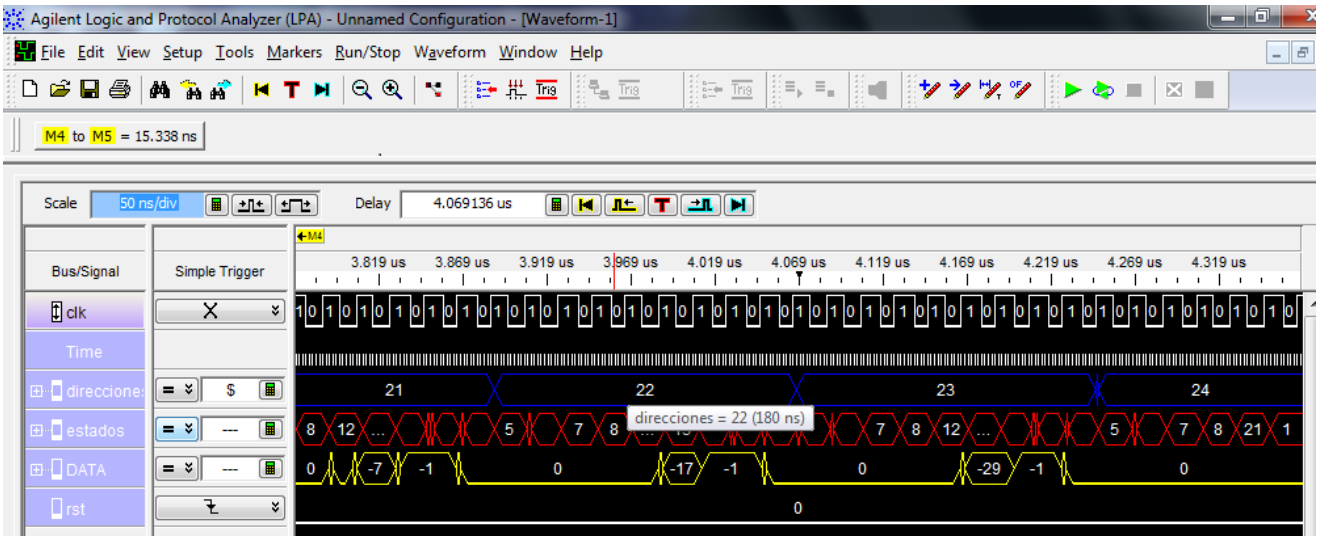
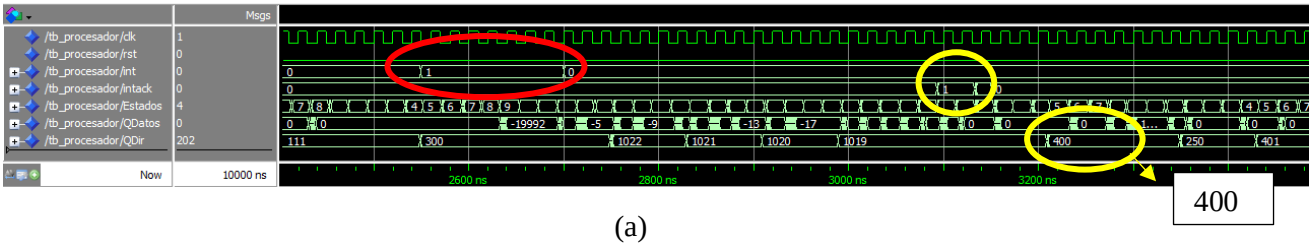


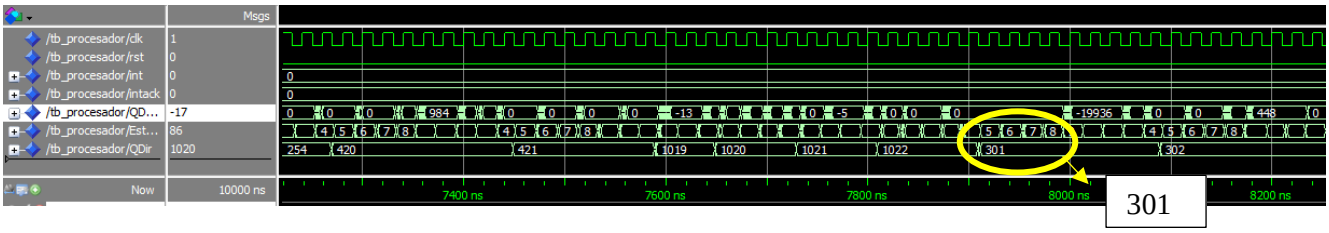
Figura 31. Resultado de implementación instrucciones ORI y ANDI.

En las Figura 30 y Figura 31 se muestra el resultado de la operación ori (encerrada en rojo y dirección 22 en azul respectivamente en las figuras) entre -50 y -21 dando como resultado -17 (resultado observado en rojo y en amarillo respectivamente en las figuras). En la dirección 23 (encerrada en amarillo y dirección 23 en azul respectivamente en las figuras) se presenta el resultado de un ANDI (operación and entre un registro y un valor inmediato) entre -25 y -21, dando como resultado -29. Estas operaciones son tipo I, al comparar los resultados de la implementación Figura 31 con los de simulación Figura 30 se muestra que los resultados son iguales y correctos lo que permite deducir que las operaciones tipo I se realizan de forma correcta.

PRUEBAS DE INTERRUPCIONES.

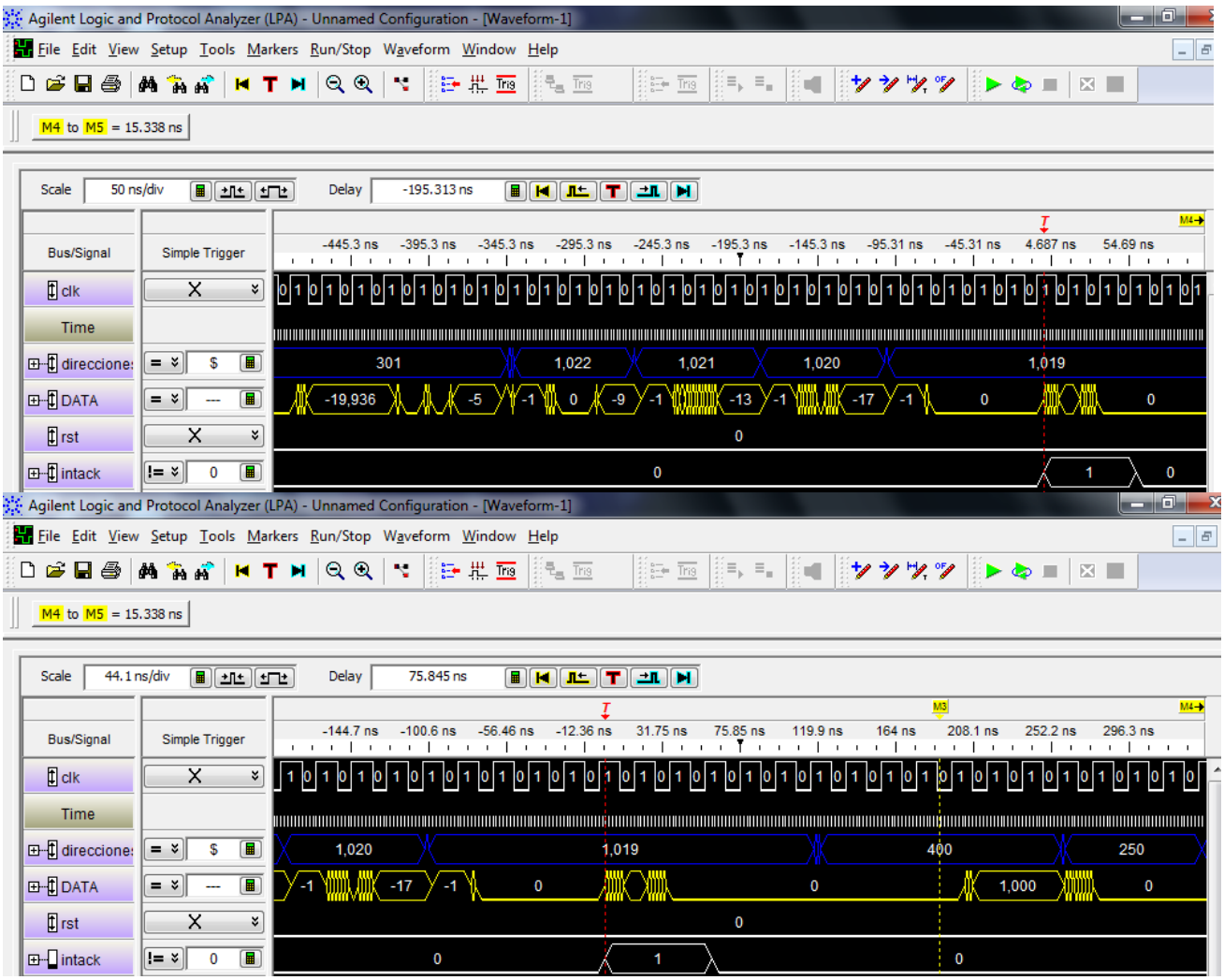


(a)

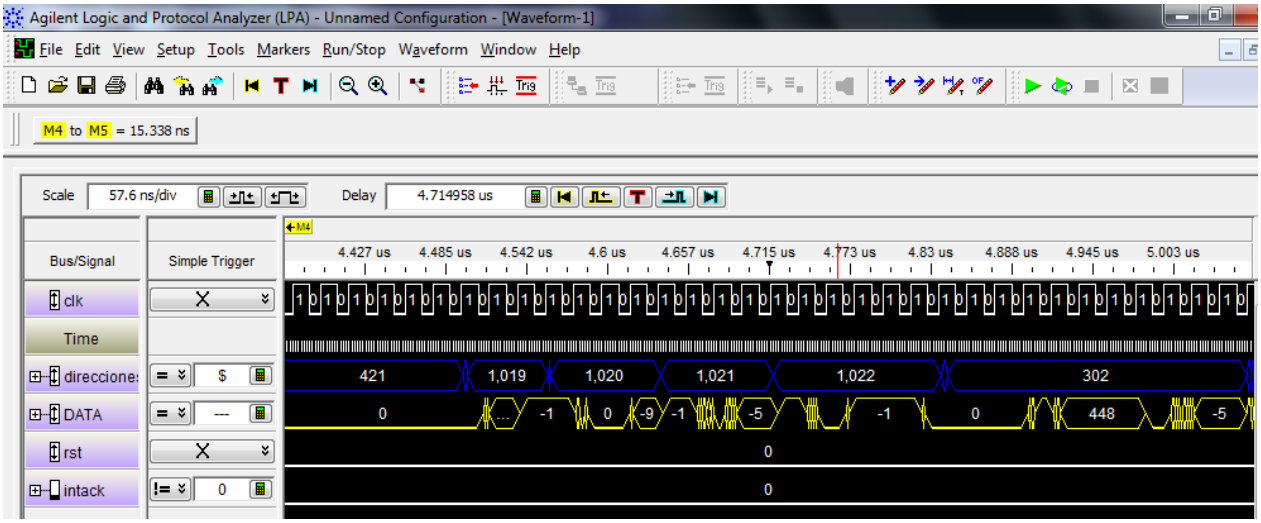


(b)

Figura 32. Interrupciones en simulación.



(a)



(b)

Figura 33 Interrupciones en implementación.

En las Figura 32 y Figura 33, parte (a) se muestra la activación de la interrupción (en rojo) en la dirección 300, en simulación y 301 en implementación, como el procesador responde que la interrupción fue activada con la señal IntAck (en amarillo Figura 32 y en blanco a IntAck Figura 33), por alrededor de 60 ns, y después se muestra cómo salta a ejecutar la interrupción ubicada en la posición 400 de la memoria.

En las Figura 32 y Figura 33 ,parte (B) se muestra el regreso de la interrupción a la dirección 301 (en simulación), 302 (en implementación), se presenta el correcto retorno del stack y evidenciamos el correcto resultado de la operación ubicada en la dirección 302 (en azul) la cual tiene como resultado 448, con esta prueba se muestra el correcto funcionamiento de las interrupciones, del manejador de excepciones presentado en el ANEXO D.

FIBONACCI.

En esta prueba se realiza la secuencia de FIBONACCI, esta secuencia se encarga de sumar los dos resultados anteriores, iniciando con los valores de 0,1. La secuencia inicial es 0,1,1,2,3,5,8,13,21…, la secuencia hasta la iteración 23 se muestra en la Figura 36.

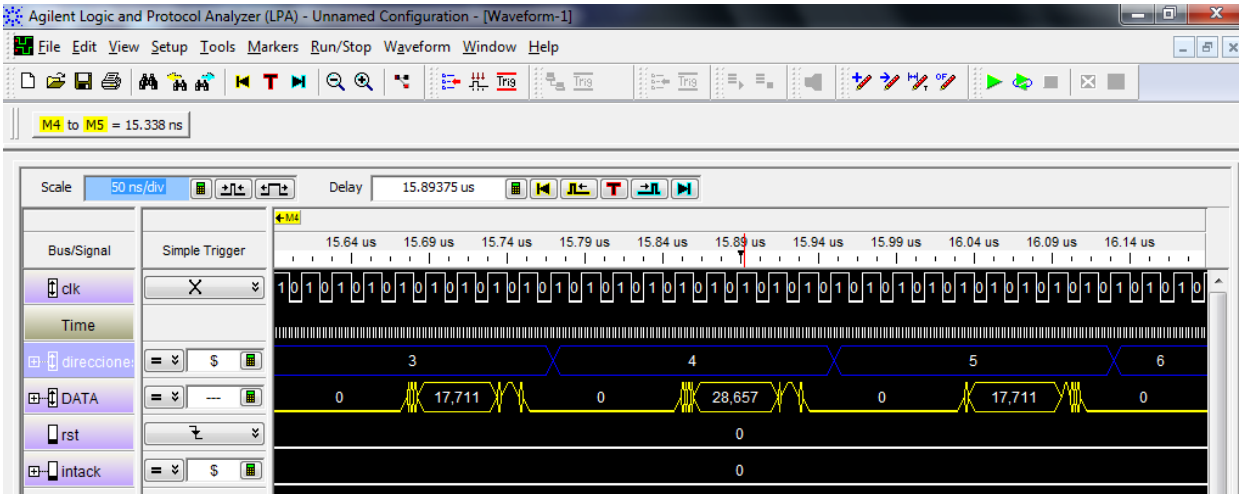


Figura 34. Resultados Implementación de FIBONACCI.

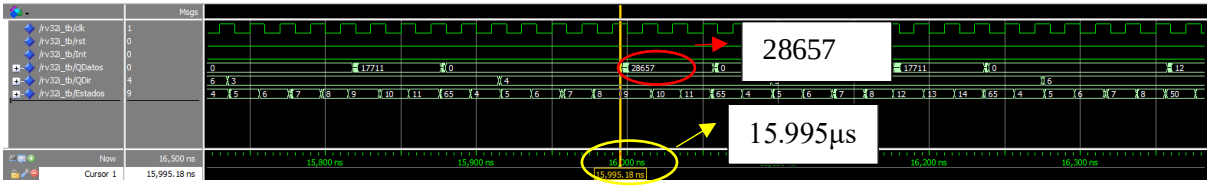


Figura 35. Resultados simulación de FIBONACCI

En las Figura 35 y Figura 34 se muestra el resultado de la secuencia de Fibonacci. Se muestra que para alcanzar la iteración 23, duro 15,85 µS en implementación Figura 34, este resultado se compara con los resultados de simulación Figura 35 , el resultado de la simulación para la misma iteración (23) tiene como tiempo a 15.995µs, la diferencia de tiempo que se observa es debido a que en la simulación se inicia a contabilizar desde que el reset esta en 1 (aproximadamente 100 ns), en la implementación el analizador lógico empieza el tiempo desde que se baja el reset. Se observa que para iteraciones largas el procesador funciona de forma correcta.

0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55
11	89
12	144
13	233
14	377
15	610
16	987
17	1597
18	2584
19	4181
20	6765
21	10946
22	17711
23	28657

Figura 36. Iteraciones de Fibonacci.

A continuación, se presentan algunos de los resultados de las pruebas realizadas.

En las Figuras de la Figura 37 a la Figura 42 se presentan los resultados de la implementación, en estas se presentan los resultados en la pantalla LCD y se pueden comparar por medio de una regla, para verificar que los resultados fueron cercanos al real (la aplicación no mide menos que un centímetro).

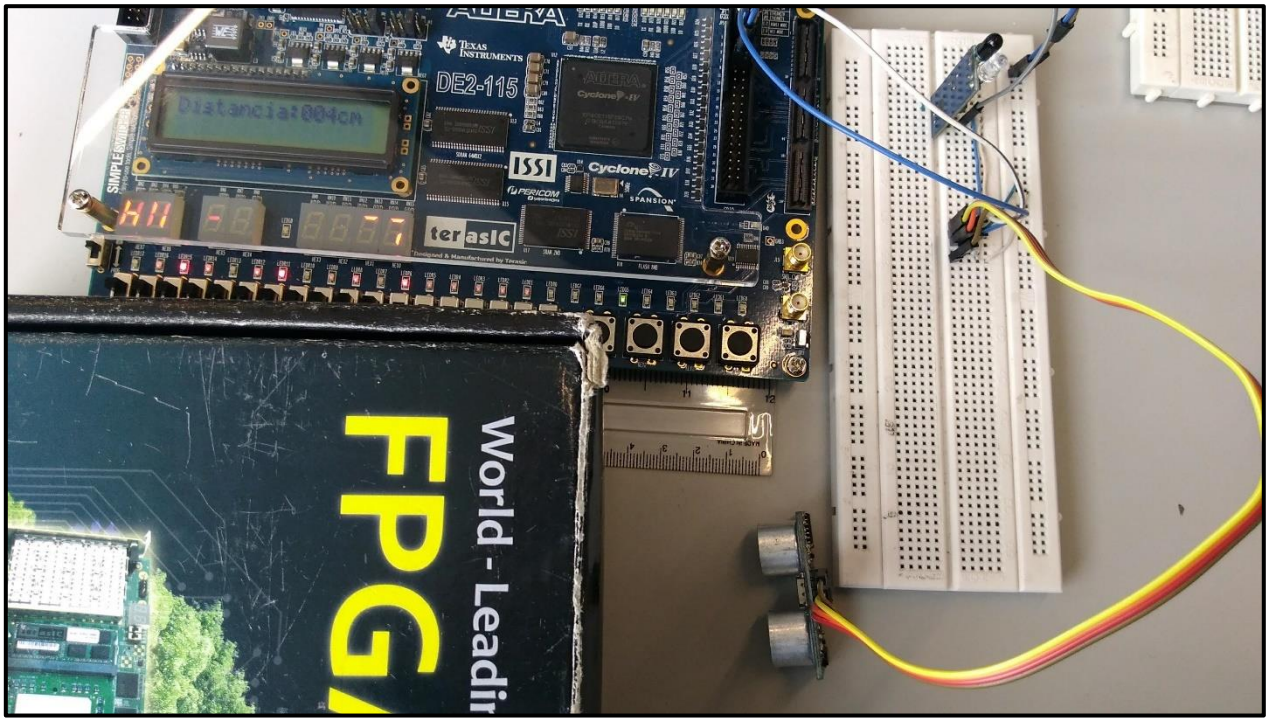


Figura 37. Resultados de implementación 1.

En la Figura 37, se presenta la medición de 4 cm, se presenta que en la regla esta la medida cercana los 4 cm que es el dato que se presenta en el LCD, se presenta que el procesador tuvo un manejo óptimo de los periféricos, y el programa que se utilizó para el control de estos.

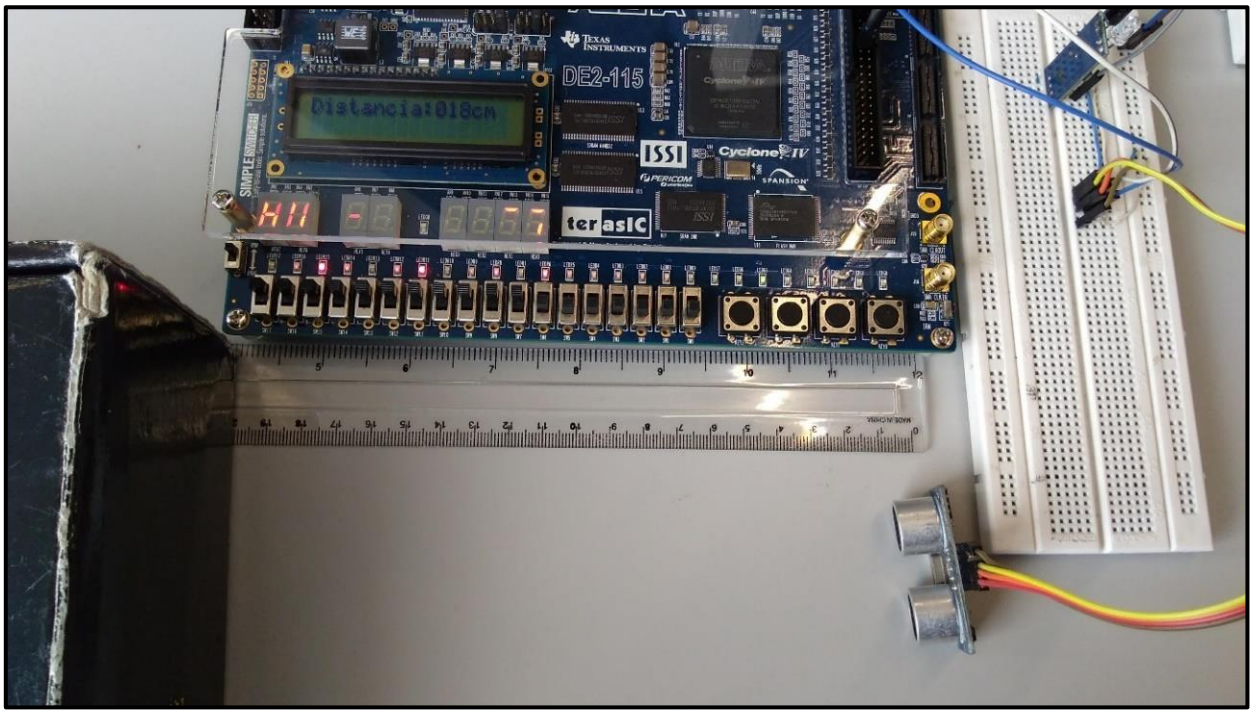


Figura 38. Resultados de implementación 2.

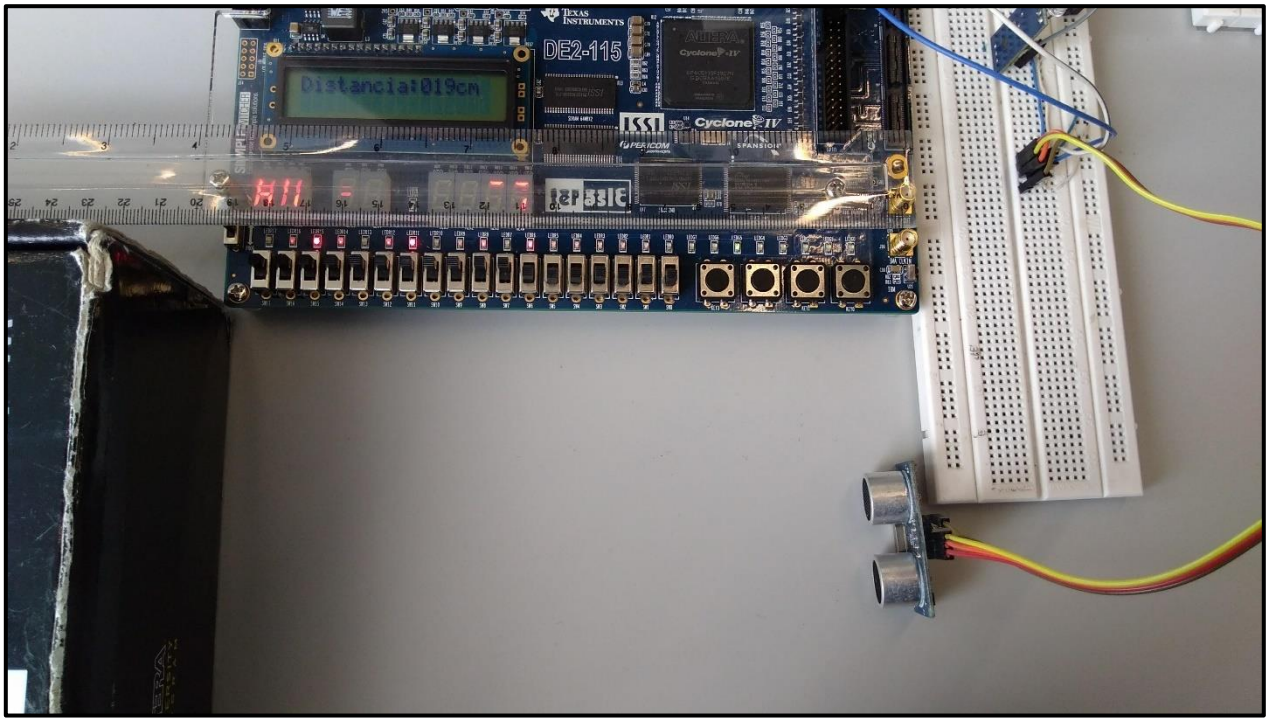


Figura 39. Resultados de implementación 3.

En la Figura 38 y Figura 39, se presenta la medición de 18 cm, se presenta que en la regla esta la medida cercana los 19 cm, pero se observa que el ultrasonido se ubica sobre un 1 cm por lo que al restar da el valor esperado que es el dato que se presenta en el LCD.

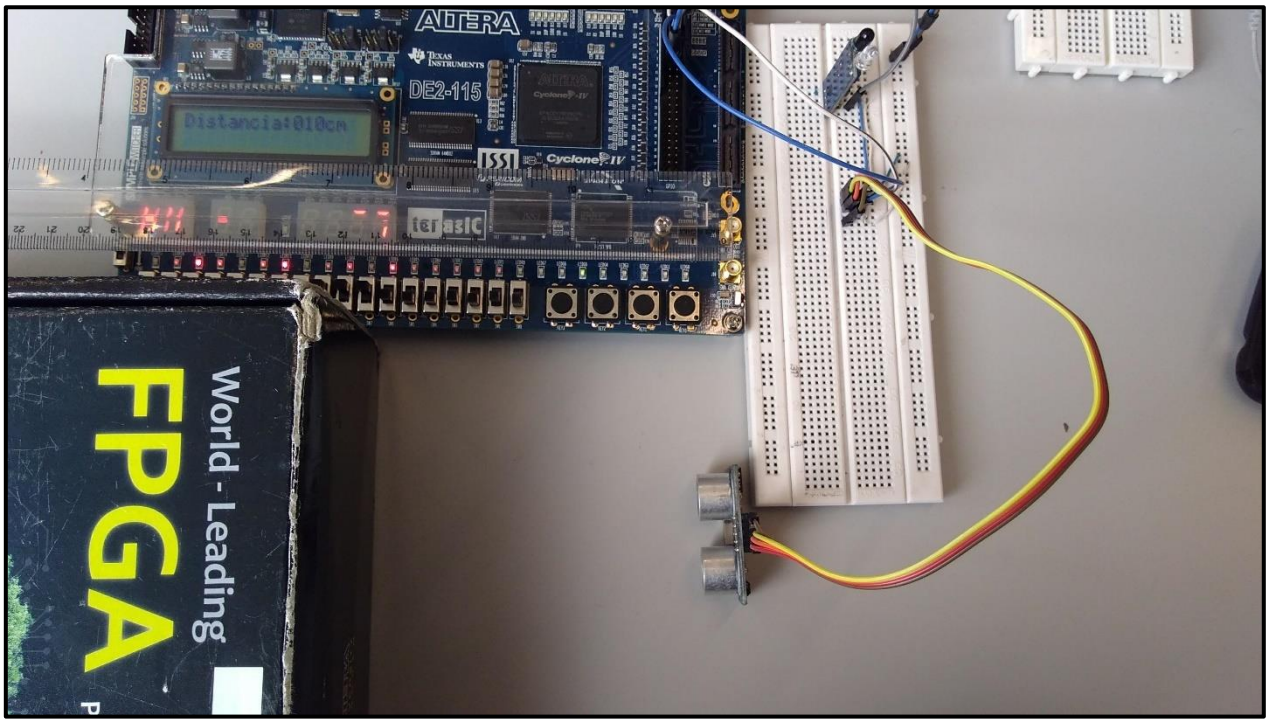


Figura 40. Resultados de implementación 4.

En la Figura 40, se presenta la medición de 10 cm, se presenta que en la regla esta la medida cercana los 11 cm, y el ultrasonido se ubica desde 1 cm por lo tanto la medida observada en el LCD es acertada.

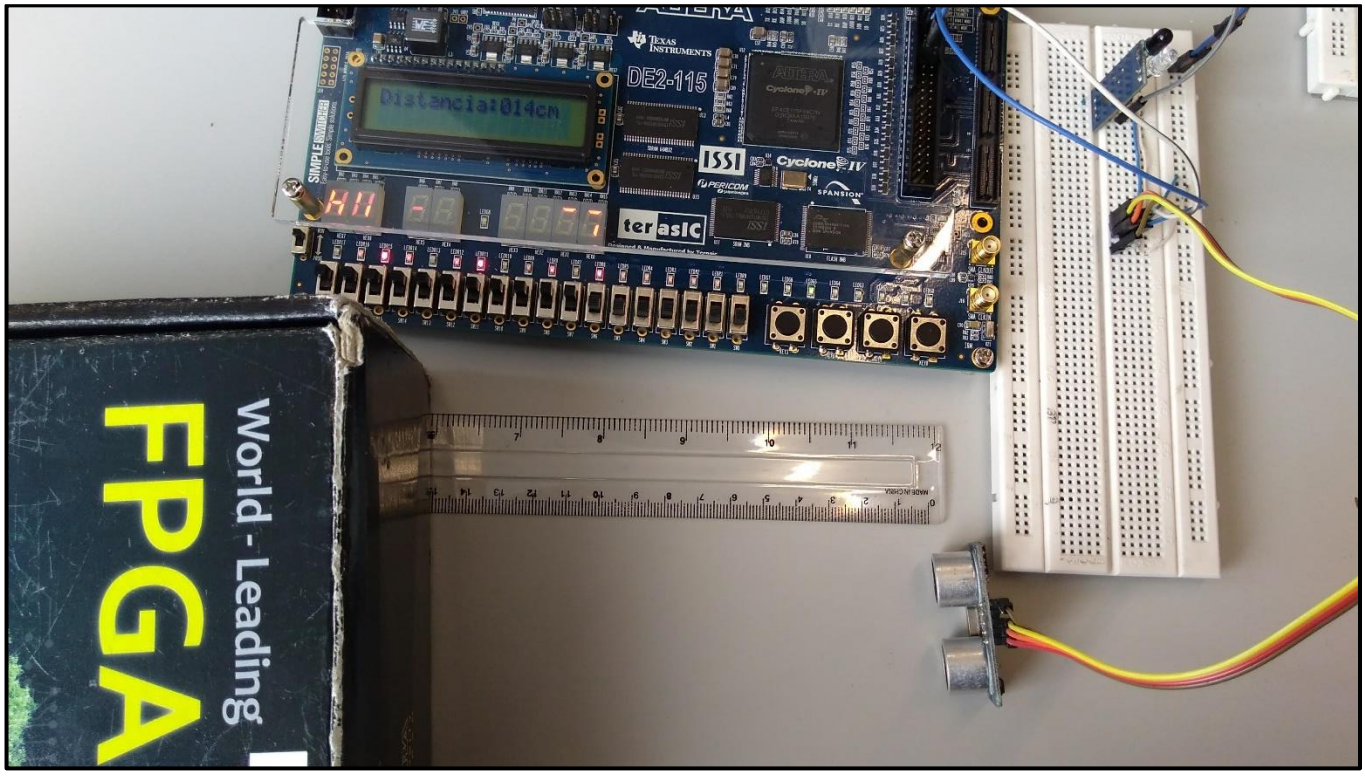


Figura 41. Resultados de implementación 5.

En la Figura 41, se presenta la medición de 15 cm, se presenta que en la regla esta la medida cercana los 15 cm y el dato que se presenta en el LCD es de 14 cm, por lo que la medida se acerca al valor real, estas variaciones se deben a que el programa de prueba está diseñado para mediciones mínimas de un cm, por lo cual se pueden presentar estas pequeñas inexactitudes.

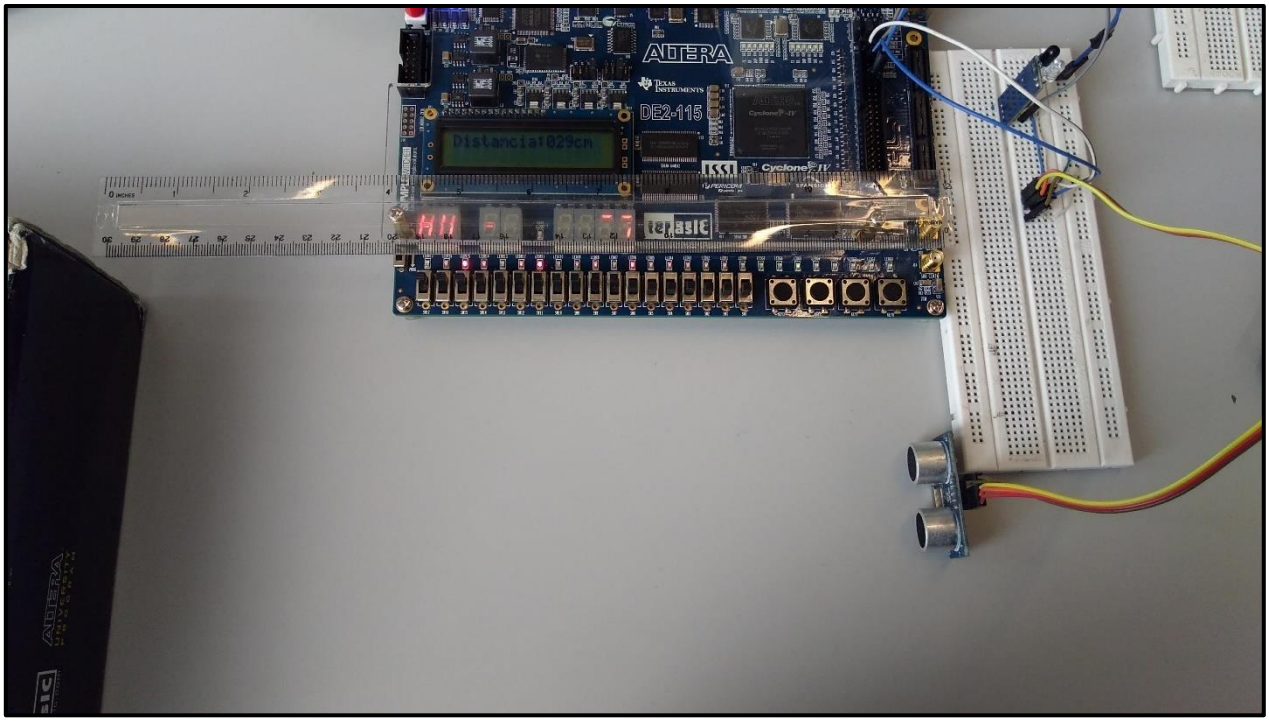


Figura 42. Resultados de implementación 6.

En la Figura 42, se presenta la medición de 28 cm, se presenta que en la regla esta la medida cercana los 29 cm el dato que se presenta en el LCD es de 29 cm, que es cercando al esperado y está en el rango de medición.

La implementación final del procesador sobre la FPGA Cyclone IV posee las siguientes características

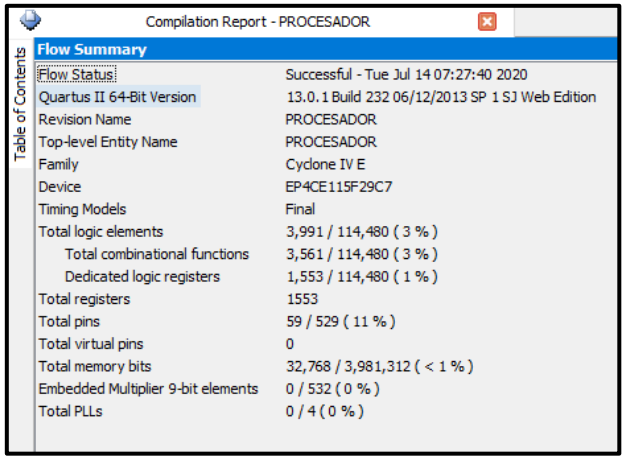
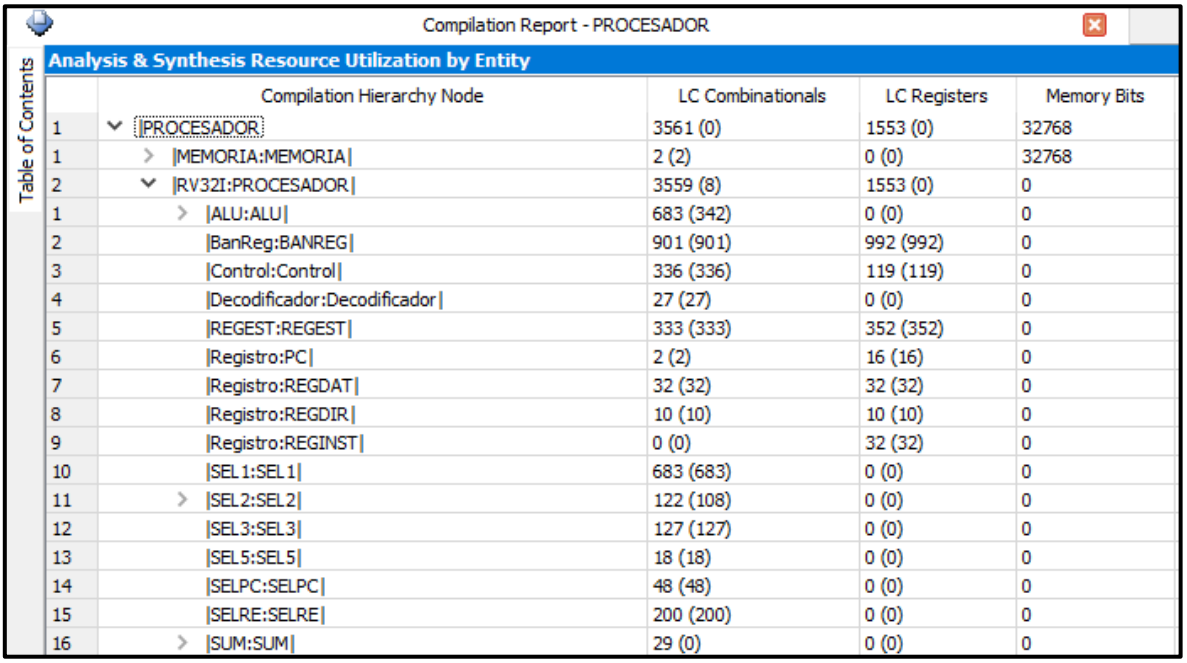


Figura 43. Características finales del procesador 1.



	Compilation Hierarchy Node	LC Combinationals	LC Registers	Memory Bits
1	|PROCESADOR	3561 (0)	1553 (0)	32768
1	|MEMORIA:MEMORIA|	2 (2)	0 (0)	32768
2	|RV32I:PROCESADOR|	3559 (8)	1553 (0)	0
1	|ALU:ALU|	683 (342)	0 (0)	0
2	|BanReg:BANREG|	901 (901)	992 (992)	0
3	|Control:Control|	336 (336)	119 (119)	0
4	|Decodificador:Decodificador|	27 (27)	0 (0)	0
5	|REGEST:REGEST|	333 (333)	352 (352)	0
6	|Registro:PC|	2 (2)	16 (16)	0
7	|Registro:REGDAT|	32 (32)	32 (32)	0
8	|Registro:REGDIR|	10 (10)	10 (10)	0
9	|Registro:REGINST|	0 (0)	32 (32)	0
10	|SEL1:SEL1|	683 (683)	0 (0)	0
11	|SEL2:SEL2|	122 (108)	0 (0)	0
12	|SEL3:SEL3|	127 (127)	0 (0)	0
13	|SEL5:SEL5|	18 (18)	0 (0)	0
14	|SELPC:SELPC|	48 (48)	0 (0)	0
15	|SELRE:SELRE|	200 (200)	0 (0)	0
16	|SUM:SUM|	29 (0)	0 (0)	0

Figura 44 Características finales del procesador 2.

En las figuras 40 y 41 se presentan la cantidad de elementos lógicos por bloques del procesador, se muestra que son una cantidad mínima de elementos lógicos 3991 en total. (Ver Figura 43), lo que permite implementar otras aplicaciones de domótica junto al procesador. Adicional a esto la máquina de estados final constó de 118 estados codificada en formato one hot (un flip-floop por estado).

6 CONCLUSIONES Y RECOMENDACIONES

6.1 CONCLUSIONES.

- Se diseño un procesador basado en la arquitectura RISC-V, con instrucciones que son de utilidad para aplicaciones de domótica, logrando un procesador pequeño *open source*.

- Se diseño la adaptación de la arquitectura para recibir múltiples niveles de interrupción, lo que permite mayor manejo de periféricos en aplicaciones de domótica.

- Se implementó el procesador basado en la arquitectura RISC-V. Su correcto funcionamiento se presentó con el analizador lógico, para los diferentes programas realizados.

- Se implemento una aplicación de domótica, en la cual se evidencian la respuesta del procesador ante las diferentes pruebas, tratamiento de datos, manejo de interrupciones vectorizadas y auto vectorizadas, manejo de periféricos mapeados en memoria.

- Debido a la situación de confinamiento de la ciudad por el COVID19 no se pudieron hacer todas las pruebas en hardware que se tenían planteadas, pero los resultados de las que se realizaron dan muy buenas expectativas.

- Se analiza la arquitectura y se ve que, al tener instrucciones de comparación y salto, estas tareas se facilitan respecto a otras arquitecturas en las cuales se necesitan dos instrucciones o más para realizarla.

6.2 TRABAJOS A FUTURO.

- Implementar un sistema completo de domótica, utilizando procesador diseñado.

- Implementar algunas optimizaciones como lo puede ser *pipelined*, para mayor velocidad, y *Rolling up the pipeline*, para reducir el tamaño del área a el procesador.

7 BIBLIOGRAFÍA.

[1] "¿Qué es el Open Source y cuál es su importancia en la actualidad?," *Universia.net*, 2014. [Online]. Available: https://noticias.universia.pr/en-portada/noticia/2014/04/04/1093711/que-es-open-source-cual-es-importancia-actualidad.html.

[2] Domotizar, "Importancia de la domótica en la actualidad," *Domotizar*. [Online]. Available: https://www.domotizar.com/importancia-de-la-domotica-actualmente/. [Accessed: 05-Mar-2020].

[3] "RISC-V." [Online]. Available: https://riscv.org/.

[4] W. STALLINGS, *Organización y arquitectura de computadores*, 7 edicion. 2006.

[5] Pedro J . Madero Serrano, "DOMÓTICA Y APLICACIONES PARA EL HOGAR," p. 14.

[6] D. Patterson and A. Waterman, "GUIA PRACTICA DE RISC-V," vol. 1.0.5, 2018.

[7] H. H. Porter, *RISC-V: An Overview of the Instruction Set Architecture*. 2018.

[8] A. Waterman, K. Asanovi, and SiFive Inc., "The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA," vol. I, 2019.

[9] A. Waterman and K. Asanovi, "The RISC-V Instruction Set Manual V 1.12," *2012 IEEE Int. Conf. Ind. Technol. ICIT 2012, Proc.*, vol. II, 2012.

[10] M. B. Petersen, "RIPES." p. https://github.com/mortbopet/Ripes, 2020.

[11] B. Landers, "RARS." .

[12] Ingeniería de Microsistemas Programados S.L, "HOJA TECNICA: Medidor ultrasónico SRF05." España.

[13] HITACHI, "HD44780U (LCD-II)." .

8 ANEXOS.

En el siguiente enlace se encuentra el repositorio con todos los documentos anexados.

https://livejaverianaedu-my.sharepoint.com/:f:/g/personal/edwin_barbosa_javeriana_edu_co/EoRxCMLArkVEr1FLVmKxV4wBgnVWBHhGC4wEjqVHCzAbvQ?e=sYGWoO

ANEXO A: AHPL

ANEXO B: ESQUEMÁTICOS

ANEXO C: VHDL (PROYECTO HECHO EN QUARTUS II)

ANEXO D: SIMULACIONES FASE I Y II

ANEXO E: SIMULACIONES DE LA IMPLEMENTACIÓN DE PERIFÉRICOS.

ANEXO E.1: PROYECTO PARA MEDICIÓN DE DISTANCIA CON INTERRUPCIONES VECTORIZADAS

ANEXO E.2: PROYECTO PARA MEDICIÓN DE DISTANCIA CON INTERRUPCIONES AUTO VECTORIZADAS.

ANEXO F: PRUEBAS CON ANALIZADOR LÓGICO.