Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Programa de Maestría en Electrónica

Implementación de un microprocesador de aplicación específica para la ejecución del algoritmo de modelos ocultos de Markov en el reconocimiento de patrones acústicos.

Documento de tesis sometido a consideración para optar por el grado académico de Maestría en Electrónica con Énfasis en Sistemas Empotrados.

Carlos Adrián Salazar García

Cartago, 09 de diciembre, 2015

Declaro que la presente tesis de Maestría ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Carlos Adrián Salazar García

Cartago, 09 de diciembre, 2015

Céd: 5-0379-0120

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Tesis de Maestría
Tribunal Evaluador

Tesis de Maestría defendida ante el presente Tribunal Evaluador como requisito para optar por el título de Máster en Electrónica con Énfasis en Sistemas Empotrados del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

MSc. Eduardo Adolfo Canessa Montero
Profesor Lector

Dr. Juan Luis Crespo Mariño
Profesor Lector

Dr. Alfonso Chacón Rodríguez
Profesor Asesor

Los miembros de este Tribunal dan fe de que la presente Tesis de Maestría para optar por el grado de Máster en Electrónica con Énfasis en Sistemas Empotrados, ha sido aprobada con mención honorífica *SUMMA CUM LAUDE* y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Cartago, 09 de diciembre, 2015

Resumen

En este trabajo, se realizó la unidad de clasificación para un sistema de reconocimiento de patrones acústicos que permita la detección de disparos y motosierras en el bosque, basado en el algoritmo de modelos ocultos de Markov (HMM), con la idea de más adelante construir una red inalámbrica de sensores de vigilancia con nodos similares. Esta técnica es considerada, una de las más poderosas en la clasificación de audio en la actualidad. La misma, se implementó mediante una solución que integra hardware y software en una FPGA. Para ello, se desarrolló un microprocesador de aplicación específica en un lenguaje de descripción de hardware y el software en el lenguaje de programación C. Se obtuvo una tasa de reconocimiento del 90.33% para motosierras y 85.43% para disparos. Además, se muestran los resultados de la verificación funcional contra el modelo de referencia utilizando datos reales tomados de un área de bosque tropical protegido. Se presentan a su vez, resultados Post Place & Route sobre: la cantidad de recursos utilizados, frecuencia máxima de operación y consumo de potencia.

Palabras clave: ASP, FPGA, HDL, HMM, microprocesador, RISC-V.

Abstract

In this work, the classification unit of an acoustic pattern recognition system based in the algorithm of Hidden Markov Models (HMM) was developed for the detection of gunshots and chainsaws in a protected tropical area, with the idea of later building a surveillance wireless sensor network with similar nodes. The HMM was implemented with a solution that integrated hardware and software. For this reason, an application-specific microprocessor was implemented and testing in a FPGA. The recognition rate obtained was: 90,33% for chainsaw and 85,43% for gunshot. Results of the functional verification compared with to reference model are shown. Some post-place-and-route results also are presented: the device utilization summary, operating frequency and power consumption.

Keywords: ASP, FPGA, HDL, HMM, microprocessor, RISC-V

a mi querida familia

Agradecimientos

En primer lugar a Dios porque él es el que me da la fuerza y me guía para poder salir adelante ante diferentes situaciones que se presentan a lo largo del camino.

A mis padres por ser el pilar fundamental de mi vida y por haberme enseñado a ser la persona que soy. Por toda la educación que me brindaron tanto académica, como de la vida, por su apoyo perfectamente mantenido a través del tiempo.

Al Ing. Alfonso Chacon por toda la colaboración brindada, durante la elaboración de la tesis y por todos los buenos consejos brindados.

Y finalmente a esa persona especial que siempre estuvo ahí, mi amada esposa Cindy que durante estos dos años ha sabido apoyarme para continuar y nunca renunciar, gracias por su amor incondicional y por toda la ayuda brinda durante la maestría.

Carlos Adrián Salazar García

Cartago, 09 de diciembre, 2015

Índice general

Índice de figuras

Índice de tablas

Lista de símbolos y abreviaciones

Abreviaciones

ADC	Convertidor analógico a digital
ALU	Unidad aritmético lógica
ASIC	Circuito integrado de aplicación específica
ASP	Procesador de aplicación específica
CISC	Conjunto de instrucciones complejas
CPI	Ciclos por instrucción
DCILAB	Laboratorio de diseño de circuitos integrados
FPGA	Del inglés Field Programmable Gate Array
FPU	Unidad de punto flotante
HDL	Lenguajes de descripción de hardware
HMM	Modelos ocultos de Markov
ISA	Arquitectura del conjunto de instrucciones
PC	Contador de programa
RAM	Memoria de acceso aleatoria
RIS	Redes Inalámbricas de Sensores
RISC-V	Arquitectura del conjunto de instrucciones utilizado en la tesis
RISC	Conjunto de instrucciones reducido
ROM	Memoria de solo lectura
RTL	Del inglés Register Tranfer Level
SiRPA	Sistema de Reconocimiento de Patrones Acústicos
SO	Sistema operativo
SoC	Sistema en chip
SPI	Interfaz periférica serial
VHDL	Del inglés VHSIC Hardware description language

Notación general

$\mathbf{A}$	Matriz. $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$
$\underline{\mathbf{x}}$	Vector.

$$\underline{\mathbf{x}} = [x_1 \; x_2 \; \ldots \; x_n]^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Capítulo 1

Introducción

1.1 Entorno de la tesis

Actualmente Costa Rica es uno de los países reconocidos en el mundo como líderes en la conservación del medio ambiente. Todo esto ha sido posible gracias a su sistema de áreas protegidas. Estas se consolidaron con la creación de los parques nacionales permitiendo al día de hoy que cerca de un 25% del territorio se encuentre dentro de alguna categoría de protección ambiental [4].

Por otra parte, los recursos naturales constituyen un medio indispensable para el crecimiento integral de los seres humanos y para el crecimiento del país. La importancia de los mismos radica en que contribuyen de manera indispensable al desarrollo mediante alimentos, materias primas, servicios, entre otros [5].

De igual forma, no se puede dejar de lado el aporte que realiza a la economía de los países mediante el turismo. Los ingresos económicos se incrementaron en un 8,3% en el año 2014 en relación con el 2013, esto debido a que se aumentó de $2.433 a un total de $2.636 millones, datos tomados según el informe del Instituto Costarricense de Turismo [6].

Sin embargo, a pesar de tener áreas protegidas y leyes que castigan a personas que practican la caza y tala indiscriminada e ilegal, estas no son suficientes, ya que existen individuos que se aprovechan del hecho de que al ser ecosistemas grandes y al no tener un presupuesto tan alto como se desearía para poder proteger estas zonas forestales, es imposible lograr una protección completa. A manera de ejemplo, solo en el 2014 se tenía un faltante aproximado de 300 funcionarios dedicados al control y protección de áreas de protección, según fuentes del Sistema Nacional de Áreas de Conservación [4].

Debido a esto, se han realizado diversos intentos para contraatacar esta problemática y lograr así mejorar la protección sobre estas zonas. Para ello primeramente se definieron los requerimientos del sistema y se determinó que debido a la escasez de personas a cargo de la protección, a la dificultad en el acceso de las zonas boscosas y principalmente la inexistencia de fuentes de alimentación continuas generan la necesidad de desarrollar una

solución cuyo consumo de potencia sea reducido. De esta forma, se implementará un sistema que opere correctamente durante una gran cantidad de tiempo utilizando para este fin únicamente un dispositivo de alimentación autónoma como lo es una batería.

Es en este contexto, donde surgió la idea de desarrollar una RIS (Red inalámbrica de sensores) para monitorizar las zonas protegidas. Una RIS consiste en dispositivos distribuidos espaciados autónomos utilizando sensores para monitorear condiciones físicas o ambientales. A cada miembro de la red se le conoce además como nodo [7]. Esta solución se consideró por el hecho de que las RIS son capaces de medir parámetros, almacenarlos, procesarlos y enviarlos entre sí, lo cual representa una característica deseable en el proyecto y además, constituye una solución que presenta bajo consumo de potencia. Por esta razón, la Escuela de Ingeniería Electrónica inició el desarrollo de un sistema basado en RIS para solventar este problema.

Desde una perspectiva de funcionamiento se tiene que la RIS obtiene muestras de señales acústicas del medio, en este caso el bosque. Luego, los datos son procesados aplicando reconocimiento de patrones y en caso de encontrar un estado de alarma: disparo o motosierra, se activaría el bloque de transmisión de información donde posteriormente la persona a cargo de la protección de la zona, recibirá una notificación donde se le indicará el tipo de evento que se generó y cuál nodo de la red fue el que disparó esa alarma. En la figura 1.1 se muestra un diagrama de bloques de un nodo de la RIS que permite de forma gráfica entender el funcionamiento de la misma.

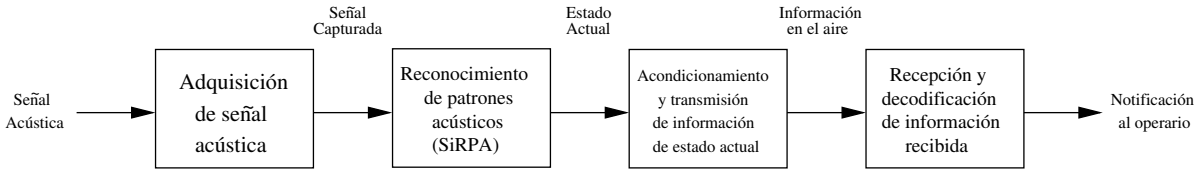

Figura 1.1: Procesamiento realizado por cada nodo miembro de la RIS [1].

Mirando hacia el interior de la unidad de procesamiento, específicamente el bloque denominado SiRPA (Sistema de reconocimiento de patrones acústicos), este como todo sistema típico de reconocimiento de patrones acústicos, está formado de un bloque de preprocesado, una etapa de identificación o extracción de características fundamentales y un bloque de clasificación.

En el primer bloque, se normaliza y digitaliza la señal acústica. El bloque de identificación está compuesto por un banco de filtros, un estimador de energía, un reductor de dimensiones y un árbol clasificador. En el banco de filtros el espectro de la señal acústica se separa en ocho bandas distintas y luego se le estima la energía por banda a cada una de ellas.

El reductor de dimensiones proyecta el espacio de ocho dimensiones a la salida del estimador de energía en un espacio de tres dimensiones, por medio de una transformación lineal. Este se lleva a cabo ya que según lo predice la *maldición de la dimensionalidad*, el conjunto de símbolos necesarios para describir de manera adecuada las observaciones rea-

lizadas es tres órdenes de magnitud mayor al trabajar en un espacio de ocho dimensiones en comparación a uno de tres. Esto también, repercute en los requisitos de memoria del sistema, ya que aumentan y se estima que se necesitaría alrededor de tres veces más de tiempo de procesamiento para realizar búsquedas en el árbol binario en la etapa posterior si se realizara sin una disminución de la dimensionalidad [1, 8].

El árbol binario o generador de símbolos, se utiliza para encontrar el centroide más cercano a un patrón de entrada de tres dimensiones, para ello se utiliza la distancia L1 o Manhattan. De esta manera, los símbolos discretos se generan, asociando estos con las trayectorias continuas en el espacio del vector tridimensional de entrada que describe la señal acústica en un determinado instante. En total el árbol binario genera símbolos que tiene como rango un alfabeto de 32 símbolos discretos [9, 10, 11, 12].

Finalmente, la etapa de clasificación es la encargada de calcular las probabilidades de que un conjunto de símbolos de entrada sean parte de una señal acústica de tipo bosque, motosierra o disparo. Para esta etapa, se utiliza un clasificador basado en la técnica de modelos ocultos de Markov (HMM) y es exactamente sobre esta unidad donde se centra el desarrollo de la tesis. En la figura 1.2 se muestra mediante un diagrama de bloques el funcionamiento interno del SiRPA.

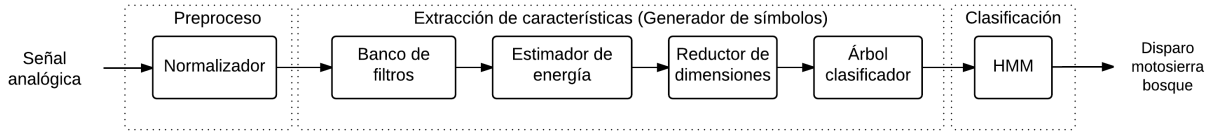

Figura 1.2: Descripción mediante bloques del Sistema de reconocimiento de patrones acústicos (SiRPA).

1.2 Descripción del problema y justificación

La idea principal del desarrollo de esta tesis nace en el laboratorio de Diseño de Circuitos Integrados (DCILAB) del Instituto Tecnológico de Costa Rica (ITCR), dentro del marco del proyecto de investigación SiRPA. El mismo surge como parte de una investigación donde se necesitaba desarrollar una unidad de clasificación en un lenguaje de descripción de hardware para un sistema de reconocimiento acústico de disparos y motosierras. El clasificador tomará decisiones del estado actual del sistema utilizando el algoritmo de modelos ocultos de Markov.

La unidad implementada debe respetar los siguiente requerimientos: tener un consumo de potencia bajo, la cantidad de hardware mínima y realizar la ejecución del algoritmo dentro de ciertos límites temporales de forma que no exista traslape de muestras a la entrada del módulo clasificador. Estos requerimientos se cuantifican en la sección 1.7.

Anteriormente, se desarrolló una unidad de clasificación por otros dos investigadores que realizaron sus trabajos de graduación en el laboratorio. La solución propuesta fue el

desarrollo de una máquina microprogramada. Sin embargo, esta presentó un problema de representación numérica. Este se debió a limitaciones en la escalabilidad que presenta el algoritmo de modelos ocultos de Markov, por lo que no fue posible obtener una detección de patrones correcta. Agregado a esto, la solución era válida únicamente para esa aplicación (arquitectura alambrada en hardware), por lo que era imposible migrar esa solución a futuros proyectos de investigación.

Diversos trabajos como los realizados en [13, 14, 15, 16] han implementado el algoritmo en estudio mediante aceleradores de hardware y han obtenido soluciones realmente eficientes en cuanto a tiempos de ejecución. Sin embargo, el aporte de este trabajo no está en implementar el algoritmo con tiempos de ejecución alta pues la velocidad de ejecución no es un factor determinante para el proyecto SiRPA. El principal aporte que surge de esta investigación, es la implementación del algoritmo de modelos ocultos de Markov dentro de un microprocesador de tamaño mínimo, que al sumar los objetivos de flexibilidad y bajo consumo, hace que la propuesta de un ASP (del inglés Application Specific Processor) resultara atractiva. Así, mediante un enfoque que balancea las cargas de trabajo del hardware y el software, se obtiene una solución eficiente en cuanto al área y el consumo de potencia del mismo.

1.3 Síntesis del problema

¿Cómo implementar un microprocesador de aplicación específica en un lenguaje de descripción de hardware eficiente en términos de potencia y recursos, que ejecute un algoritmo de reconocimiento de patrones basado en modelos ocultos de Markov?

1.4 Enfoque de la solución

Para la implementación de la unidad de cálculo del algoritmo de reconocimiento de patrones acústicos basado en la técnica de modelos ocultos de Markov, se implementará un microprocesador segmentado multiciclo de aplicación específica basado en el conjunto de instrucciones RISC-V de la Universidad de California, Berkeley [17].

Este se implementará en el lenguaje de descripción de hardware Verilog y contendrá estrictamente el hardware necesario para ejecutar el algoritmo, es decir, de todo el conjunto de instrucciones RISC-V, solo se desarrollarán las instrucciones que son necesarias para la aplicación. De esta forma, se obtendrá un microprocesador donde el consumo de potencia y el área sea muy reducida.

Posteriormente, se desarrollará una aplicación en el lenguaje de programación C y utilizando el GNU Toolchain [18] que provee la herramienta RISC-V, se compilará la aplicación para el hardware implementado. Finalmente, se integrará el software con el hardware logrando obtener un sistema que cumple con los requerimientos planteados por el proyecto

SiRPA.

1.5 Trabajos anteriores

El SiRPA es un proyecto de investigación donde se han explorado y analizado diferentes enfoques de solución por distintos investigadores que han ido dando forma a cada una de las etapas que compone el sistema. Tanto a nivel de software como a nivel de hardware, los aportes incluyen desde propuestas de diseño, implementación y mejoras o rediseño de etapas que han presentado problemas. Entre estos trabajos se encuentran los siguiente:

- En [19], se realizó la primera implementación del SiRPA utilizando MATLAB y tenía como objetivo el análisis de las señales acústicas y la etapa de identificación de características utilizando la teoría de onditas (wavelets en inglés). La misma constituyó el primer modelo de referencias del sistema. Cabe mencionar que la estrategia de extracción de características mediante onditas se descartó posteriormente.
- En [20], se implementó el SiRPA en una FPGA (acrónimo inglés de Field Programmable Gate Array) utilizando el lenguaje de descripción de hardware VHDL de donde se rescató la estrategia de implementación del banco de filtros utilizando una misma unidad de filtro para cada una de las ocho bandas que conformaba el banco. Además, se implementó la etapa de clasificación utilizando el algoritmo de HMM donde mediante un MAP (del inglés Matrix and Arrays Processor), el cual es una estructura digital en donde la arquitectura se optimiza para la ejecución combinacional de operaciones en punto flotante de 32 bits.
- En [21], se realizó un módulo de reductor de dimensiones utilizando la técnica de transformación lineal basado en el algoritmo de discriminantes de Fisher. Con esto se logró pasar de un espacio de 8D a un espacio de 3D, lo que facilitó la clasificación de señales acústicas y a su vez permitió una reducción energética.
- En [11], se desarrolló una aplicación con interfaz gráfica llamada HMMSoft, que realizaba el entrenamiento del algoritmo de HMM utilizado por el clasificador. El mismo cuenta con diferentes técnicas de entrenamiento: simple, múltiple y negativo. Además, esta aplicación permitió la identificación de características de patrones acústicos para la verificación del reconocimiento de audio. Esta implementación constituyó un desarrollo importante ya que no solo permitió el entrenamiento del sistema sino que también representó un modelo de referencia del sistema para la verificación. Todo esto implementado en una aplicación con interfaz gráfica desarrollada en C/C++.
- En [1], se desarrolló la sección de extracción de los patrones acústicos y entrenamiento a partir de conjuntos de datos de audio similares a los que se esperaría detectar. Lo anterior se integró dentro de la aplicación HMMSoft que se desarrolló en [11]. Además, se implementó el SiRPA dentro de un sistema empotrado(BeagleBoard xM [22]) que ejecuta un sistema operativo GNU/Linux, usando el lenguaje de programación C. Esta aplicación, viene a constituir el principal modelo de referencia utilizado

en el desarrollo de este trabajo ya que brindó un punto de comparación para la implementación y desarrollo en la FPGA y en el ASIC (del inglés Application-Specific Integrated Circuit).

- En [23], se desarrollaron los módulos de hardware del reductor de dimensiones, generador de símbolos y una unidad de HMM utilizando el lenguaje de descripción de hardware Verilog. A diferencia del diseño de la etapa de HMM planteada en [20], esta vez se desarrolló el HMM utilizando una máquina microprogramada basada en la versión en C planteada en [1] para la tarjeta empotrada. Sin embargo, al comparar esta aproximación del HMM con el modelo de referencia elaborado en [1] se observó que los resultados divergen de lo esperado y por tanto son erróneos. Lo cuál, imposibilita por tanto clasificar los patrones acústicos correctamente. Explorando el hardware se observó que los cálculos dentro de la unidad tienden a cero rápidamente. Esto se debe a que, no se consideraron los problemas de escalamiento que se describen ampliamente en [24].
- En [25], se realizó la verificación de los módulos descritos en un lenguaje de descripción de hardware que conforman el SiRPA.
- En [26], se tomó como base el banco de filtros realizado en [20] y se rediseñó la unidad de manera que se consideraran correctamente los efecto del submuestreo. Se realizó un estudio del bloque de reducción de dimensiones elaborado en [23] y se determinó la necesidad de aumentar el formato de palabra de 16 a 24 bits. Se integraron además todos los bloques para formar la etapa de identificación o extracción de características. Finalmente, se realizó la verificación de esta etapa, tomando como referencia dorada el trabajo realizado en [1].

1.6 Estado de arte actual

En esta sección se presenta de manera resumida el estado de arte actual relacionado a unidades de modelos ocultos de Markov desarrolladas en hardware. El objetivo de la revisión bibliográfica fue conocer qué se ha realizado y cómo distintos autores han tratado de resolver el problema de reconocimiento de patrones acústicos. Además la revisión bibliográfica tiene como objetivo buscar información para obtener resultados adecuados de acuerdo con los requerimientos en la implementación del microprocesador. A continuación se citan algunos de los principales trabajos realizados por distintos autores, fundamentalmente en el reconocimiento acústico de la voz humana, utilizando la técnica de modelos ocultos de Markov:

- En [27], mediante una solución enfocada a un codiseño hardware-software, se generó un modelo de solución en software y finalmente se realizó un mapeo del software al lenguaje de descripción de hardware VHDL. El principal aporte aquí, fue la modificación que realizaron al algoritmo de modelos ocultos de Markov para eliminar la necesidad de realizar un escalamiento de las constantes. Con esta observación los autores lograron reducir la mitad del área del circuito digital logrando obtener una

tasa de reconocimiento de más del 95% con tiempos de ejecución de aproximadamente $5\mu s$.

- En [28], la solución integra hardware y software. El software se ejecuta sobre el procesador Microblaze de Xilinx [29] y un acelerador de hardware se encarga de ejecutar el algoritmo.
- En [30], se resolvió el problema utilizando un SoC (Acrónimo inglés de Sistema en Chip) específicamente: el ARM7TDMI [31] y un coprocesador. Esta solución se enfocó en mejorar el rendimiento cuando se utiliza un sistema empotrado. El sistema final operaba a una frecuencia de $24MHz$ con un consumo de potencia de $0.56mW/MHz$.
- En [32], el problema fue solucionado utilizando un SoPC (acrónimo inglés de Sistema en chip programable), para ello utilizaron el procesador Nios II de Altera [33] y con un programa pequeño lograron reducir el tamaño de la memoria de programa. Específicamente se obtuvo un tamaño del código de aproximadamente 312KBytes. Se utilizó el algoritmo de GMM (acrónimo inglés de modelo mixto Gaussiano) para modelar el vector de información de la voz. El resultado de esta etapa, pasa finalmente por el HMM donde se tomaban las decisiones sobre reconocimiento de palabras.
- En [34] las probabilidades de observación son calculadas utilizando una función de distribución Gaussiana. Se expuso además el cálculo del logaritmo en base dos de esa probabilidad en hardware.

1.7 Requerimientos de la investigación

Para el desarrollo de la tesis se establecieron los siguientes requerimientos que debía cumplir la solución seleccionada.

- Consumo de potencia dinámica menor a $127.3mW$ a $100MHz$.
- El consumo de recursos en FPGA de la unidad debe ser de 3491 slices, 1544 flip flops y 6601 LUT.
- El tiempo de ejecución del algoritmo deberá ser menor a $58.1395ms$.

Los dos primeros requerimientos fueron tomados del trabajo realizado en [23]. Estos constituyen los datos obtenidos de la última iteración realizada en la unidad de clasificación y la idea de esta tesis, es lograr una solución que reduzca estos.

En relación con el último requerimiento, en el trabajo [1], se estudiaron distintos modelos de HMM y los mejores resultados de clasificación, se presentaron cuando el modelo tenía una longitud de cadena igual a 20, diez estados ocultos y un muestreo igual a uno. De esta forma, y sabiendo que como resultado del submuestreo realizado por el banco de filtros en la etapa de identificación del SiRPA (Ver [26]) , cada símbolo se generaba en un tiempo igual a $2.9070ms$. Así fácilmente, se llega a un tiempo igual a $58.1395ms$ para un total de 20 símbolos.

1.8 Objetivos

1.8.1 Objetivo General

- Desarrollar un microprocesador de aplicación específica en hardware que ejecute el algoritmo de modelos ocultos de Markov para la etapa de clasificación de una unidad de reconocimiento de patrones acústicos.

1.8.2 Objetivos específicos

- Comparar diferentes arquitecturas de conjuntos de instrucciones y seleccionar la que mejor se adapte para el desarrollo del microprocesador.
- Implementar el conjunto de instrucciones seleccionado en una arquitectura que disminuya el consumo de potencia y área del circuito necesario.
- Desarrollar en un lenguaje de alto nivel un clasificador de patrones acústico basado en la técnica de modelos ocultos de Markov que ejecute sobre el microprocesador desarrollado.
- Integrar la unidad de clasificación con la etapa de identificación del SiRPA y validar el correcto funcionamiento del sistema.

1.9 Estructura de la tesis

El capítulo 2 muestran la información necesaria que permite comprender la solución del problema. Incluye todos los fundamentos teóricos básicos que en conjunto contribuyeron al funcionamiento de la solución desarrolla.

El capítulo 3 es la sección más importante del documento y es donde se muestra de manera detallada la solución. Este capítulo, se divide en cuatro secciones principales: elección del conjunto de instrucciones para la arquitectura, implementación del microprocesador para ese conjunto de instrucciones, elaboración del algoritmo de modelos ocultos de Markov en un lenguaje de alto nivel y finalmente la integración del hardware software.

En el capítulo 4 se presentan los resultados obtenidos y simulaciones del circuito que validan el funcionamiento correcto del sistema digital implementado en hardware. Se analizan estos resultados para comprobar que efectivamente el sistema cumple con los requisitos planteados en el problema en estudio. Finalmente en el capítulo 5 se presentan las conclusiones y recomendaciones obtenidas del capítulo 4.

Capítulo 2

Marco teórico

En este capítulo se encuentran las principales bases teóricas que permitieron llevar a cabo la tesis. Además se presenta la propuesta metodológica utilizada en este trabajo.

2.1 Reconocimiento de patrones

Primeramente, cuando se habla de un patrón, se refiere a cualquier entidad de interés que se desea reconocer o identificar. Algunos ejemplos de patrones incluyen: un píxel en una imagen, un gesto facial, un rostro humano, la voz de un individuo o la forma de un animal [35].

El reconocimiento de patrones por tanto es la disciplina científica de aprendizaje computacional o inteligencia artificial que tiene como objetivo la clasificación de datos (patrones) en una serie de categorías o clases [35].

Un sistema de reconocimiento de patrones es un sistema automático que tiene como objetivo clasificar el patrón de entrada en una clase específica. Está compuesto típicamente de una etapa de preprocesamiento donde se acondiciona y se digitaliza la señal. Luego de la digitalización, se procede con la extracción de las características del modelo en estudio, este proceso es llamado etapa de identificación. La clasificación por su parte, permite reconocer un objeto (o un patrón) mediante el uso de algunas características de la etapa de extracción [35].

El esquema de clasificación se basa generalmente en la disponibilidad del conjunto de entrenamiento, que es un grupo de patrones que ya ha sido clasificado. Esta estrategia de aprendizaje se conoce como aprendizaje supervisado en oposición al aprendizaje no supervisado. Una estrategia de aprendizaje se dice que es sin supervisión, cuando en el sistema no posee información a priori sobre las clases; esta establece las propias clases basadas en las regularidades de las características, las cuales son aquellas mediciones que se extraen de un patrón para que lo represente en el espacio. En otras palabras, el análisis del patrón nos permite utilizar algunas características para describir y representar la señal

en lugar de utilizar el propio patrón [35].

En la figura 2.1 se muestra el modelo de un sistema de reconocimiento de patrones acústicos genérico donde se encuentran claramente identificadas las etapas de adquisición de la señal, preprocesado, identificación o análisis, el entrenamiento y la etapa final de clasificación.

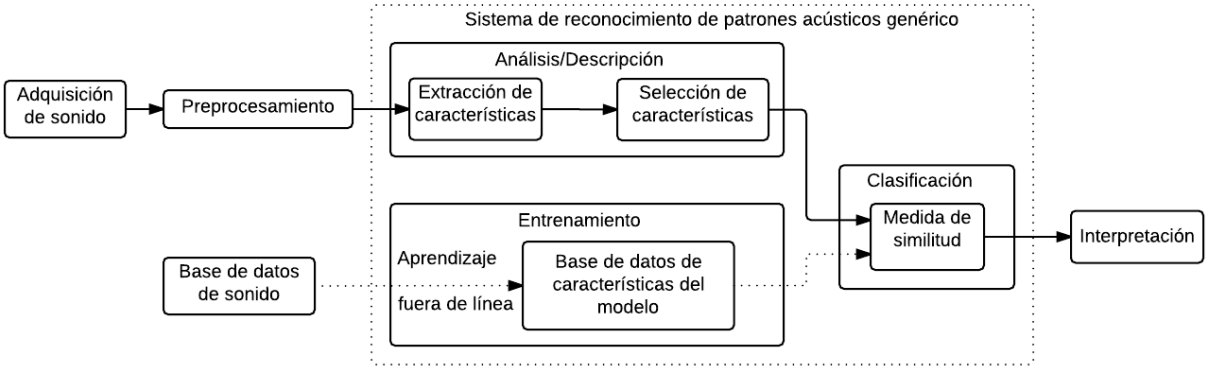

Figura 2.1: Modelo de un sistema de reconocimiento de patrones genérico.

2.2 Modelos Ocultos de Markov (HMM)

Actualmente existen distintos tipos de clasificadores, en esta sección se estudiará el clasificador utilizado en el SiRPA denominado HMM. Este es una herramienta para representar distribuciones de probabilidad sobre secuencias de observación. Esta herramienta se utiliza en casi todos los sistemas actuales de reconocimiento de voz, en numerosas aplicaciones computacionales de biología molecular, en algoritmos de compresión de datos y en áreas de reconocimiento facial y reconocimiento de patrones [36].

2.2.1 Elementos fundamentales de un HMM

Los HMM se caracterizan por una serie de elementos fundamentales que se describen a continuación [24]:

- N, es el número de estados del modelo. Aunque los estados están ocultos, para muchas aplicaciones prácticas a menudo hay algún significado físico unido a los estados o grupos de estados del modelo. En general, los estados están interconectados de tal manera que cualquier estado puede llegar desde cualquier otro estado.
- M, es el número de símbolos distintos de observación por estado, es decir, el tamaño del alfabeto discreto. Estos denotan la salida del sistema que se desea modelar. Se denota los símbolos individuales como $\underline{\boldsymbol{V}} = \{v_1, v_2, ..., v_M\}$.
- La distribución de probabilidad de transición de estados $\mathbf{A} = \{a_{ij}\}$ donde cada uno de los coeficientes a_{ij} define la probabilidad de pasar del estado i al estado j. La matriz $\mathbf{A}$ es una matriz cuadrada de tamaño N. Para el caso especial donde

cualquier estado puede alcanzar cualquier otro estado en un paso simple, se tiene $a_{ij} > 0$ para todo i, j. Para algunas clases de HMMs, se tiene que $a_{ij} = 0$ para uno o más i, j pares.

- La matriz $\mathbf{B}$ representa la distribución de probabilidad de observación de un símbolo. Se define como $\mathbf{B} = \{b_j(k)\}$, donde cada coeficiente que denota la probabilidad de observación del símbolo k en el estado i es una matriz que tiene dimensiones de $N \times M$.
- La distribución del estado inicial $\underline{\boldsymbol{\pi}} = \{\pi_i\}$ es de tamaño $1 \times N$ e indica la probabilidad de que al inicializarse el modelo se esté en el estado i.

Dada la cadena de observación $\underline{\boldsymbol{O}} = \{O_1 O_2 ... O_T\}$ a la entrada del modelo. El algoritmo de HMM puede representarse en su totalidad por las matrices $\mathbf{A}$, $\mathbf{B}$ y $\underline{\boldsymbol{\pi}}$ mediante el modelo [24]:

$$\lambda = (\mathbf{A}, \mathbf{B}, \underline{\boldsymbol{\pi}}) \tag{2.1}$$

2.2.2 Principales retos de HMM

Hay tres problemas básicos que deben ser resueltos para que el modelo de la ecuación (2.1) sea utilizado en aplicaciones reales. Estos problemas son los siguientes [24]:

1. Dada la secuencia de entrada $\underline{\boldsymbol{O}} = \{O_1 O_2 ... O_T\}$ y el modelo $\lambda = (\mathbf{A}, \mathbf{B}, \underline{\boldsymbol{\pi}})$ ¿cómo calcular eficientemente la probabilidad de observación de una secuencia de observación $P(O|\lambda)$ a partir del modelo?

2. Dada la secuencia de entrada $\underline{\boldsymbol{O}} = \{O_1 O_2 ... O_T\}$ y el modelo λ, ¿cómo escoger la secuencia de estado $Q = q_1 q_2 ... q_T$ que es óptima en el sentido estricto?

3. ¿Cómo ajustar el parámetro del modelo $\lambda = (\mathbf{A}, \mathbf{B}, \underline{\boldsymbol{\pi}})$ para maximizar $P(O|\lambda)$?

Para el caso del SiRPA los problemas que toman relevancia son el primero y el tercero. El problema tres se relaciona con las estrategias de entrenamiento y ya fue resuelto en [11], por lo que nos vamos a concentrar únicamente en el problema uno, el cuál consiste en calcular la probabilidad de observación de una secuencia $\underline{\boldsymbol{O}} = \{O_1 O_2 ... O_T\}$, dado el modelo λ [24]. Para ello se plantean dos soluciones que se resumen en breve.

Método directo

El camino directo es a través de enumerar cada secuencia de estado posible de longitud T (número de observaciones) [24]. Para ello se considera una secuencia de estado fijo:

$$\underline{\boldsymbol{Q}} = \{q_1 q_2 ... q_T\} \tag{2.2}$$

donde q_1 es el estado inicial. La probabilidad de observación de la secuencia $\underline{\boldsymbol{O}}$ para la secuencia de estado de la ecuación (2.2), suponiendo independencia estadística de observaciones es:

$$\begin{aligned} P(O|Q,\lambda) &= \prod_{t=1}^{T} P(O_t|q_t,\lambda) \\ &= b_{q_1}(O_1)\cdot b_{q_2}(O_2)\cdots b_{q_T}(O_T) \end{aligned} \tag{2.3}$$

La probabilidad de una secuencia de estado $\underline{\boldsymbol{Q}}$ puede escribirse como:

$$P(\underline{\boldsymbol{Q}}|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \cdots a_{q_{T-1} q_T} \tag{2.4}$$

La probabilidad conjunta de $\underline{\boldsymbol{O}}$ y $\underline{\boldsymbol{Q}}$, que es la probabilidad de que $\underline{\boldsymbol{O}}$ y $\underline{\boldsymbol{Q}}$ ocurran simultáneamente, es simplemente el producto de los dos términos anteriores, es decir:

$$P(O,Q|\lambda) = P(O|Q,\lambda)P(Q,\lambda) \tag{2.5}$$

La probabilidad de $\underline{\boldsymbol{O}}$ se obtiene de sumar la probabilidad conjunta de todas las secuencias de estado q, la cual está dada por:

$$P(O,\lambda) = \sum_{allQ} P(O|Q,\lambda)P(Q,\lambda) \tag{2.6}$$

Sin embargo, para calcular la $P(O|\lambda)$ con la ecuación (2.6) se requiere aproximadamente $2T \cdot N^T$ y este cálculo es computacionalmente inviable [24]; por ejemplo, para un $N = 5$ y un $T = 100$ se requieren aproximadamente 10^{72} cálculos.

Por esta razón, realizar el cálculo de forma directa no es posible. Por fortuna existe un algoritmo más eficiente que resuelve el problema y se conoce como *Algoritmo de avance* [24].

Algoritmo de avance

Considere la variable de avance $\alpha_t(i)$ definida como:

$$\alpha_t(i) = P(O_1 O_2 \cdots O_t, q_t = S_i|\lambda) \tag{2.7}$$

La probabilidad de observación parcial de la secuencia $O_1 O_2 \cdots O_t$ hasta el tiempo t y el estado S_i en el tiempo t, dado el modelo λ, se resuelve por inducción; para ello, primeramente se inicializan las probabilidades de avance como la probabilidad conjunta del estado S_i e inicializa la observación O_1 [24], mediante:

$$\alpha_1(i) = \pi_i b_i(O_i), 1 \leq i \leq N \tag{2.8}$$

Seguidamente, se alcanza S_j en el tiempo $t+1$ de los N posibles estados, S_i, $1 \leq i \leq N$, en el tiempo t. Desde que $\alpha_t(i)$ es la probabilidad conjunta de que el evento $O_1 O_2 \cdots O_t$ sea observado, y el estado en el tiempo t es S_i. El producto $\alpha_t(i)a_{ij}$ es entonces la probabilidad conjunta de que la secuencia $O_1 O_2 \cdots O_t$ sea observada, y el estado S_j se alcance en el tiempo $t+1$ a través del estado S_i en el tiempo t. Sumando estos productos sobre todas los N posibles estados S_i, $1 \leq i \leq N$ en el tiempo t, se obtiene la probabilidad de S_j en el tiempo $t+1$ con todas las observaciones parciales. Una vez que esto finaliza y S_j esté lista, es fácil ver que $\alpha_{t+1}(j)$ se obtiene al contabilizar las observaciones O_{t+1} en el estado j, es decir, multiplicando la cantidad sumada por todos los estados j. El cálculo de:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(O_{t+1}), \; 1 \leq t \leq T-1 \qquad 1 \leq j \leq N \tag{2.9}$$

El paso final da el resultado deseado de $P(O|\lambda)$ como la suma de las variables de avance $\alpha_T(i)$:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.10}$$

Si se examina el *algoritmo de avance* se puede deducir que se requieren N^2T cálculos, en lugar de $2TN^T$ del método directo [24]. Por ejemplo para $N = 5$, $T = 100$ se tiene aproximadamente 3000 cálculos contra 10^{72} cálculos del método directo.

2.2.3 Problema de implementación del HMM

Anteriormente se mencionó que la probabilidad de observación de la secuencia $\underline{\boldsymbol{O}}$ para la secuencia de estado de la ecuación (2.2) esta dado por la ecuación (2.10). Sin embargo, este procedimiento presenta un problema numérico en su implementación. Esto sucede debido a que conforme crece el número de observaciones T, el número de estados del modelo N o el tamaño del alfabeto discreto M, las muestras de las probabilidades de salida tienden a cero de forma prácticamente exponencial. Recordemos que las probabilidades son menores que uno, y de hecho muchas veces son significativamente menores que uno por lo que, al realizar los productos el resultado tiende a cero rápidamente. Para resolver esto de forma eficiente se debe realizar un procedimiento de escalado que se encuentra ampliamente documentado en [24].

Para realizar el escalado, se determina el factor de escala c_t como:

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{2.11}$$

Con este cambio de escala, simplemente se modifican las ecuaciones (2.8) y (2.9) mediante un cambio de variable α por α', donde se obtiene que:

$$\alpha'_t(i) = c_t \alpha_t(i),\ 1 \leq i \leq N \tag{2.12}$$

De esta forma, para calcular la probabilidad de observación utilizando el *algoritmo de avance*, se utilizan las ecuaciones (2.8) y (2.9). Posteriormente, se realiza un escalamiento mediante la ecuación (2.12) y finalmente, para compensar ese escalamiento, ya no es posible utilizar la ecuación (2.10) y en su lugar se utiliza el valor de su logaritmo mediante los coeficientes c_t tal que:

$$log(P(O|\lambda)) = -\sum_{t=1}^{T} log(c_t) \tag{2.13}$$

De esta manera, los resultados son mapeados de un rango que va desde 0 a 1 hacia un nuevo rango que va desde $-\infty$ a 0 [24].

2.3 Matrices de confusión

Parte importante de la validación de un clasificador de patrones de múltiples clases es la utilización de la herramienta llamada matrices de confusión. Estas aportan información sobre las tasas de reconocimiento de los HMM y se utilizan con el objetivo de conocer el efecto de diferentes variaciones en los parámetros del modelo sobre la tasa de reconocimiento. Variaciones en la longitud de la cadena de observación, número de estados ocultos, efectos del submuestreo, son algunos de los factores que influyen sobre la tasa de reconocimiento y su efecto se puede estudiar con esta técnica [37].

La matriz de confusión $\mathbf{C}$, es de tamaño $N \times N$, donde N es el número de clases usadas en la clasificación y para el caso particular del SiRPA $N = 3$. Las filas indican a cuál de estas clases fueron asignados por el clasificador. Las columnas por su parte indican la clase real a la cual pertenecen los objetos. Así, el elemento c_{ij} de la matriz de confusión indica un objeto que fue asignado a la clase i, pero pertenece realmente a la clase j [37].

Esta matriz puede resumir aun más los resultados de la clasificación mediante una matriz de confusión en bruto. A partir de esta, se pueden obtener dos valores que aportan más información sobre los resultados, que son la sensitividad y VPP (Valor predictivo positivo) [37].

En este trabajo, la sensitividad indica el grado de efectividad del clasificador para las cadenas de observación, es decir cuál es la probabilidad de que un objeto que pertenece

a una clase específica sea clasificado correctamente. Mientras que el VPP indica qué tan efectivo es el modelo para reconocer las cadenas de su propia clase, es decir, indica cuál es la probabilidad de que un objeto asignado a una clase permanezca verdaderamente a esa clase. El valor deseable de estos indicadores es cuando ambos son altos, cercanos al 100% [37].

2.4 Procesador de aplicación específica ASP

Los procesadores de propósito general están diseñados para ejecutar múltiples aplicaciones y tareas. Sin embargo, estos pueden ser bastante caros, especialmente para pequeños dispositivos que están diseñados para realizar tareas específicas. También, los procesadores de propósito general pueden carecer de alto rendimiento necesario muchas veces para tareas particulares [38].

Por lo tanto, los procesadores de aplicación específica surgieron como una solución de alto rendimiento y siendo además rentables. Estos se han convertido en parte de cotidiana de la vida y se pueden encontrar en casi todos los dispositivos que se utilizan hoy en día. Dispositivos tales como: refrigeradores, consolas de video juegos y dispositivos móviles, todos ellos tienen un procesador de aplicación específica particular y adecuado a la aplicación para la cual fueron diseñados. Usualmente, estos presentan las siguientes características: bajo costo, consumo de energía moderado y alto rendimiento [38].

2.5 Arquitectura segmentada multiciclo de un microprocesador

Luego de estudiar los conceptos de reconocimiento de patrones acústicos, resulta importante el estudio de la arquitectura interna de un microprocesador. En este trabajo, se utilizó la arquitectura segmentada multiciclo para la implementación del microprocesador por lo que es importante realizar un estudio de la misma.

Es una metodología de implementación de arquitectura donde una unidad funcional puede ser utilizado más de una vez por instrucción, siempre y cuando se utilice en un diferente ciclo de reloj. Este enfoque puede ayudar a reutilizar una gran cantidad de hardware y es más conservadora en términos de consumo, al disminuirse el paralelismo [2].

Las capacidades para permitir que las instrucciones tomen diferentes números de ciclos de reloj y de compartir unidades funcionales dentro de la ejecución de una sola instrucción, son las principales ventajas de un diseño basado en una arquitectura multiciclo [2].

Algunas de las principales características de esta arquitectura son [2]:

- Una unidad de memoria simple que se utiliza tanto para datos como para instrucciones.

- Todas las operaciones se realizan dentro de una Unidad Aritmético Lógica (ALU).
- Se requiere de registros intermedios entre unidades funcionales para almacenar la salida hasta que ocurra el siguiente flanco de reloj, donde el valor es utilizado por la siguiente unidad funcional.

2.6 Proceso de compilación

El proceso de compilación de un programa típico desarrollado en C esta formado de cuatro pasos fundamentales que se describen [39]:

- La primera etapa del proceso de compilación es el uso del preprocesador para expandir macros y ficheros de cabecera incluidos.
- La siguiente etapa del proceso es la compilación real de código fuente preprocesado a lenguaje ensamblador, para un procesador específico.
- El propósito del ensamblador es convertir el lenguaje ensamblador en código máquina y generar un archivo de objeto. Cuando hay llamadas a funciones externas en el archivo de origen de montaje, este deja las direcciones de las funciones externas indefinidas, a rellenar más tarde por el enlazador.
- La etapa final de la compilación es la vinculación de archivos de objetos para crear un ejecutable.

Además, es importante el concepto de compilación cruzada, que consiste en compilar un código para crear un ejecutable que va dirigido hacia una plataforma distinta a aquella en la que el compilador se ejecuta [40]. Toma especial importancia en el proyecto ya que se realiza una compilación cruzada para generar el código fuente que va ser ejecutado sobre la arquitectura RISC-V.

El tipo de compilación cruzada que toma importancia en el desarrollo de la tesis es la *bare metal*, esto porque al ser un microprocesador de aplicación específica, no se cuenta con un SO (sistema operativo). Básicamente es un software que se ejecuta directamente sobre el hardware [40]. Para realizar este tipo de compilación, se debe escribir correctamente un archivo link.ld donde se inicialice todas las posiciones de memoria y realizar la compilación con el siguiente comando.

```
$riscv64-unknown-elf-gcc -nostdlib -nostartfiles -Tlink.ld -o Main Main.s
```

Un punto importante es que al compilar en *bare metal*, no se cuenta con soporte de bibliotecas que usualmente requieren de servicios del SO [40], por ejemplo si se desea utilizar una operación logaritmo, se debe crear una biblioteca propia ya que el compilador *gcc* utiliza el SO para ciertos servicios que posee esta función. Además, para la generación de constantes de tipo flotantes o dobles, *gcc* utiliza igualmente el SO para generar este tipo de constantes aritméticas.

2.7 Propuesta metodológica

La metodología seguida en este trabajo de investigación se basó en lo que se conoce como proceso de diseño en ingeniería, la cual se basa en una serie de pasos que en conjunto permitieron darle solución al problema planteado. A continuación se expone la estrategia metodológica seguida.

- Estudio de los trabajos anteriores para conocer cuál ha sido el avance de otros investigadores en el desarrollo del SiRPA.
- Investigación bibliográfica del estado de arte actual de los sistemas de reconocimiento de patrones acústicos basado en la técnica de modelos ocultos de Markov.
- Determinación del enfoque de solución para resolver el problema.
- Selección adecuada del ISA (acrónimo inglés de Arquitectura del conjunto de instrucciones).
- Selección adecuada de la arquitectura interna del microprocesador.
- Implementación del hardware del microprocesador en un lenguaje de descripción de hardware.
- Implementación del software de la aplicación.
- Integración del sistema y verificación exhaustiva del mismo.

Capítulo 3

Microprocesador de aplicación específica

En este capítulo se describió la solución implementada para el desarrollo de un microprocesador de aplicación específica que sea capaz de ejecutar el algoritmo de HMM. La solución presentó un enfoque de trabajo basado en un esquema arriba-bajo o Top-Down [41], partiéndose de un estudio del algoritmo para determinar el hardware que debía tener el microprocesador. Una vez establecidos los requerimientos de hardware, se implementó y verificó el mismo, usando el HDL Verilog. Finalmente, se elaboró la aplicación en el lenguaje de alto nivel C y se realizó la integración hardware/software para formar una solución de aplicación específica.

Adicionalmente, para el esquema de verificación, se utilizó una FPGA, ya que como todo problema de ingeniería, una correcta solución requiere de una plataforma existente para verificar el funcionamiento del diseño digital. De esta forma, una vez que se tuvo descrito el RTL del microprocesador y el software integrado dentro del mismo, se trasladó la aplicación a una plataforma de desarrollo Digilent Nexys 4 [42] la cual posee en su interior una FPGA Artix 7 [43] y se realizó la verificación del sistema.

En la figura 3.1 se observa como la integración del hardware y el software producen la aplicación final realizada en este trabajo.

Para solucionar el problema planteado en esta tesis y, por ende, encontrar solución al mismo, se realizaron una serie de pasos que se describen con detalle en las siguientes secciones del capítulo.

3.1 Selección de la arquitectura del conjunto de instrucciones (ISA)

La primera labor y probablemente el punto más importante en la descripción de la solución, radica en encontrar la arquitectura del conjunto de instrucciones que mejor se adapte

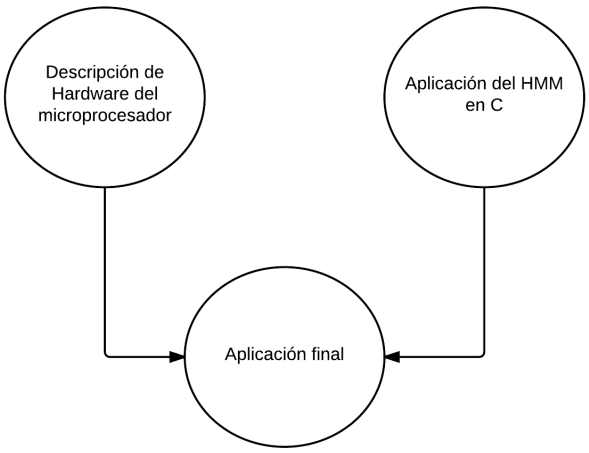

Figura 3.1: Integración hardware/software del esquema de trabajo del ASP.

Tabla 3.1: Comparación entre RISC y CISC [3].

RISC	CISC
Instrucciones simples	Intrucciones complejas
Carga pesada software	Carga pesada hardware
Compilador complejo	Compilador simple
Tamaño de instrucciones fijo	Tamaño de instrucciones variable
Pocas instrucciones	Muchas instrucciones
Requiere pocos ciclos de reloj para ejecutar instrucciones	Requiere muchos ciclos de reloj para ejecutar una instrucción
Decodificación simple en hardware	Decodificación compleja en hardware

a los requisitos del proyecto. Para esto, se analizaron primeramente los dos conjuntos de instrucciones principales: CISC (del inglés Complex Instruction Set Computing) y RISC (del inglés Reduced Instruction Set Computing), de donde se determinó de manera evidente, las ventajas que posee RISC sobre CISC para un sistema de bajo consumo de recursos como lo es un ASP. En la tabla 3.1 se muestra una comparación entre CISC y RISC.

Todos los aspectos expuestos anteriormente, permitieron tomar la decisión de utilizar un conjunto de instrucciones reducido. En este punto la interrogante que se planteó fue ¿cuál conjunto de instrucciones RISC utilizar?. Los candidatos de solución ante este cuestionamiento fueron los siguientes conjuntos de instrucciones: MIPS, OpenRisc y RISC-V.

El conjunto de instrucciones MIPS es uno de los más utilizados actualmente, pese a existir gran cantidad de información al respecto como por ejemplo la que se muestra en [2, 44, 45]; presenta el inconveniente de que en el 2012 [46], los derechos comerciales de la patente MIPS fueron adquiridos por la empresa Imagination Technologies [47], imposibilitando tan siquiera considerar a este conjunto de instrucciones como una opción, esto porque, es necesario realizar trámites sobre el uso de la patente y probablemente requieren de una

Tabla 3.2: Comparación utilizada para seleccionar cuál arquitectura del conjunto de instrucciones debe utilizarse : RISC-V u OpenRISC. Cada aspecto tiene un peso máximo de cinco unidades, siendo cinco la máxima puntuación.

Aspecto	**RISC-V**	**OpenRISC**
Mantenimiento	4	2
Soporte en línea	5	3
Información disponible	4	3
Usabilidad	5	5
Tiempo en el mercado	3	5
Peso total:	21	18

inversión económica alta, por lo consiguiente se descartó de inmediato esta alternativa.

Por esta razón, se decidió migrar hacia alternativas de arquitecturas del conjunto de instrucciones abiertas: OpenRisc y RISC-V. La primera es una arquitectura del conjunto de instrucciones de software abierto, simple como el MIPS tradicional usando tres operandos y operaciones a memoria de tipo *LOAD* y *STORE*, con 16 o 32 registros de propósito general y una longitud fija de instrucciones de 32 bits. El conjunto de instrucciones, es principalmente idéntica entre los 32 y 64 bits de la especificación. Sin embargo, la principal diferencia se percibe en el ancho de registro (32 o 64 bits) y el diseño tabla de páginas [48].

RISC-V por su parte, es una arquitectura del conjunto de instrucciones que se desarrolló originalmente en la División de Ciencias de la Computación del Departamento de Ingeniería Eléctrica e Informática en la Universidad de California, Berkeley. Se creó en sus inicios, para apoyar la investigación en el área de arquitectura de computadoras y actualmente se ha convertido en un estándar abierto para muchos investigadores en el área [49].

RISC-V a pesar de ser un estándar relativamente nuevo en comparación con OpenRISC, ha demostrado tener múltiples características que lo ponen arriba sobre OpenRISC, entre ellas se encuentra por ejemplo que OpenRisc no provee un espacio de direccionamiento de 64 bits a diferencia de RISC-V. OpenRISC no tiene soporte para instrucciones comprimidas, permitiendo las mismas apoyar una codificación de instrucciones densa para reducir el tamaño de código estático en sistemas embebidos y asimismo, disminuir el tráfico de instrucciones dinámicas en servidores de gama alta como sí lo hace RISC-V [50].

Si se analiza críticamente ambos conjuntos de instrucciones, es posible determinar que para la aplicación desarrollada, al ser muy específica, ambos podrían seleccionarse y funcionarían adecuadamente para los objetivos de la investigación. Por esta razón, se elaboró la tabla 3.2, donde se seleccionaron cinco aspectos principales y se evaluaron los mismos sobre RISC-V y OpenRISC. A cada aspecto, se le asignó un peso de cinco unidades, siendo este la calificación más alta. De esta forma, se llegó a la conclusión de que la mejor alternativa para trabajar en la tesis fue RISC-V.

3.2 Selección de la arquitectura interna del microprocesador

En este apartado, se muestra los detalles que permitieron realizar una selección adecuada de la arquitectura interna del microprocesador. Cuando se habla de una arquitectura interna de un microprocesador, la definición más simple se basa en especificar cómo dentro de un microprocesador las instrucciones van a ser decodificadas para la ejecución de las mismas. Existen distintos tipos de arquitecturas entre ellas se encuentran: uniciclo, segmentada, multi core, multi hilo, superescalar, VLIW (acrónimo inglés de Very Long Instruction Word), entre otras [44].

Para seleccionar la arquitectura adecuadamente, se consideraron especialmente dos de los requerimientos principales que debe cumplir el microprocesador: bajo consumo de potencia y capacidad de funcionar adecuadamente a la frecuencia de trabajo mínima de 100MHz. Para que la arquitectura tenga un bajo consumo de potencia es conveniente dejar de lado las arquitecturas avanzadas y mirar hacia las arquitecturas más simples, que en su defecto son: uniciclo y segmentada. Esto porque, las arquitecturas avanzadas tienen como objetivo mejorar el rendimiento en cuanto a ejecución de instrucciones, sin considerar en primera instancia el consumo asociado de potencia [2, 44].

En el caso de una arquitectura uniciclo, se ejecuta una única instrucción en tan solo un ciclo de reloj. En la figura 3.2 se muestra la ruta de datos típica para una instrucción tipo R (operación sobre dos registros) [2]. Sin embargo, esta primer arquitectura se descartó debido a que si se observa en detalle, a pesar de ser una solución funcionalmente correcta, tiene el impedimento de que debido a que una instrucción se ejecuta en solo un ciclo de reloj, se tienen rutas combinacionales largas sin registros intermedios lo que claramente va a originar problemas temporales haciendo además los caminos combinacionales largos por lo que inducen a un consumo potencia dinámico por glitches. Esta arquitectura, tiene el problema asociado de que la velocidad del sistema esta limitado por la operación más lenta [51, 52].

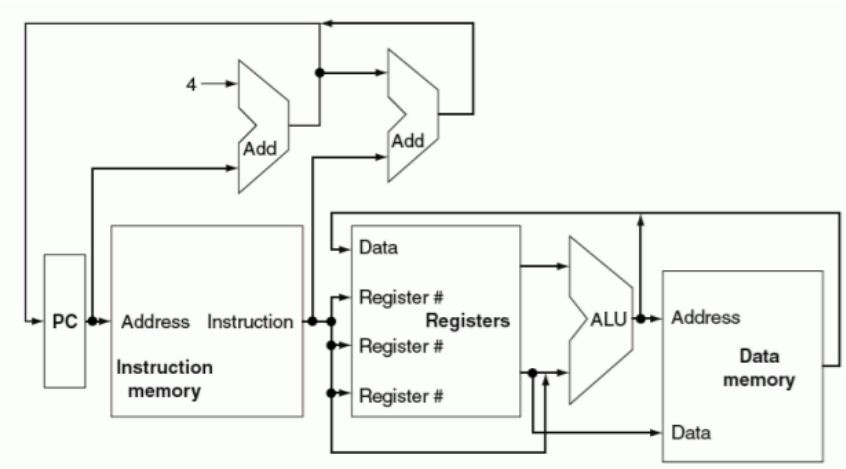

Figura 3.2: Arquitectura uniciclo. Imagen tomada de [2].

La arquitectura segmentada por su parte, vienen a solucionar el problema de temporizado que presenta la arquitectura descrita anteriormente y es más ahorrativa en potencia pues, hay menos consumo por glitches. Esto se logró mediante la incorporación de registros en la ruta de datos, reduciendo el tamaño de las rutas combinacionales largas y por ende se mejora el temporizado.

Para comprender la mejora introducida en el rendimiento que produce la incorporación de registros, se debe considerar la figura 3.3 a) donde se observa una ruta combinacional con un tiempo de propagación T_p, suponiendo que esta es la ruta crítica, es posible observar que el máximo reloj que puede tener el sistema digital es semejante a $1/T_p$. En cambio si se observa la figura 3.3 b), es posible observar como la incorporación de dos registros en medio de la ruta combinacional origina que el tiempo máximo de propagación de las señales se divida en tres nuevos retardos de propagación T_{p1}, T_{p2} y T_{p3}, con $Tp \approx T_{p1} + T_{p2} + T_{p3}$. Así,la máxima frecuencia de operación esta dada por el mayor tiempo de propagación entre T_{p1}, T_{p2} y T_{p3}.

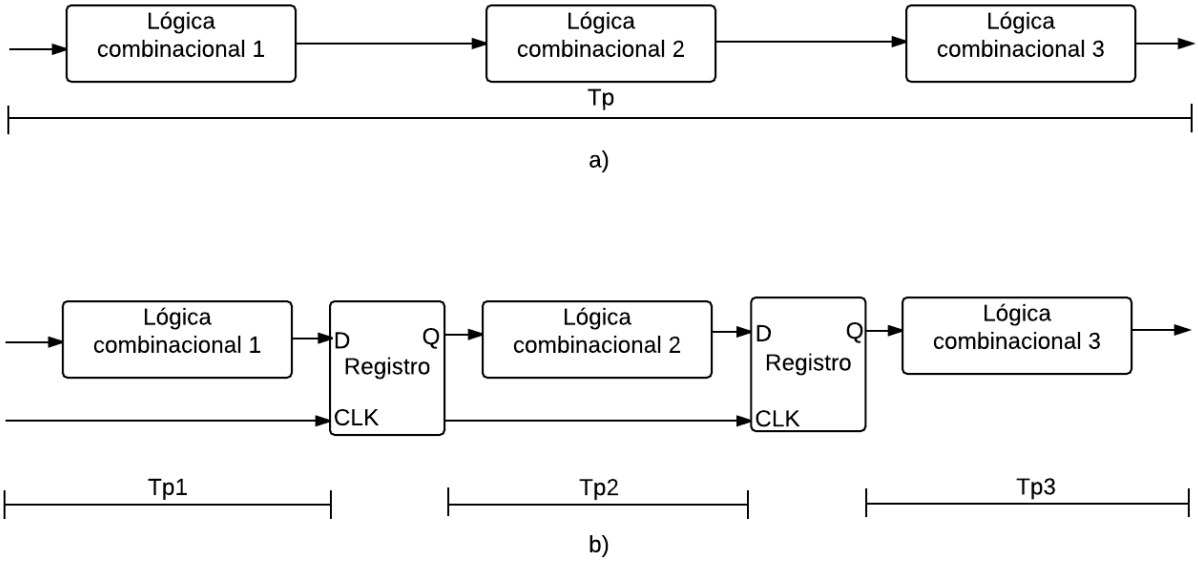

Figura 3.3: a) Ruta combinacional con tiempo de propagación grande debido a la inexistencia de registros intermedio. b) Incremento de la frecuencia de operación con la incorporación de registros entre rutas combinacionales largas.

Sin embargo, cuando se habla de una arquitectura segmentada es posible identificar dos tipos en específico: uniciclo y multiciclo. La multiciclo como la que se muestra en la figura 3.4 es un tipo de arquitectura donde la ruta de datos para cada instrucción se parte en segmentos, cada segmento está compuesto de una lógica combinacional separada entre un registro a la entrada y un registro a la salida. La ejecución es simple ya que para cada una de las instrucciones su ejecución se parte en distintas etapas hasta completar en cierta cantidad de ciclos la ejecución de una instrucción [2, 44].

Por otra parte, la arquitectura segmentada uniciclo como la que se muestra en la figura 3.5 es un tipo de arquitectura donde al igual que la multiciclo la ruta de datos para cada instrucción se parte en segmentos. La diferencia que existe con respecto a la multiciclo,

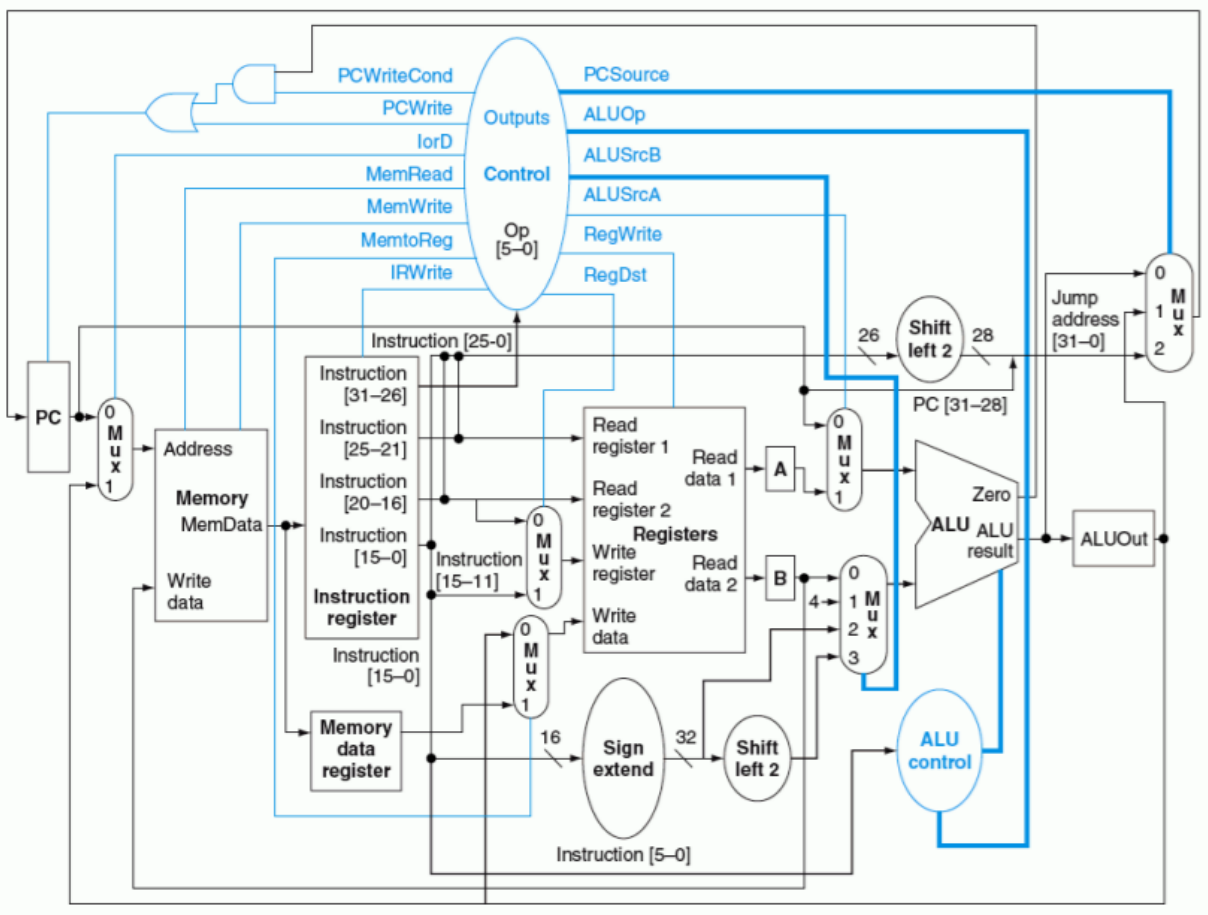

Figura 3.4: Arquitectura segmentada multiciclo: En color negro la ruta de datos y en color azul la ruta de control. Imagen tomada de [2].

es que aprovecha al máximo la ejecución en los distintos segmentos, esto porque mientras en un segmento se está ejecutando una instrucción, en el segmento que sigue se está ejecutando otra instrucción y así con todos los distintos segmentos. De esta forma en todo momento se estará ejecutando una instrucción en alguno de los segmentos y por lo tanto es de esperar que con cada ciclo de reloj una instrucción finalice su ejecución, de ahí es donde surge el nombre de arquitectura segmentada uniciclo [2, 44].

Es claro que ambas arquitecturas al tener el mismo enfoque de segmentación pueden operar a la misma frecuencia de operación. De hecho, para la misma tecnología y el mismo diseño, es de esperar que estas se puedan ejecutar a la misma frecuencia máxima de operación. Ahora, si se analiza el rendimiento de ambas arquitecturas, fácilmente se puede observar que la sementada uniciclo tiene un CPI (acrónimo inglés de instrucciones por ciclo) idealmente cercano a la unidad mientras que para el caso de la arquitectura segmentada multiciclo el CPI no alcanza tan siquiera el 0.25

Si ambas arquitecturas se analizan desde el punto de vista de consumo de potencia, en este caso las cosas cambian, ya que, en la arquitectura segmentada uniciclo, para lograr la ejecución de una instrucción por ciclo necesita agregar distintas unidades de control. Específicamente se agrega un hardware extra tanto para la detección de riesgos de datos

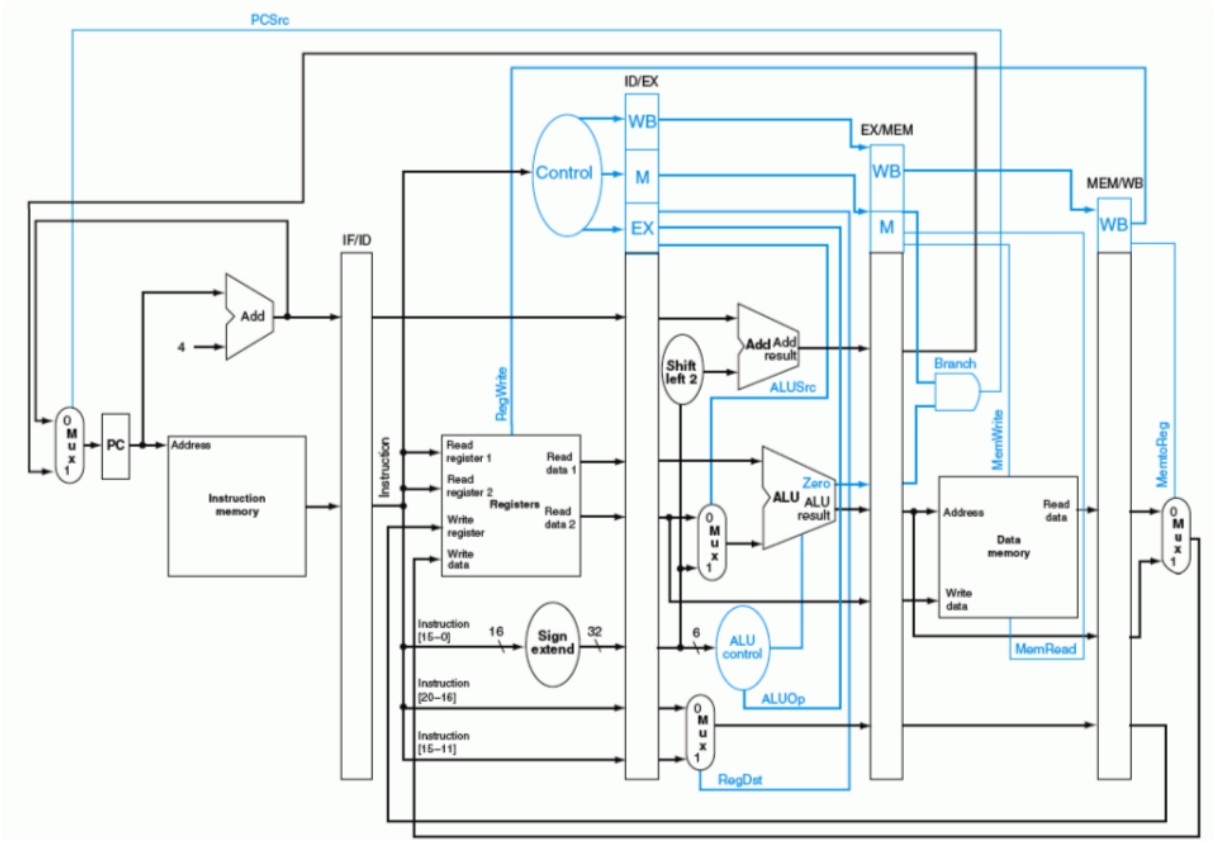

Figura 3.5: Arquitectura segmentada uniciclo: En color negro la ruta de datos y en color azul la ruta de control. Imagen tomada de [2].

como para la detección de riesgos de control. La incorporación de este hardware extra origina tanto un incremento del área como un incremento en la potencia. En su lugar la arquitectura segmentada multiciclo no presenta riesgos de datos ni de control, por lo que no es necesario adicionar estas unidades funcionales [2, 44].

Debido a esta razón, y recordando que el objetivo principal del microprocesador es que sea implementado en una arquitectura que tenga un bajo consumo de potencia, se determinó que a pesar de que el rendimiento de la arquitectura segmentada uniciclo es mejor, con solo el rendimiento de la arquitectura multiciclo basta para el desarrollo de la aplicación, por lo que esta fue seleccionada.

3.3 Selección de las instrucciones implementadas

Una vez definida la arquitectura interna y el conjunto de instrucciones para el ASP, se seleccionó la base entera que se iba a implementar. RISC-V tiene dos estándares de base entera [17] que se diferencian por el tamaño del bus de datos y el espacio de direccionamiento : RV32I (32 bits) y RV64I (64 bits).

Debido a que el microprocesador es de aplicación específica y dada la necesidad de ahorro

Tabla 3.3: Instrucciones implementadas para la base entera RV32I con codificación U, UJ, SB y S.

Intrucción	Codificación	Descripción funcional
LUI	U	$Rd = \{imm[31:12] : 12d0\}$
AUIPC	U	$Rd = \{imm[31:12] : 12d0\} + PC$
JAL	UJ	$PC = PC + 4 + ExtSigno\{inm\}$, $Rd = PC + 4$
BEQ	SB	Salta si $[RS1] = [RS2]$
BNE	SB	Salta si $[RS1] \neq [RS2]$
BLT	SB	Salta si $[RS1] < [RS2]$, con signo
BGE	SB	Salta si $[RS1] \geq [RS2]$, con signo
BLTU	SB	Salta si $[RS1] < [RS2]$, sin signo
BGEU	SB	Salta si $[RS1] \geq [RS2]$, sin signo
SB	S	$MEM\,[RS1 + inm]\,[7:0] = [RS2]$
SH	S	$MEM\,[RS1 + inm]\,[15:0] = [RS2]$
SW	S	$MEM\,[RS1 + inm]\,[31:0] = [RS2]$

de recursos para disminuir el área y la potencia del mismo, y además analizando el algoritmo de modelos ocultos de Markov para la aplicación, es fácil deducir que no es necesario operaciones entre números enteros de valores mayores a los que permite representarse con 32 bits. Por esta razón, se determinó que utilizar una base entera de 32 bits brinda los mejores resultados relacionándolo con los requerimientos establecidos para este trabajo.

Posteriormente, se seleccionó cuáles instrucciones de esa base entera se iban a requerir. Para ello, tras un estudio del algoritmo se determinó que las instrucciones de las tablas 3.3 3.4 y 3.5 son las que se tenían que implementarse dentro del microprocesador.

Las instrucciones de la base entera RV32I que no se implementaron fueron las que van orientadas al modelo de la memoria y las instrucciones del sistema. Las primeras se utilizan para el manejo de procesos concurrentes para la ejecución de múltiples hilos. Cada hilo tiene su propio registro de estado de usuario y su contador de programa y ejecutan instrucciones independientes. Las dos instrucciones de modelo del sistema son: *FENCE* y *FENCE.I*. Las instrucciones del sistema se utilizan para acceder a funcionalidades que requieren privilegios de acceso. Entre ellas se encuentran las siguientes instrucciones: *SCALL*, *SBREAK*, *RDCYCLE*, *RDCYCLEH*, *RDTIME*, *RDTIMEH*, *RDINSTRET* y *RDINSTRETF* [17].

Este tipo de instrucciones no se implementaron debido a que al ser una única aplicación de propósito específico y debido a la naturaleza del mismo, no se tiene contemplado utilizar hilos ni tampoco se tiene la necesidad de tener un sistema operativo ejecutándose. Así, es fácil observar que estas instrucciones pueden no implementarse sin repercutir esto en el comportamiento de la aplicación y disminuyendo el consumo de potencia de la misma, originando esto una reducción en el hardware implementado.

Una vez definido el estándar base que se iba a utilizar y definidas también cuáles de estas

Tabla 3.4: Instrucciones implementadas para la base entera RV32I con codificación I.

Intrucción	**Descripción funcional**
JALR	$PC = RS1 + imm$, $Rd = PC + 4$
LB	$Rd = ExtSigno\{MEM\,[RS1 + inm]\,[7:0]\}$
LH	$Rd = ExtSigno\{MEM\,[RS1 + inm]\,[15:0]\}$
LW	$Rd = ExtSigno\{MEM\,[RS1 + inm]\,[31:0]\}$
LBU	$Rd = ExtCero\{MEM\,[RS1 + inm]\,[7:0]\}$
LHU	$Rd = ExtCero\{MEM\,[RS1 + inm]\,[15:0]\}$
ADDI	$Rd = RS1 + ExtSigno\{inm\}$
SLTI	Si $RS1 < ExtSigno\{inm\}$,$Rd = 1$ sino $Rd = 0$
SLTIU	Si $RS1 < ExtSigno\{inm\}$,$Rd = 1$ sino $Rd = 0$ sin signo
XORI	$Rd = RS1\hat{}ExtSigno\{inm\}$
ORI	$Rd = RS1 \mid ExtSigno\{inm\}$
ANDI	$Rd = RS1\&ExtSigno\{inm\}$
SLLI	$Rd = RS1 \ll inm[4:0]$
SRLI	$Rd = RS1 \gg inm[4:0]$
SRAI	$Rd = RS1 \ggg inm[4:0]$

Tabla 3.5: Instrucciones implementadas para la base entera RV32I con codificación R.

Intrucción	**Descripción funcional**
ADD	$Rd = RS1 + RS2$
SUB	$Rd = RS1 - RS2$
SLL	$Rd = RS1 \ll RS2[4:0]$
SLT	Si $RS1 < RS2$,$Rd = 1$ sino $Rd = 0$
SLTU	Si $RS1 < RS2$,$Rd = 1$ sino $Rd = 0$ sin signo
XOR	$Rd = RS1\hat{}RS2$
SRL	$Rd = RS1 \gg RS2[4:0]$
SRA	$Rd = RS1 \ggg RS2[4:0]$
OR	$Rd = RS1 \mid RS2$
AND	$Rd = RS1\&RS2$

Tabla 3.6: Instrucciones implementadas para la extensión M con codificación R.

Instrucción	Descripción funcional
MUL	$Rd = RS1 * RS2$ en Rd se copian los 32 bits LSB
MULH	$Rd = RS1 * RS2$ en Rd se copian los 32 bits MSB, supone $RS1$ y $RS2$ con signo
MULHSU	$Rd = RS1 * RS2$ en Rd se copian los 32 bits MSB, supone $RS1$ sin signo y $RS2$ con signo
MULHU	$Rd = RS1 * RS2$ en Rd se copian los 32 bits MSB, supone $RS1$ y $RS2$ sin signo

instrucciones se debían implementar, se procedió a determinar cuáles extensiones adicionales se requerían. Además de los dos estándares base, RISC-V agrupada en extensiones algunas otras instrucciones. Las que resultaban de utilidad en este trabajo fueron: M, F y D [17].

La extensión M es una extensión estándar para el manejo de multiplicación y división entera. RISC-V agrupa las multiplicaciones y divisiones enteras en un estándar adicional porque en algunos tipos de aplicaciones, no resulta necesario la utilización de multiplicaciones o divisiones enteras o son manejadas por medio de aceleradores de hardware; de ahí la razón por la que esta modalidad no se encuentra decodificada dentro de las bases enteras [17].

La extensión F es una extensión estándar para el manejo de operaciones en punto flotante de precisión simple. RISC-V agrupa un conjunto de instrucciones que son exclusivas para el manejo de datos en notación IEEE 754. Este estándar además, agrega 32 nuevos registros de propósito específico para el manejo de estas instrucciones de tamaño igual a 32 bits. Además, RISC-V cuenta con la extensión D que es una extensión estándar para el manejo de operaciones en punto flotante con precisión doble. Esta extensión amplía el ancho de los registros de propósito específico a 64 bits [17].

En las tablas 3.6 y 3.7 se muestra las instrucciones implementadas que pertenecen al grupo de las extensiones M, F y D necesarias para la aplicación. Se menciona nuevamente que no todas las instrucciones que estas extensiones poseen se implementaron. Lo que se realizó fue una selección adecuada de las mismas en función de la aplicación para determinar cuáles son las que se necesitan realmente.

3.4 Administración de la memoria

En esta sección, se describe la administración de la memoria dentro del microprocesador. El enfoque de implementación de la memoria, se basó en una arquitectura de memoria de tipo Hardvare. Es decir, se tiene una memoria para datos y una para programa físicamente separadas, donde existen dos rutas distintas para el acceso de datos o de

Tabla 3.7: Instrucciones implementadas para la extensión F y D con codificación I, S, R4 y R.

Instrucción	Codificación	Extensión	Descripción funcional
FLD, FLW	I	D, F	$Rd = MEM\,[RS1 + inm]\,[31:0]$
FSD, FLW	S	D, F	$MEM\,[RS1 + inm]\,[31:0] = [RS2]$
FMADD	R4	D, F	$Rd = RS1 * RS2 + RS3$
FMSUB	R4	D, F	$Rd = RS1 * RS2 - RS3$
FNMSUB	R4	D, F	$Rd = -(RS1 * RS2 - RS3)$
FNMADD	R4	D, F	$Rd = -(RS1 * RS2 + RS3)$
FADD	R	D, F	$Rd = RS1 + RS2$
FSUB	R	D, F	$Rd = RS1 - RS2$
FMUL	R	D, F	$Rd = RS1 * RS2$
FSGNJ	R	D, F	$Rd = \{RS2\,[31]\,, RS1\,[30:0]\}$
FSGNJN	R	D, F	$Rd = \{\sim RS2\,[31]\,, RS1\,[30:0]\}$
FSGNJX	R	D, F	$Rd = \{RS2\,[31]\,\hat{}\,RS1\,[31]\,, RS1\,[30:0]\}$
FCVT.S.D	R	D	$Rd = Simple \quad a \quad Doble([\text{RS1 }])$
FCVT.D.S	R	D	$Rd = Doble \quad a \quad Simple([\text{RS1 }])$
FEQ	R	D, F	Si $[RS1] = [RS2]$, $Rd = 1$
FLT	R	D, F	Si $[RS1] < [RS2]$, $Rd = 1$
FLE	R	D, F	Si $[RS1] \leq [RS2]$, $Rd = 1$

instrucciones. En la figura 3.6 se muestra un diagrama de bloques, donde se puede observar el microprocesador intercomunicado con dos memorias: una para datos y otra para instrucciones; y los puertos de entrada/salida [53].

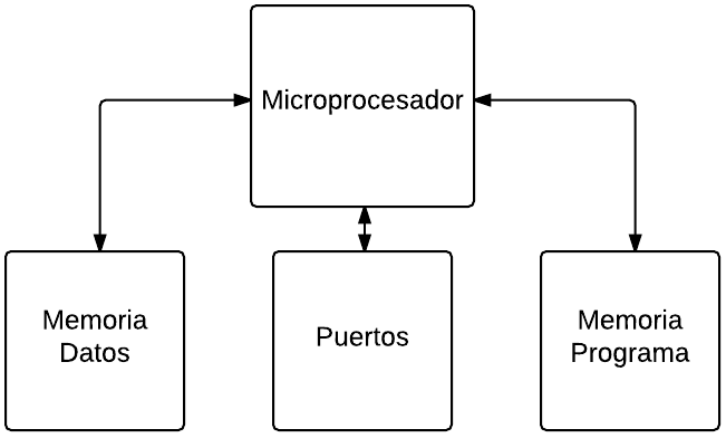

Figura 3.6: Arquitectura de memoria Hardvard, utilizada para el microprocesador.

En las siguientes subsecciones se presenta el mapa de memoria, la interfaz de memoria de datos, la interfaz de la memoria de programa y el manejo de puertos utilizadas en el microprocesdador.

3.4.1 Mapa de memoria

En la figura 3.7 se muestra el mapa de memoria del microprocesador. El mismo inicia en la dirección física 0x00000 con el vector de reinicio. De la dirección física 0x00004 a la 0x001FF se encuentran atados los puertos de entrada y salida del microprocesador, es decir, los puertos de entrada y salida se encuentran igualmente mapeados en la memoria. De la dirección 0x00200 a la dirección 0x003FF se encuentra los puertos para el manejo de interrupciones. En la dirección de memoria 0x00400 a la dirección 0x103FF se encuentra la memoria de datos y el programa esta contenido desde la dirección 0x20000 hasta la dirección 0x21FFF.

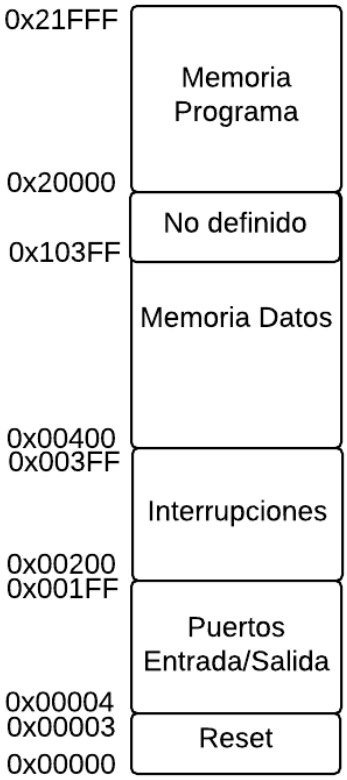

Figura 3.7: Mapa de memoria del microprocesador de aplicación específica.

Un aspecto importante en relación con el mapa de memoria de la figura 3.7 es que este, por medio de la modificación de parámetros, fácilmente puede extender o disminuir su tamaño. De forma que, la distribución de la memoria de la figura 3.7, representa únicamente la distribución de la memoria para la aplicación específica en estudio.

3.4.2 Interfaz de memoria de programa

En la figura 3.8 se muestra cómo se realiza la lectura de la memoria de instrucciones. La interfaz de memoria de programa está constituida por cuatro componentes digitales básicos: un contador, un multiplexor, un sumador, una memoria ROM (sin embargo,

Tabla 3.8: Descripción de los parámetros necesarios para ajustar el tamaño adecuado de la memoria de programa dentro del ASP implementado.

Parámetro	Funcionalidad
ROM_ADDR_BITS	Número de bits del bus de direcciones.
ROM_WIDTH	Tamaño en bits de los datos que almacena la ROM. Por defecto 8 bits.
BEGIN_ADDR_ROM_PROGRAM	Dirección de inicio del programa.
END_ADDR_ROM_PROGRAM	Dirección final del programa.
ProgramStartAddressPC	Dirección de la etiqueta "main" del programa.
ROM_ADDR_START_BITS	Es el número de bits de offset. Por defecto como inicia en la dirección 0x8000 el offset es de 15 bits.

esta puede adaptarse fácilmente a una memoria RAM) y un registro. Se tiene un multiplexor a la entrada donde la máquina de estados selecciona si se aumenta el contador de programa en cuatro o se realiza una carga de un valor arbitrario al contador. Una vez que el contador de programa se incrementa, se apunta hacia la siguiente instrucción. Consecuentemente, se translada el contenido de la memoria de programa en el registro de instrucciones y nuevamente se repite el ciclo. Lo anterior, se conoce como ciclo de búsqueda de instrucciones [2].

Se debe observar en la figura 3.8, que el bus de direcciones de la memoria de programa es de solo diez bits. Esto constituye un aspecto de implementación que se consideró con cuidado para reducir el impacto de tener memorias grandes. Por medio de un parámetro se puede ajustar estáticamente el tamaño de la memoria tanto de datos como de instrucciones, para calibrar el tamaño de la memoria y ahorrar espacio que repercute en área y en consumo de potencia estático.

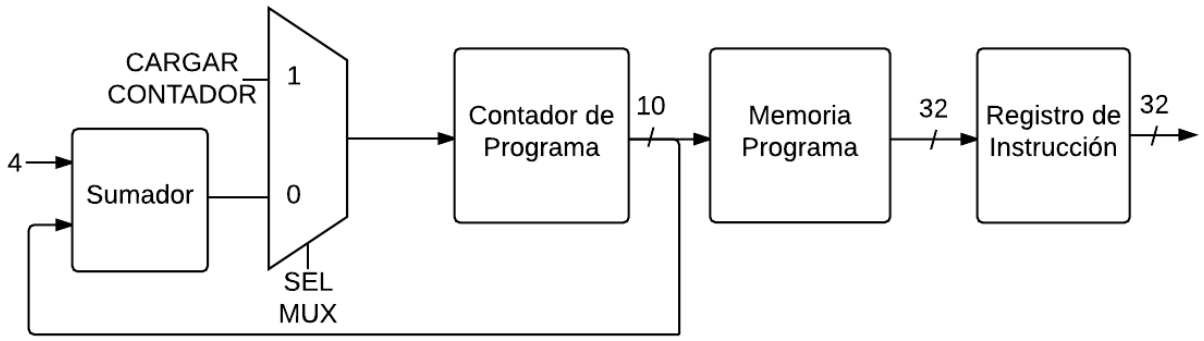

Figura 3.8: Diagrama de bloques de la memoria de programa implementada en este trabajo.

Para el ajuste del tamaño adecuado de la memoria de programa y como parte de la implementación de la misma en el lenguaje de descripción de hardware, se tiene un conjunto de parámetros globales que permiten ajustar adecuadamente el tamaño de la memoria. Los mismos y su funcionalidad se describieron en la tabla 3.8.

3.4.3 Interfaz de memoria de datos

Para la memoria de datos se elaboró en un lenguaje de descripción de hardware, un circuito con la interfaz como la que se muestra en la figura 3.9; donde se contaba con siete señales de entrada y una salida. Las señales son: señal de reloj, un reset para los registros internos de la interfaz, una señal Enable que se encarga de habilitar la lectura o escritura de la memoria, una señal Write que se encarga de habilitar la escritura y un bus de direcciones de 16 bits necesarios para direccionar datos entre la dirección de memoria física 0x00400 hasta la dirección 0x103FF. Finalmente, la señal *funct3*, son tres bits que son parte de la codificación de las instrucciones *load* y *store* y sirve para determinar si se esta trabajando con bytes (ocho bits), media palabra (16 bits), una palabra (32 bits), precisión simple IEEE 754 (32 bits) o precisión doble IEEE 754 (64 bits).

Para realizar una lectura o escritura de memoria se implementaron las instrucciones del RISC-V: *LW* (carga 32 bits),*LH* (carga 16 bits), *LB* (carga ocho bits), *SB* (almacena ocho bits), *SH* (almacena 16 bits), *SW* (almacena 32 bits), *FLW* (carga 32 bits), *FSW* (almacena 32 bits), *FSD* (almacena 64 bits) y *FLD* (carga 64 bits).

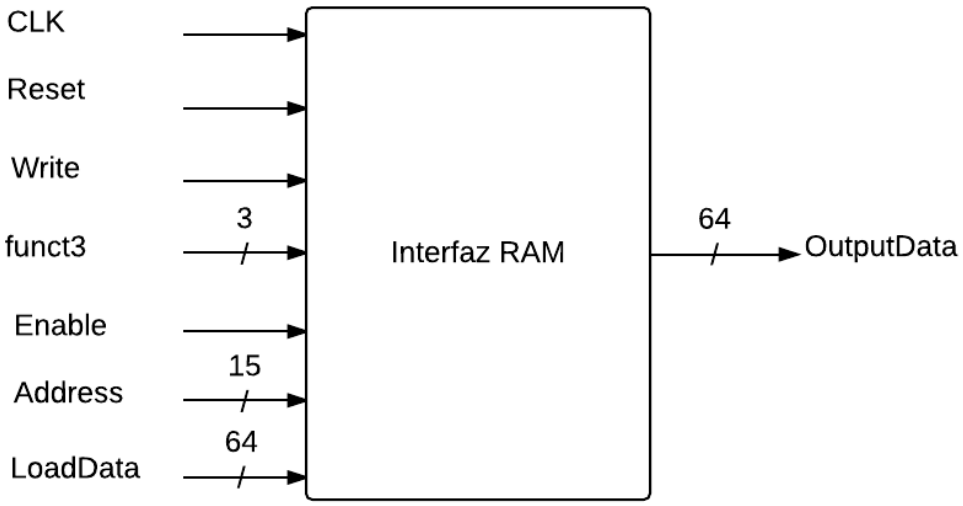

Figura 3.9: Diagrama de puertos de la interfaz de memoria de datos implementada en este trabajo.

De igual forma que sucedió con la implementación de la memoria de programa, dentro de la descripción de hardware de la memoria de datos, se agregaron una serie de puertos parametrizables que controlaban: el tamaño del bus de datos, la capacidad de almacenamiento, cantidad de datos precargadados, entre otros. El objetivo de este ajuste fue reducir área y disminuir consumo de potencia estática, así como evitar el desperdiciar recursos innecesarios. En la tabla 3.9 se muestran la lista de los parámetros que gobierna la interfaz de memoria RAM y su funcionalidad.

El proceso de lectura y escritura en la memoria RAM utilizada para almacenar datos se compone de una secuencia de pasos que se observan en la figura 3.10. Cada uno de estos pasos corresponde a un ciclo de reloj, por lo que es fácil observar que se necesitan tres

Tabla 3.9: Descripción de la funcionalidad de los parámetros para ajustar el tamaño adecuado de la memoria de datos dentro del microprocesador.

Parámetro	Funcionalidad
RAM_ADDR_BITS	Número de bits del bus de direcciones de la memoria.
RAM_WIDTH	Ancho de los datos de entrada y salida de la RAM. Por defecto se trabaja con memorias de 8 bits.
RAM_ADDR_START_BITS	Es el número de bits de offset. Por defecto inicia en la dirección 0x0400. Se requieren 10 bits de offset. para iniciar en la dirección 0x0400.
InicializarRAM	Si es igual a uno significa que la RAM tendrá valores precargados en memoria se utiliza para almacenar las constantes del modelo del algoritmo.
DIRInicioInicializarRAM	Dirección de memoria donde se desea que inicien los datos cargados en memoria.
DIRFinInicializarRAM	Dirección de memoria donde se desea que finalicen los datos cargados en memoria.

ciclos de reloj para realizar una escritura de la memoria RAM y siete ciclos de reloj para realizar una lectura de la memoria RAM.

3.4.4 Manejo de puertos entrada y salida

El acceso a los puertos de entrada y salida dentro del microprocesador se realizó por medio de las instrucciones *Load* y *Store*. Como estas instrucciones también se utilizaron para acceder los datos de la memoria RAM, fue necesario diferenciar de alguna manera cuando se está accediendo un puerto o la memoria. Por esta razón, cada vez que se calcula la dirección efectiva de acceso para estas instrucciones, si la misma es menor que 512 significa que se desea acceder un puerto en cuyo caso, se deshabilita la escritura/lectura de la RAM y se habilita la escritura/lectura desde un puerto. Lo que constituye básicamente un decodificador.

3.5 Unidad aritmética lógica entera y flotante

Para la ejecución de las operaciones aritmético/lógicas con representación numérica entera se cuenta con una ALU. Por su parte, la ejecución de operaciones aritméticas con números en representación IEEE 754 con precisión simple y doble se cuenta con una FPU que ha sido desarrollado en paralelo en el DCILAB, bajo la supervisión de este postulante y todos los detalles de la implementación de la misma se muestran en [54].

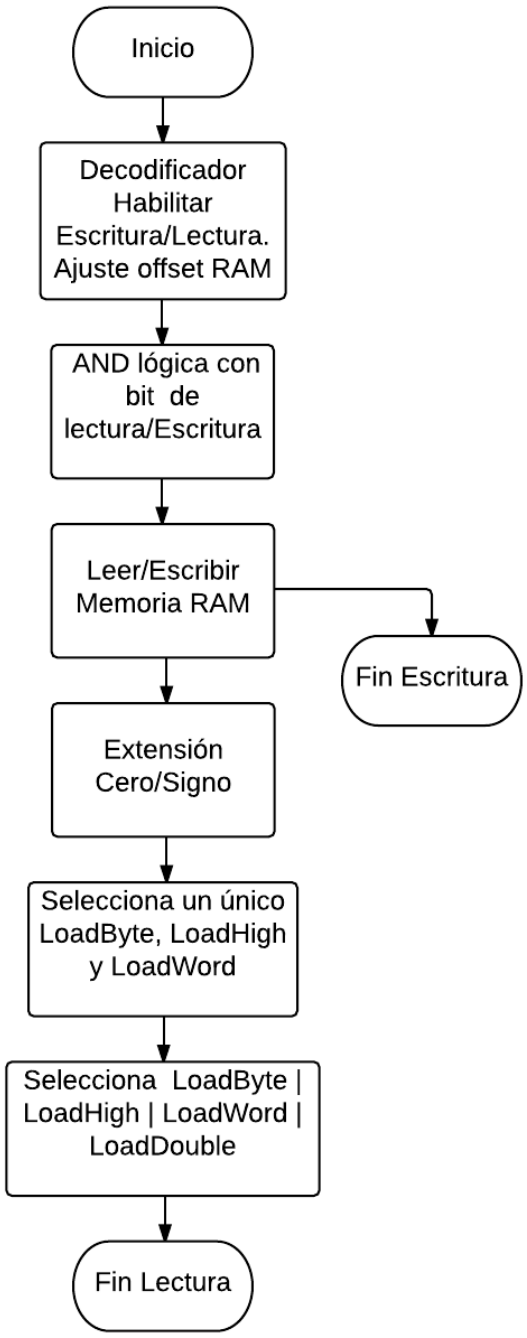

Figura 3.10: Diagrama de flujo del proceso de escritura y lectura de la memoria de datos.

3.5.1 Unidad aritmética lógica de representación entera

Las operaciones que realiza la ALU fueron tomadas de la base entera RV32I y la extensión de instrucciones específicamente las de tipo M. En la tabla 3.10 se muestran todas las operaciones que realiza la ALU y el código de operación del selector de la ALU para realizar las operaciones. Además, se presenta una pequeña descripción de la operación aritmética. Para más detalles de la funcionalidad de las operaciones se puede consultar [17].

Tabla 3.10: Operaciones que realiza la ALU.

Operación	Selector	Descripción
Add	0	Suma el contenido de dos registros.
SLL	1	Desplazamiento a la izquierda.
SLT	2	Coloca un uno si es menor que (con signo).
SLTU	3	Coloca un uno si es menor que (sin signo).
XOR	4	XOR lógica.
SRL	5	Desplazamiento a la derecha.
OR	6	OR lógica.
AND	7	AND lógica.
SUB	8	Resta el contenido de dos registros.
BEQ	12	Coloca un uno si los dos registros son iguales.
SRA	13	Desplazamiento aritmético.
IDLE	16	Ninguna operación, salida atada a cero.
MUL	24	Multiplicación entera, salida parte menos significativa.
MULH	25	Multiplicación entera, salida parte más significativa (dos operandos con representación con signo).
MULHSU	26	Multiplicación entera, salida parte más significativa (un operandos con representación con signo, el otro operando con representación sin signo).
MULHU	27	Multiplicación entera, salida parte más significativa (dos operandos con representación sin signo).

3.5.2 Unidad aritmética lógica de representación coma flotante

Para realizar operaciones con tipos flotantes (32 bits) o dobles (64 bits) se cuenta con una FPU que puede realizar las tres operaciones básicas de mayor uso y necesarias para el funcionamiento del algoritmo a saber: suma, resta y multiplicación. Además, se cuenta con soporte para comparaciones de tipo aritméticas entre números flotantes y operaciones de inyección de signo.

En relación con la suma, resta y multiplicación en coma flotante, cada una de estas unidades divide la ejecución de las operaciones en múltiples ciclos de reloj. En cada uno de los ciclos se completa una parte distinta del algoritmo. La duración de cada una de estas operaciones es función de las mantisas y el exponente y se lograron resultados con un porcentaje de error menor al 1%.

3.6 Implementación del microprocesador

Una vez codificadas en un lenguaje de descripción de hardware las principales unidades funcionales: ALU, FPU, administración de la memoria de datos, programa y puertos de entrada/salida, se integraron todas estas de manera que se obtuvo el desarrollo del microprocesador final.

Es posible observar que el primer paso en la implementación del microprocesador fue definir la interfaz de entrada y de salida. En la figura 3.11 se observa, el diagrama de entradas y salidas del microprocesador. En la tabla 3.11 es posible observar la descripción de la funcionalidad de cada uno de los puertos.

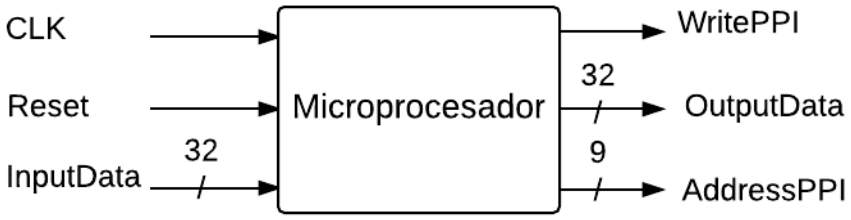

Figura 3.11: Diagrama de puertos de entrada/salida del microprocesador.

Se implementaron todas las instrucciones de las tablas 3.3, 3.4, 3.5, 3.6 y 3.7. Para ello se estableció como principal criterio de implementación no permitir rutas combinacionales largas para lograr una frecuencia de trabajo de al menos 100MHz. Por esta razón, se construyeron las instrucciones a partir de bloques básicos combinacionales colocadas entre registros. De esta manera, se logró un microprocesador segmentado multiciclo que fuera capaz de operar a la frecuencia deseada. En la figura 3.12 se muestra el diagrama de bloques del microprocesador de aplicación específica desarrollado.

En relación con la figura 3.12, se puede verificar que entre cada bloque combinacional existe un registro, tanto a la entrada como a la salida de estos. Además, un detalle

Tabla 3.11: Descripción de puertos de entrada y de salida del microprocesador implementado en esta tesis.

Puerto	Dirección	Número de Bits	Descripción
CLK	Entrada	1	Señal de reloj, frecuencia de operación 100MHz.
Reset	Entrada	1	Reinicio Maestro.
InputData	Entrada	32	Dato de Entrada proviene de algún PPI.
WritePPI	Salida	1	Bit que habilita la escritura de un PPI.
OutputData	Salida	32	Dato de Salida sirve para escribir un PPI.
AddressPPI	Salida	9	Dirección del PPI que se desea leer/escribir.

importante es que esta figura es posible dividirla en dos variantes: una con FPU y otra sin FPU. Esto porque la codificación del microprocesador, se realizó de tal forma que con solo la modificación de un parámetro es posible inhabilitar todo el soporte para instrucciones en coma flotante.

3.7 Interfaz de comunicación entre el microprocesador y la etapa de extracción de las características

Una vez implementado el núcleo del microprocesador descrito en las figuras 3.11 3.12, se realizó una interfaz de comunicación entre el microprocesador y la etapa anterior del SiRPA, conocida como árbol binario. El árbol binario genera un alfabeto discreto compuesto por 32 símbolos. Estos símbolos constituyen las observaciones de entrada del HMM. La salida por su parte, está compuesta de una señal binaria donde se indica si existe una señal de alerta.

En relación con la entrada del clasificador, se cuenta con una IPP (Interfaz periférica paralela) de entrada que almacena el número de observaciones necesarias para realizar una clasificación (el número de observaciones necesarias es función del modelo utilizado). Esta almacena los cinco bits discretos del símbolo que genera el árbol binario en un registro de desplazamiento. Cada vez que la etapa de identificación del SiRPA envía la señal que indica que un símbolo esta listo, se van desplazando estos a través del registro de desplazamiento. De esta forma, si por ejemplo el HMM utiliza una longitud de cadena de 20 símbolos, al tiempo que se tengan las 20 observaciones almacenadas, la primera observación estará guardada en los bits MSB del registro, mientras que en los bits LSB quedará almacenado la última de las 20 observaciones.

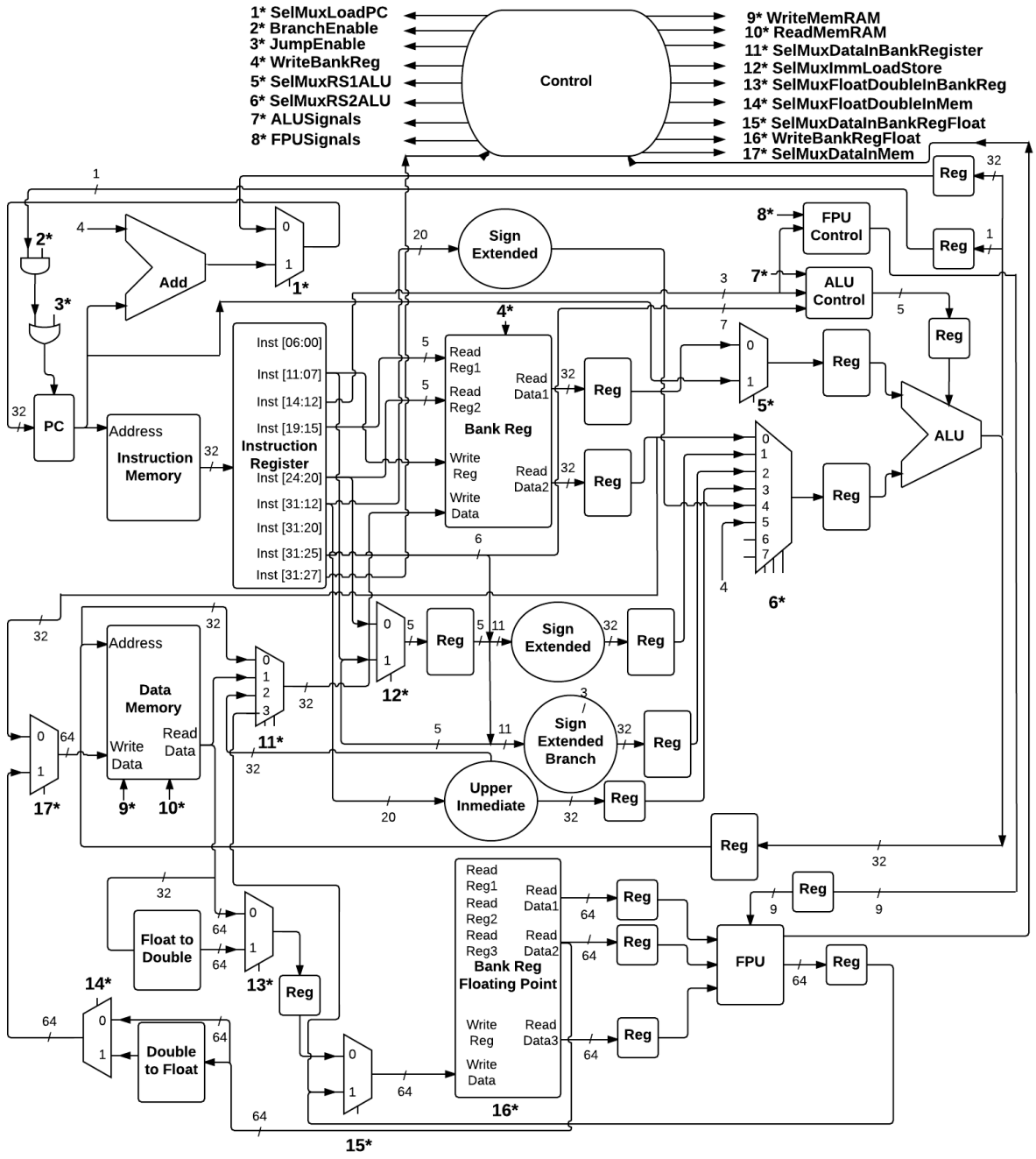

Figura 3.12: Diagrama de bloques del núcleo del microprocesador de aplicación específica desarrollado en esta tesis.

Consecutivamente, cada vez que la etapa de identificación ha enviado una cantidad de observaciones igual a la longitud de la cadena de observación definida por el modelo, esta generará una señal indicándole al clasificador que debe iniciar la clasificación. Esta señal se almacena en un segundo IPP de entrada, que es básicamente un registro, y cuando el microprocesador detecta mediante una interrupción por flanco positivo este evento, inicia la ejecución del algoritmo de HMM.

Para el caso de la salida se cuenta con una IPP. Esta consiste de un registro y un comparador que tiene asociada una dirección de memoria en el mapa de puertos. Esta además,

es direccionado por el microprocesador cada vez que finaliza la ejecución del algoritmo HMM para indicar el estado actual del SiRPA. En la figura 3.13, se muestra el diagrama de bloques del microprocesador intercomunicado con la etapa de identificación del SiRPA.

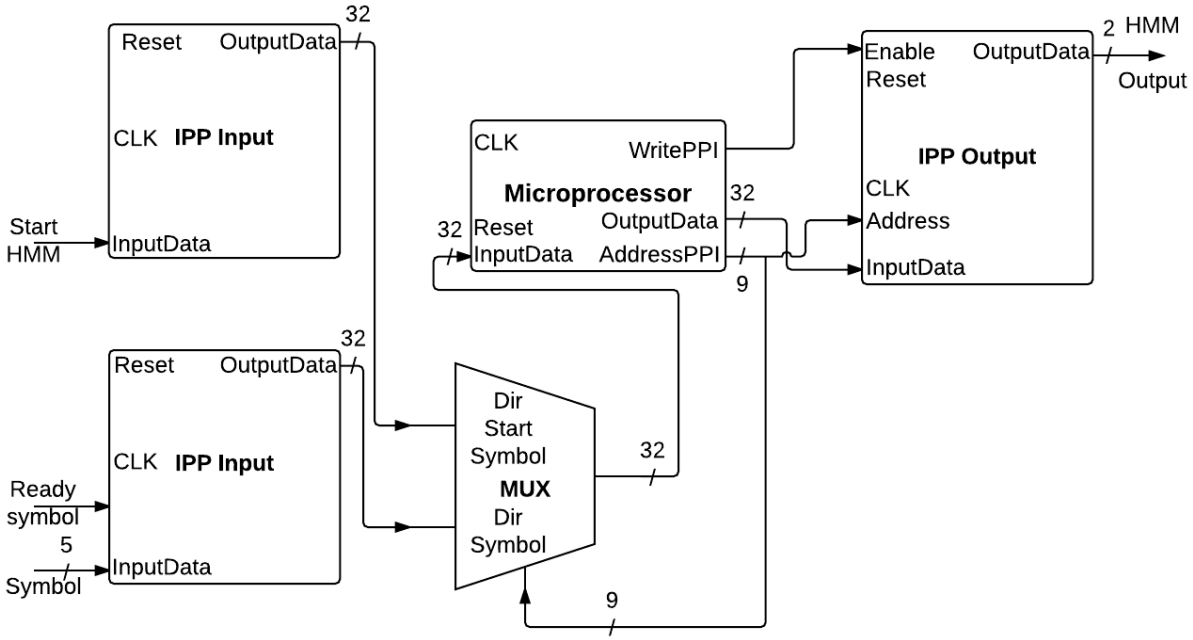

Figura 3.13: Diagrama de bloques del microprocesador intercomunicado con la etapa de identificación del sistema de reconocimiento de patrones acústicos.

3.8 Implementación del algoritmo de evaluación.

Para la implementación del software del algoritmo de evaluación de HMM, se realizó una modificación de la biblioteca [55] con el objetivo de adaptar el código a la arquitectura del ASP. El diagrama de flujo de la aplicación se resume en la figura 3.14.

Al inicio se cargan las constantes dentro de la memoria de datos. Luego, el sistema queda dentro de un ciclo infinito a la espera de que ocurra una interrupción. Cuando esta se origina (el evento es una señal que proviene de la etapa de identificación del SiRPA), se debe iniciar la ejecución del algoritmo. En ese momento el ASP realiza la lectura de un número de observaciones iguales a las definidas por el modelo. Estas observaciones se encuentran almacenadas dentro de un registro de desplazamiento y fueron guardadas ahí previamente por la etapa de extracción de características a una frecuencia de $344Hz$. Posteriormente, se ejecuta el algoritmo de HMM para cada una de las tres clases y se comparan los resultados obtenidos por el clasificador para tomar una decisión con respecto al estado actual del sistema. Finalmente el software, se sale de la interrupción y vuelve a quedar dentro del ciclo infinito hasta que ocurra la siguiente interrupción.

El algoritmo hacia adelante se muestra en el pseudocódigo descrito abajo, es posible observar en este las tres etapas fundamentales del algoritmo: cálculo del coeficiente α_0 inicial,

cálculo de los coeficientes α_t y finalmente el logaritmo de la probabilidad de una observación de una secuencia dada $log(P(0|\lambda))$. Es importante mencionar que el clasificador descrito es para evaluar si una cadena de observación pertenece a una clase en particular, en este caso como se tienen tres clases particulares: bosque, motosierra y disparo, es necesario aplicar tres veces el algoritmo para determinar si una cadena de observación pertenece a alguna de las tres clases.

```
T = 20; %% Longitud de la cadena de observación
N = 10; %% Número de estados ocultos

%% ********** Cálculo alpha(0,i) *****************
ct(0) = 0;
for i=0:N-1
alpha(0,i) = b(i,o(0))*pi(i);  %% Alpha
ct(0) = ct(0) + alpha(0,i);
end

%% Escala de coeficientes
ct(0) = 1/ct(0);

for i=0:N-1
alpha(0,i) = ct(0)*alpha(0,i);
end

%% ********** Cálculo alpha(t,i) *****************

for t=1:T-1      %% Recursividad
ct(t) = 0;
for i=0:N-1
z=0; %% Alpha
for j=0:N-1
z=z+a(i,j)*alpha(t-1,j);
end
alpha(t,i)=z*b(i,o(t));
ct(t) = ct(t) + alpha(t,i);
end

%% Escala de coeficientes

ct(t) = 1/ct(t);

for i=0:N-1
alpha(t,i) = ct(t)*alpha(t,i);
```

```
end
end

%% ********** Cálculo log[P(O|lambda)] ***************

Mult = 1;
for i=0:T-1
Mult = Mult*ct(i);
end
logProb = -log10(Mult);
```

Estudiando el pseudocódigo descrito anteriormente, es posible observar que se requiere realizar el cálculo de una función logaritmo y operaciones de división. Debido a que no se cuenta con hardware que evalúe el logaritmo decimal, ni tampoco hardware que realice la división fue necesario implementarlo por software. En cuanto al software, dado que la compilación es bare metal, no se puede utilizar ninguna biblioteca que requiera llamadas al sistema, y sucede que la división y el logaritmo Neperiano, requieren de estas. Por consiguiente, se construyeron dos bibliotecas matemáticas: la primera se encargaba de realizar divisiones por software y la segunda de realizar el cálculo del logaritmo Neperiano.

Para elaborar la biblioteca encargada de realizar la división por software, se seleccionó el método iterativo de Newton Raphson. La descripción elaborada del algoritmo se muestra en [56]. Básicamente, en este algoritmo se encuentra el recíproco del denominador y después se multiplica el resultado por el numerador, de esta forma se obtiene el cociente de la división.

Para elaborar la biblioteca encargada de realizar el cálculo del logaritmo Neperiano, se utilizó el algoritmo de Cordic (información más detallada puede encontrarse en [57]). Si se observa el pseudocódigo del algoritmo HMM, es posible identificar la necesidad de utilizar un logaritmo decimal. Sin embargo en términos del rendimiento, es más simple calcular un logaritmo decimal y luego multiplicar el resultado por una constante igual a ln(10), es decir, realizar un cambio de base en el logaritmo. En la ecuación (3.1) se representa la transformación de logaritmo Neperiano a logaritmo en base 10 utilizada en la biblioteca.

$$\log(x) = \frac{\ln(x)}{\ln(10)} \tag{3.1}$$

Un aspecto importante es que al igual que sucedió con las constantes del algoritmo de HMM, específicamente con las matrices $\mathbf{A}$, $\mathbf{B}$ y $\underline{\boldsymbol{\pi}}$ para bosque, motosierra y disparo, fue necesario almacenar las constantes que requiere la biblioteca de división y del logaritmo en la memoria RAM. Las constantes, así como las direcciones de memoria donde las mismas se almacenaron, pueden consultarse en las tablas A.1 y A.2. (Ver apéndice A).

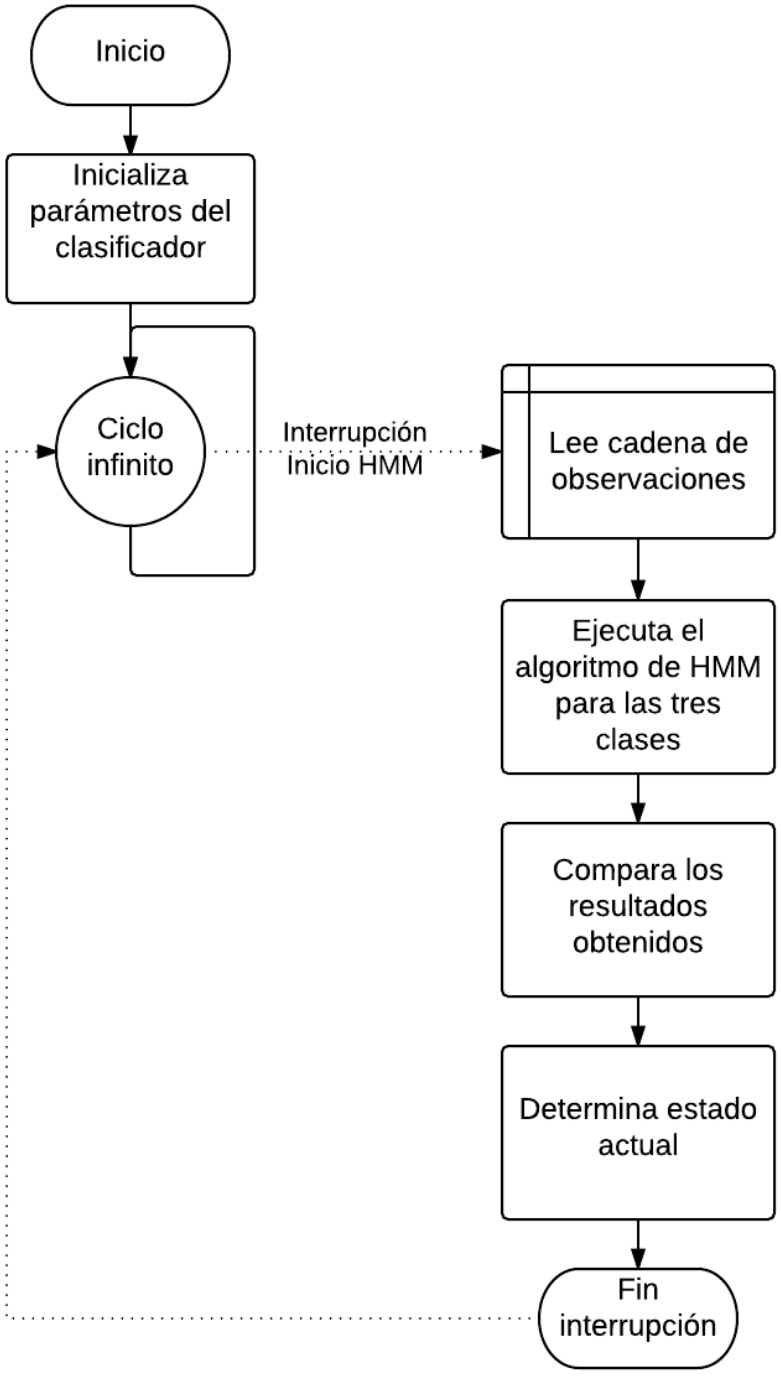

Figura 3.14: Diagrama de flujo de la ejecución del algoritmo de modelos ocultos de Markov en el microprocesador de aplicación específica.

Capítulo 4

Resultados y análisis

En este capítulo se muestran y discuten los resultados obtenidos del sistema desarrollado, partiendo de las pruebas aplicadas al mismo. Este se ha dividido en cuatro secciones según los resultados obtenidos: el ambiente de verificación utilizado para generar datos experimentales, datos relevantes del hardware, software y el sistema final interconectado, así como los principales datos relacionados con el funcionamiento del clasificador.

4.1 Ambiente de verificación

Para la generación de resultados y la verificación funcional del ASP, se desarrolló un ambiente de verificación como el que se muestra en la figura 4.1.

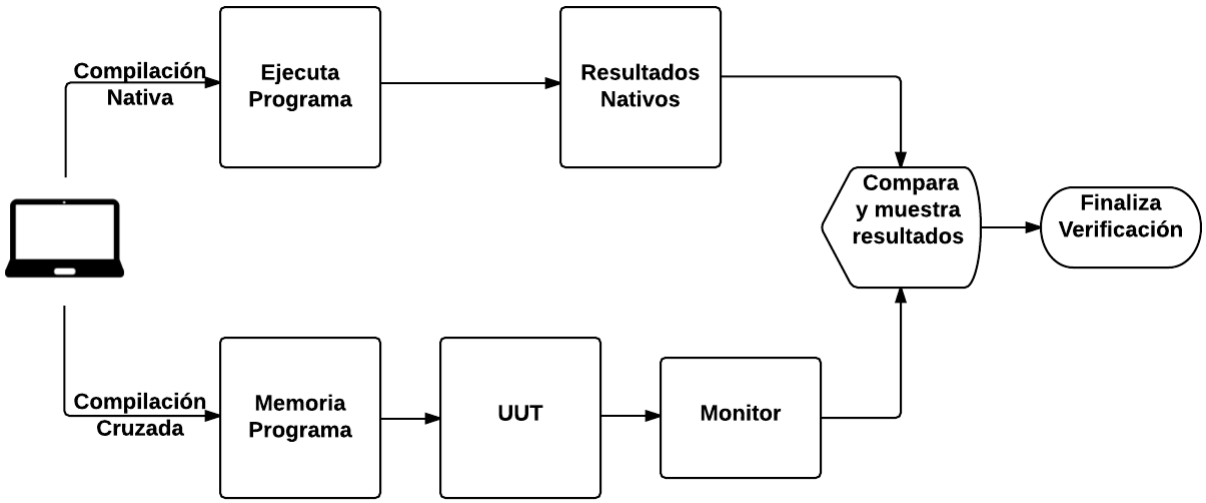

Figura 4.1: Ambiente de verificación utilizado para la verificación del ASP.

Un punto clave en la verificación funcional es el papel del monitor, ya que extrae la información necesaria para validar el dispositivo a prueba. Entre los resultados principales se tiene: observar el contenido interno de los registros, señales de entrada/salida de unidades

funcionales, tipo de instrucción que se está ejecutando, estados de la máquina de estados, tiempo de ejecución y número de ciclos transcurridos.

En la figura 4.2 se observa un ejemplo de los datos que el monitor ofrece para una instrucción *ADDI* y *SW* en un instante de tiempo particular.

```
New Instruction ADDI:81010113 . El PC es: 000080a4. El PC es: 00010148

State:  1.  RS1 ALU:           0. RS2 ALU:           0. ALU Out:           0.
State:  2.  RS1 ALU:           0. RS2 ALU:           0. ALU Out:           0.
State:  6.  RS1 ALU:           0. RS2 ALU:           0. ALU Out:           0.
State:  3.  RS1 ALU:           0. RS2 ALU:           0. ALU Out:           0.
State:  4.  RS1 ALU:       32768. RS2 ALU:       -2032. ALU Out:       30736.
State:  5.  RS1 ALU:       32768. RS2 ALU:       -2032. ALU Out:       30736.

New Instruction SW:7e112623 . El PC es: 000080a6. El PC es: 0001014c

State:  1.  RS1 ALU:       32768. RS2 ALU:       -2032. ALU Out:           0.
State:  2.  RS1 ALU:       32768. RS2 ALU:       -2032. ALU Out:           0.
State:  6.  RS1 ALU:       32768. RS2 ALU:       -2032. ALU Out:       30736.
State:  3.  RS1 ALU:       32768. RS2 ALU:       -2032. ALU Out:       30736.
State:  7.  RS1 ALU:       30736. RS2 ALU:        2028. ALU Out:       32764.
State: 11.  RS1 ALU:       30736. RS2 ALU:        2028. ALU Out:       32764.
State: 12.  RS1 ALU:       30736. RS2 ALU:        2028. ALU Out:           0.
State: 12.  RS1 ALU:       30736. RS2 ALU:        2028. ALU Out:           0.
```

Figura 4.2: Impresión de pantalla de parte de los resultados obtenidos por el monitor para una operación *ADDI* y *SW*. Nótese la posibilidad de monitorear el estado de los registros internos y el PC, y como la operación resulta decodificada.

4.2 Resultados del desarrollo del hardware

Se ofrece a continuación los resultados relacionados al tiempo de ejecución de las instrucciones implementadas dentro del microprocesador y la cantidad de recursos lógicos y eléctricos que requiere el sistema.

4.2.1 Tiempos de ejecución de las instrucciones implementadas

Se elaboró una aplicación en lenguaje ensamblador para estimular todas las instrucciones desarrolladas para el microprocesador. Al ser una máquina multiciclo, cada instrucción tiene tiempos de ejecución variables que pueden impactar la eficiencia del procesador para determinadas tareas. En la tabla 4.1 se muestran los resultados obtenidos en relación con la duración en ciclos de reloj de la ejecución de las instrucciones de la base entera RV32I y en la tabla 4.2 se muestra la duración de las instrucciones implementadas complementarias al conjunto de instrucciones base. Además se muestra el tiempo de ejecución de las mismas.

Tabla 4.1: Cantidad de ciclos de reloj y tiempo de ejecución de las instrucciones de la base entera RV32I implementadas en un microprocesador segmentado multiciclo a una frecuencia de operación de 100MHz.

Instrucción	**Número de ciclos de reloj**	**Tiempo de ejecución (ns)**
LUI	3	30
AUIPC	5	50
JAL	6	60
JALR	7	70
Branch	7	70
Load	14	140
Store	10	100
Aritmética/Lógica inmediatas	6	60
Aritmética/Lógica	5	50

Tabla 4.2: Cantidad de ciclos de reloj y tiempo de ejecución de las instrucciones con extensión M, F y D implementadas en un microprocesador segmentado multiciclo a una frecuencia de operación de 100MHz.

Instrucción	**Número de ciclos de reloj**	**Tiempo de ejecución (ns)**
MUL, MULH, MULHSU, MULHU	5	50
FLW y FLD	10	100
FSW y FSD	15	150
FMADD, FMSUB, FNMSUB y FNMADD	-	-
FADD, FSUB y FMUL	-	-
FSGNJ, FSGNJN y FSGNJX	5	50
FEQ, FLT y FLE	11	110
FCVT.S.D y FCVT.D.S	5	50

Tabla 4.3: Resumen de utilización de recursos Post Place & Route del microprocesador de aplicación específica sin unidad de coma flotante. Elaborado para la tarjeta de desarrollo Digilent Nexys 4. Xilinx ISE Design 14.04.

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers as Flip Flop	1703	126800	1%
Number of Slice LUTs	1586	63400	2%
Number of occupied Slices	704	15850	4%
Number with an unused Flip Flop	831	2476	33%
Number with an unused LUT	890	2476	35%
Number of fully used LUT-FF pairs	755	2476	30%
Number of RAMB36E1/FIFO36E1s	24	135	17%
Number of DSP48E1s	8	240	3%

En relación con la tabla 4.2, es posible observar que las instrucciones de la base entera RV32I que tardan más ciclos de reloj en su ejecución son: *Load* (14) y *Store* (10). La instrucción *Load* requiere más ciclos de ejecución que la *Store* pues necesita una lógica adicional para decodificar la lectura de la memoria, es decir, determinar si se requiere leer un byte, media palabra , una palabra o dos palabras. Es importante destacar que los tiempos de ejecución de las operaciones con código de operación: *FADD*, *FSUB*, *FMUL*, *FMADD*, *FMSUB*, *FNMSUB* y *FNMADD* son variables, pues el número de ciclos es función de la mantisa y el exponente particular de cada operando. (Para más información sobre la implementación y duración de las operaciones suma, resta, y multiplicación en coma flotante puede consultarse [54]).

4.2.2 Recursos utilizados

Para la implementación del microprocesador se desarrollaron dos tipos de arquitecturas: con y sin soporte de coma flotante. Un parámetro en el HDL habilita o deshabilitala la FPU. En las tablas 4.3 y 4.4 se muestra la utilización de recursos para ambos casos de arquitectura.

Puede verse como los recursos del sistema se triplican cuando se incorpora la FPU en la arquitectura interna del ASP.

Si se compara los recursos del clasificador con la etapa de identificación [26], se nota que el consumo de recursos de ambas son semejantes , por lo que existe un balance entre ambas etapas del SiRPA.

Tabla 4.4: Resumen de utilización de recursos Post Place & Route del microprocesador de aplicación específica con unidad de coma flotante. Elaborado para la tarjeta de desarrollo Digilent Nexys 4. Herramienta Xilinx ISE Design 14.04.

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers as Flip Flop	6174	126800	4%
Number of Slice LUTs	5505	63400	8%
Number of occupied Slices	2322	15850	14%
Number with an unused Flip Flop	2440	8137	29%
Number with an unused LUT	2632	8137	32%
Number of fully used LUT-FF pairs	3065	8137	29%
Number of RAMB36E1/FIFO36E1s	24	135	17%
Number of DSP48E1s	23	240	9%

Tabla 4.5: Cantidad de instrucciones y tamaño de la ROM en función del tipo de compilación de la aplicación.

Tipo de compilación	Número de instrucciones	Tamaño de bits ROM
Solo enteros	4578	15
Flotantes precisión simple	2052	14
Flotante precisión doble	2052	14

4.3 Resultados del desarrollo de software

Para la implementación del software se elaboraron tres bibliotecas: una para la implementación de la división, otra para el logaritmo natural y la tercera es la encargada de la ejecución del algoritmo de HMM. Todas estas fueron implementadas utilizando el lenguaje de alto nivel C. El software se compiló para datos enteros y para datos en coma flotante. En relación con los datos en coma flotante, se implementaron dos versiones: con precisión simple y doble. En la tabla 4.5 se exponen los resultados obtenidos.

Se observó que cuando se utilizaba una compilación para una arquitectura entera mediante la bandera *RV32I*, el resultado es un código con una cantidad de instrucciones del doble de las requeridas cuando se compila para flotante de coma simple y doble. El resultado anterior es de esperar, ya que al tener menos instrucciones disponibles el compilador requiere de un esfuerzo adicional al tratar de traducir el programa de alto a bajo nivel. Esto representa un impacto en la cantidad de instrucciones.

Cuando se compila la aplicación para una arquitectura con FPU y el programa tiene datos flotantes de diferente tipo (*float*, *double*), no existen cambios notables en el tamaño de la memoria de programa. Sin embargo, sí se presentan cambios en el tamaño de la

Tabla 4.6: Consumo de potencia del ASP con/sin FPU. Tomado de la herramienta XPower Analyzer de Xilinx para la tarjeta Digilent Nexys 4 Artix 7.

On-Chip	Potencia con FPU (mW)	Potencia sin FPU (mW)
Clocks	17	7
Logic	4	3
Signals	9	6
BRAMs	66	65
DSPs	3	3
IOs	10	10
Leakage	89	89
Total	198	184

memoria de datos, la razón de esto radica en función del tipo de variables que se utilice en el programa. Por ejemplo si las constantes que se utilizaron fueron datos de tipo *float* se logra una reducir el tamaño de la memoria de datos a la mitad en comparación a utilizar los datos para la misma aplicación con datos de tipo *double*.

Específicamente hablando sobre el ASP compilado para la aplicación del algoritmo de HMM, si se utilizaba constantes para datos de precisión simple, se requirió de una memoria de datos de 32kB, mientras que para el caso de precisión doble, fue necesario una memoria de 64kB.

4.4 Resultados del sistema integrado hardware/software

De la integración final del diseño, se obtuvieron los resultados de la tabla 4.6. Es posible observar los resultados de consumo de potencia para el ASP sin y con FPU.

Desde el punto de vista de potencia, se detectó que el mayor consumo de potencia sin contar el consumo estático, se originó debido a los bloque de la memoria RAM tanto para el caso del ASP con y sin FPU. El consumo estático en una FPGA es dominante sobre el dinámico así que era de esperar que este fuera alto.

La máxima frecuencia de operación a la que el microprocesador puede ejecutar para el caso del ASP con FPU es de 113.237 MHz, lo que implica un periodo mínimo de 8.831ns. Para el caso del ASP sin FPU se puede llevar la velocidad del sistema hasta 122.684 MHz para un periodo mínimo de 8.151ns. Estos datos son tomados de la herramienta Timing Analizer de Xilinx.

Al analizar los resultados obtenidos relacionados a la frecuencia máxima de operación se observó que ambos enfoques de solución son capaces de operar a 100MHz, lo que significa que dentro de la tarjeta Digilent Nexys 4, el sistema es capaz de operar correctamente sin la presencia de problemas de temporizado de señales. Para comprobar esto, se realizaron pruebas dentro de la tarjeta y además, se realizaron simulaciones Post Place & Route.

Tabla 4.7: Desviación estándar de los datos de la figuras 4.3, 4.4, 4.5, correspondientes a la comparación de resultados teóricos contra los experimentales de la aplicación de modelos ocultos de Markov para la detección de disparos, motosierras y bosques.

Indicador	**Bosque**	**Motosierra**	**Disparo**
STD	2.8896×10^{-5}	1.4336×10^{-5}	1.0271×10^{-5}

Con respecto al tiempo de ejecución del algoritmo, se compararon los resultados entre el ASP sin y con FPU. Para el primer caso, la cantidad de instrucciones necesarias para ejecutar el algoritmo creció exponencialmente en comparación al caso con FPU. Para ejecutar los tres algoritmos de HMM para detectar si el patrón pertenece a un bosque, un disparo o una motosierra, se requiere de un tiempo de ejecución de $80.43ms$.

Pese a que el ASP es funcional sin FPU, para ciertos casos los tiempos de ejecución violan los requerimientos de ventana disponible para procesar los datos. Por ejemplo, al utilizar un modelo con una cadena de 20 símbolos de longitud, con un muestreo igual a uno y diez estados ocultos, se posee una ventana de tiempo de $58.139ms$ para poder cumplir con el muestreo realizado en la etapa de preprocesamiento del SiRPA ($44.1kHz$).

Por otra parte, si se analiza el tiempo de ejecución del algoritmo con FPU, se observó que el tiempo de ejecución promedio fue de $23.01ms$, tomado a partir de una muestra de 20000 ejecuciones del algoritmo. Asimismo, en el estudio se encontró que el tiempo máximo de ejecución alcanzado fue de $31.57ms$. Es importante notar dos cosas: con FPU la solución cumple los requisitos temporales, ya que el algoritmo se ejecuta en un tiempo menor a $58ms$. La segunda es que en lugar de hablar de un tiempo de ejecución absoluto, se habla de un tiempo de ejecución promedio, pues la suma, resta y multiplicación en coma flotante requieren tiempos de ejecución variable en función de los operandos, por lo que no es posible definir un tiempo exacto en la ejecución.

Para la verificación funcional del sistema con FPU, se desarrolló un modelo de referencia del algoritmo de HMM en Octave. Se contrastaron los resultados del modelo de referencia contra el HDL del ASP. Las figuras 4.3, 4.4 y 4.5 resumen los resultados contra las respuesta del algoritmo de referencia en alto nivel.

El análisis anterior demostró que el algoritmo sobre el ASP se ejecutó con un porcentaje de error inferior a un 1% con respecto al modelo de referencia. La desviación estándar del error relativo es prácticamente cero (ver tabla 4.7).

Puede además notarse que cuando se tiene un resultado en el algoritmo de HMM inferior a -300, es porque la clasificación del elemento falló. Para determinar que el umbral límite era menor a -300 para esta aplicación, se realizó una evaluación del algoritmo utilizando los mismos datos del conjunto de entrenamiento y se concluyó que cuando el clasificador arroja datos menores a -300, la cadena de observación no pertenece a esa clase en particular.

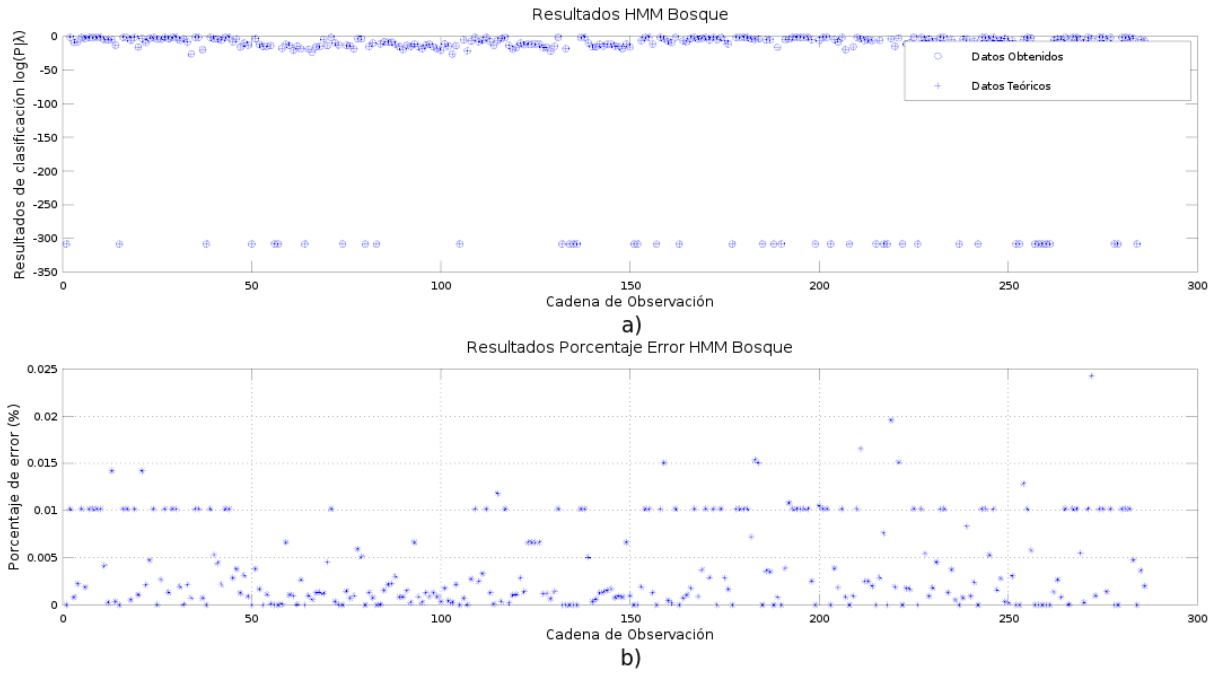

Figura 4.3: a) Resultados teóricos y experimentales del HMM para bosque. b) Porcentaje de error obtenido en la verificación funcional. Modelo teórico de referencia elaborado en Octave utilizando 5754 símbolos. Cuando la figura a) es menor a -300 significa que la cadena de observación evaluada no pertenece a la clase bosque.

Existe un caso en particular cuando dos (o los tres) clasificadores indican que una cadena de observación pertenece a diferentes clases de forma simultánea. Por ejemplo: si se analiza una cadena de observación y el algoritmo de HMM para el modelo bosque da como resultado -100.40, mientras que para el modelo de un disparo arroja un resultado igual a -4.98, ambos casos son mayores que -300, por lo que ambos clasificadores están indicando que el elemento pertenece a bosque y disparo al mismo tiempo.

Para solucionar este problema, se tomó el criterio extraído de la teoría de que conforme más cerca de cero se encuentre el resultado de una clasificación, existe mayor afinidad de que ese elemento pertenezca a esa clase. Por lo que en el caso anterior, el resultado de la clasificación será que la cadena de observación pertenece a un patrón de un disparo.

4.5 Resultados del clasificador

Como parte de la evaluación del clasificador, se eligieron cinco modelos diferentes para evaluar los resultados obtenidos y el tiempo de ejecución del ASP en función del modelo. Los cinco modelos que se analizaron poseen distintos estados ocultos, diferente cantidad de símbolos y diferente tasa de submuestreo y fueron obtenidos de la aplicación HMMSoft.

En la tabla 4.8 se muestran los principales datos de las matrices de confusión que surgieron como resultado de clasificar los cinco modelos generados, además se muestran los tiempos

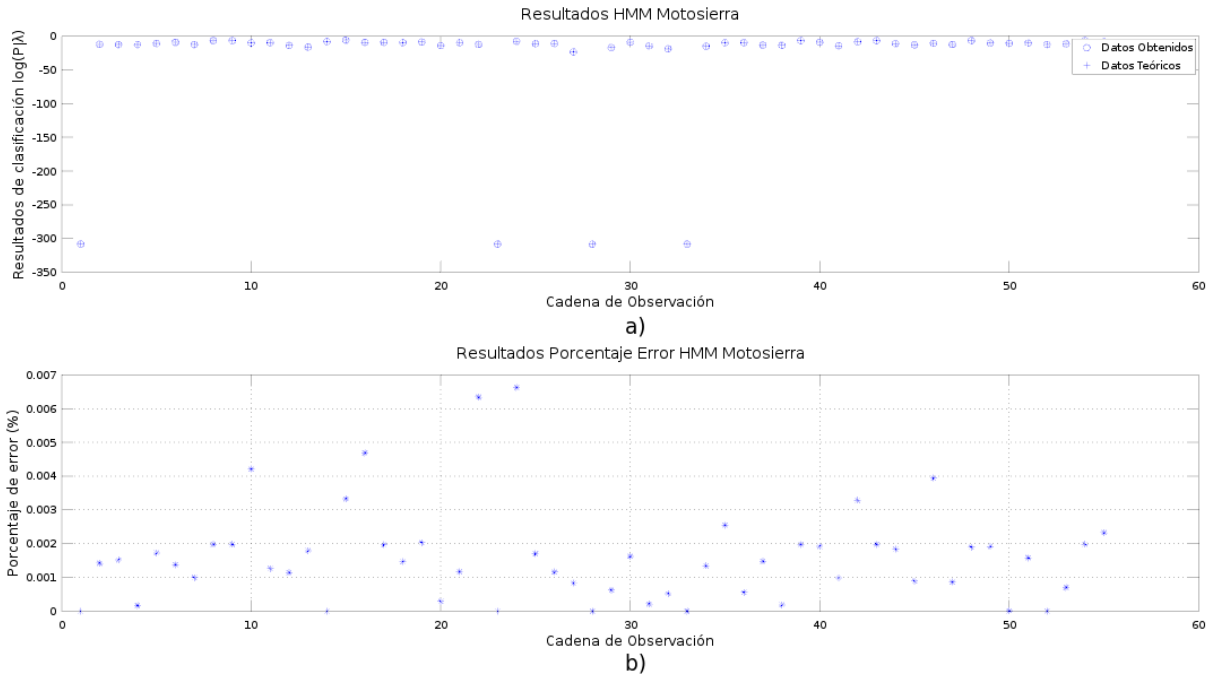

Figura 4.4: a) Resultados teóricos y experimentales del HMM para motosierra. b) Porcentaje de error obtenido en la verificación funcional. Modelo teórico de referencia elaborado en Octave utilizando 1106 símbolos. Cuando la figura a) es menor a -300 significa que la cadena de observación evaluada no pertenece a la clase motosierra.

Tabla 4.8: Tasa de reconocimiento obtenido por el clasificador dentro del ASP. Datos de reconocimiento tomados del HMMSoft y datos temporales tomados de la verificación funcional.

Características Modelo			Sensitividad (%)			VPP (%)			Tiempo ejecución (ms)	
Muestreo	Long. Cadena	Estado	Bosque	Motosierra	Disparo	Bosque	Motosierra	Disparo	Sin FPU	Con FPU
1	20	10	91.37	90.33	85.43	82.33	86	100	80.43	23.01
1	10	10	81.32	73.68	90.43	65.89	63.44	100	39.80	10.12
5	20	5	92.34	90.23	0.00	91.67	0	78.23	18.16	0.92
1	5	3	57.34	54.78	88.52	62.78	97.01	53.29	4.79	0.78
10	25	15	96.37	83.34	0	90.23	88.23	0	123.34	45.97

de ejecución del algoritmo implementado en el ASP con y sin FPU. Un punto importante en relación a los modelos es que se utilizaron las mismas cadenas de evaluación para la implementación del entrenamiento de cada uno de estos.

De la tabla 4.8 puede deducirse que el modelo que brinda los mejores resultados de los estudiados, fue el de un muestreo igual a uno, con una longitud de cadena igual a 20 y diez estados ocultos, el mismo que se utilizó en la verificación funcional. Sin embargo, aparecen detalles como por ejemplo que la clasificación de disparos da mejores resultados con una menor cantidad de símbolos, esto debido probablemente a la naturaleza impulsiva de un disparo. Mientras que en la clasificación de sonidos de motosierra y bosque, se obtienen mejores resultados cuando se incrementa la longitud de la cadena.

Otro resultado importante, es que existen modelos en la tabla 4.8, por ejemplo el de la fila cuatro y cinco, que pueden ser ejecutados en el ASP sin FPU, es decir, utilizando

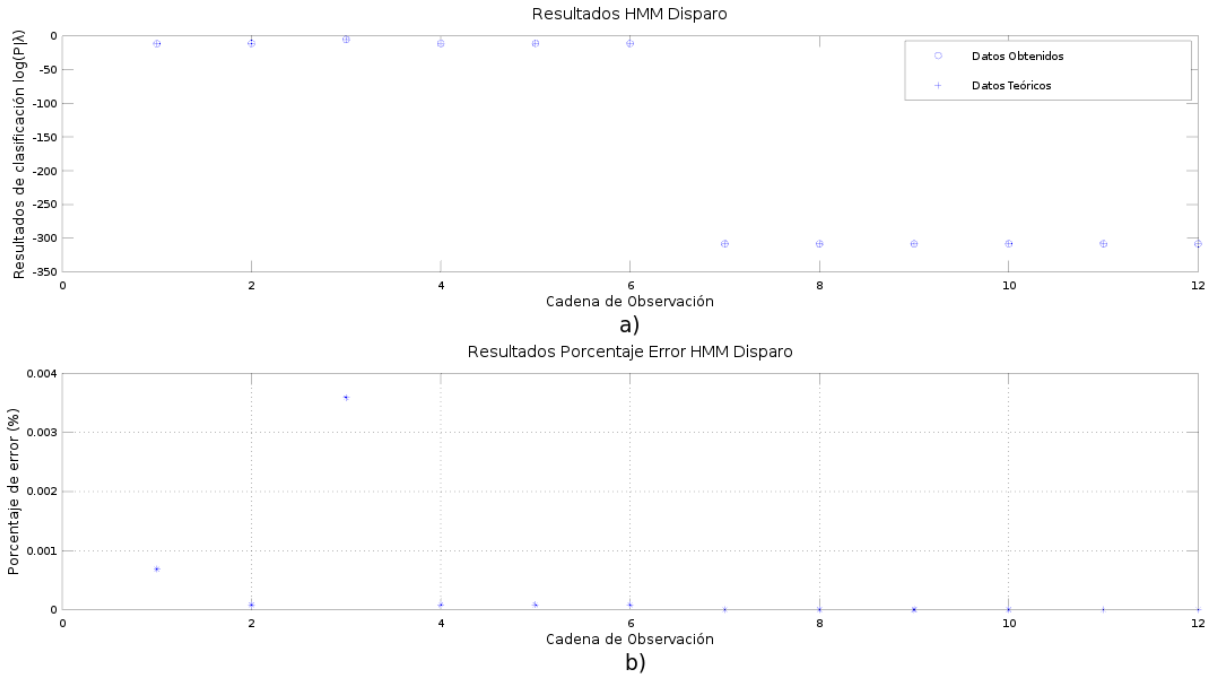

Figura 4.5: a) Resultados teóricos y experimentales del HMM para disparo. b) Porcentaje de error obtenido en la verificación funcional. Modelo teórico de referencia elaborado en Octave utilizando 258 símbolos. Cuando la figura a) es menor a -300 significa que la cadena de observación evaluada no pertenece a la clase disparo.

únicamente aritmética entera. Esto porque para estos casos no se violan las restricciones temporales. Lo anterior tiene como consecuencia una reducción de los recursos de acuerdo a lo estudiado en la tabla 4.3.

Capítulo 5

Conclusiones y recomendaciones

5.1 Conclusiones

Realizando un estudio de los principales conjuntos de instrucciones y de las diferentes arquitecturas que se utilizan actualmente, se determinó que la arquitectura del conjunto de instrucciones que ofrece las mayores ventajas para el ASP era RISC-V. Por otro lado, se concluyó que una arquitectura multiciclo segmentada, ofrece un consumo de potencia adecuada, permite operar a frecuencias relativamente altas y brinda un rendimiento aceptable para la aplicación de HMM.

También, se desarrolló un microprocesador de aplicación específica que ejecuta el algoritmo de modelos ocultos de Markov, para la detección de patrones acústicos de disparos y motosierras dentro de ambientes boscosos. El software se implementó en el lenguaje de alto nivel C y realizando una compilación cruzada de tipo *bare metal*, se logró obtener una solución funcional que resolviera adecuadamente el problema de clasificación del SiRPA.

Los mejores valores de sensitividad para las clases bosque, motosierra y disparo se obtuvieron cuando se realizó un entrenamiento múltiple para diez estados ocultos, muestreo igual a uno y una longitud de la cadena igual a 20. Los resultados obtenidos fueron de 89,47%, 90% y 86,67%.

El consumo de potencia dinámica del microprocesador fue de $99mW$ a una frecuencia de operación de $100MHz$. El tiempo de ejecución promedio del algoritmo dentro del ASP fue menor a $30ms$, esto utilizando una FPU y considerando el modelo estudiado durante la tesis.

Realizando una verificación funcional del ASP, se comprobó experimentalmente que el porcentaje de error de los datos teóricos contra los experimentales fue menor al 1%.

Dependiendo del tipo de modelo que se utilice para el algoritmo clasificador de HMM, una arquitectura sin FPU puede ser utilizada, contribuyendo esto en una reducción de los recurso en al menos tres ordenes de magnitud.

5.2 Recomendaciones

Se recomienda realizar la integración del sistema en un ASIC y con esto disminuir el consumo de potencia del ASP. Se propone además, la utilización de técnicas de ahorro de potencia dinámica como conmutación de alimentación (power gating), de manera que mientras no se detecte una señal con alta concentración energética, es decir, mientras no exista un posible disparo o motosierra, la unidad se encuentre apagada o en estado de espera (stand-by), y esta solo se active cuando realmente se detecte un posible estado de alarma.

Además, si se realizara una optimización en cuanto a la ejecución de las operaciones suma, resta y multiplicación de coma flotante, es posible obtener tiempos de ejecución menores, lo que permite pensar en disminuir la frecuencia de operación del ASP y de esta forma, lograr obtener una reducción en la potencia dinámica del sistema.

Finalmente, se propone emprender un estudio completo para determinar las características del modelo más adecuado de manera que se logre maximizar la tasa de reconocimiento de patrones acústicos de disparos, motosierra y bosque.

Bibliografía

[1] J. Montero, "Implementación de un sistema de reconocimiento de patrones acústicos de disparos y motosierras en un sistema embebido," Tesis de Licenciatura, Escuela de Ingeniería Electrónica, ITCR, Cartago, Costa Rica, Abril 2013.

[2] J. Hennessy and D. Patterson, *Computer Organization Design. The hardware/software interface*, 3rd ed. Morgan Kaufmann, 2005, vol. 1.

[3] S. Heath, *Microprocessor architectures RISC, CISC and DSP*, 2nd ed. Butterworth-Heinemann Ltd. Oxford, UK, UK, 1995.

[4] SINAC. Áreas Silvestres Protegidas. [Online]. Available: http://www.sinac.go.cr/competencias/ASP/Paginas/default.aspx

[5] INBio. (2015) Biodiversidad en Costa Rica. [Online]. Available: http://www.inbio.ac.cr/es/biod/bio_biodiver.htm

[6] ICT. (2014) INFORME ESTADÍSTICO TRIMESTRAL, CUARTO TRIMESTRE 2014 (IVT-2014). [Online]. Available: http://www.visitcostarica.com/ict/backoffice/treeDoc/files/Informe_Estadistico_Trimestral_IV_2014.pdf

[7] N. Instruments. (2009) ¿Qué es una Red de Sensores Inalámbricos? [Online]. Available: http://www.ni.com/white-paper/7142/es/

[8] C. M. Bishop, *Neural Networks for Pattern Recognition*. Clarendon Press Oxford, 1995.

[9] E. Salas. and P. Alvarado, "Implementación de un banco de filtros digitales multitasa para la estimación energética espectral en una aplicación de protección ambiental," in *XXX IEEE Convención de Centroamérica y Panamá*, Noviembre 2010.

[10] M. Sequeira and P. Alvarado, "Módulo de reducción de dimensiones espectrales en un sistema empotrado nodal de una red inalámbrica de sensores," in *Proceedings of the Embedded Technology Conference 2011*, San José, Costa Rica, 2011.

[11] J. Cárdenas, "Training Strategies for HMM in the acoustic pattern recognition," Tesis de Maestría, Escuela de Ingeniería en Computación, Instituto Tecnológico de Costa Rica, Mayo 2012.

[12] E. Salas and P. Alvarado, "Implementation of an automatic gain control for audio signals in an application for environmental protection," in *Proceedings of the Conference on Technologies for Sustainable Development TSD2011*, Cartago, Costa Rica, 2011.

[13] P. Cheung and W. Luk., "Hardware acceleration of hidden Markov model decoding for person detection," in *Design, Automation and Test in Europe. Proceedings*, 2005.

[14] K. Sujuan, Y. Hou, Z. Huang, and H. Li, "A HMM Speech Recognition System Based on FPGA," in *Image and Signal Processing, 2008. CISP '08. Congress on (Volume:5)*, Sanya, China, May 2008.

[15] V. Amudha, B. Venkataramani, R. Vinoth, and S. Ravishankar, "SOC Implementation of HMM Based Speaker Independent Isolated Digit Recognition System," in *VLSI Design. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, 2007.

[16] A. Mantri, M. Tiwari, and J. Singh, "Development of FPGA based Human Voice Recognition System with MFCC feature," in *International Journal of Engineering Trends and Technology (IJETT). Volume 8 Number 10*, Febrary 2014.

[17] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, "The RISC-V Instruction Set Manual. Volume I: User-Level ISA," 2014. [Online]. Available: http://riscv.org/download.html#tab_spec_user_isa

[18] B. University of California. (2015) Risc-V GNU Compiler Toolchain. [Online]. Available: http://riscv.org/download.html#tab_gnu-toolchain

[19] M. Saénz, "Reconocimiento de patrones acústicos para la protección del ambiente utilizando wavelets y Modelos Ocultos de Markov." Tesis de Licenciatura, Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica, Noviembre 2006.

[20] E. Salas, "Reconocimiento en tiempo real de patrones acústicos de motosierras y disparos por medio de una implementación en FPGA de modelos ocultos de Markov." Tesis de Licenciatura, Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica, Mayo 2010.

[21] M. Sequeira, "Módulo de reducción de dimensiones espectrales en un sistema de reconocimiento de patrones acústicos de motosierras y disparos por medio de una implementación en FPGA." Tesis de Licenciatura, Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica, Junio 2011.

[22] Beageboard. (2010) Beagleboard-xM Rev C System Reference Manual. [Online]. Available: http://beagleboard.org/static/BBxMSRM_latest.pdf

[23] L. Alfaro-Hidalgo, "Implementación en hardware del Sistema de Reconocimiento de Patrones Acústicos (SiRPA)." Tesis de Licenciatura, Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica, Junio 2013.

[24] L. Rabiner, "A tutorial on Hidden Markov Models and selected applications." in *IEEE*, 1990.

[25] M. Carvajal-Delgado, "Banco de Pruebas del Sistema de reconocimiento de patrones acústicos de motosierras y disparos." Tesis de Licenciatura, Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica, Noviembre 2013.

[26] C. Salazar-García, L. Alfaro-Hidalgo, M. Carvajal-Delgado, J. Montero, R. Castro, A. Chacón, and P. Alvarado, "Digital integrated circuit implementation of an identification stage for the detection of illegal hunting and logging," in *IEEE LASCAS*, Montevideo-Uruguay, Febrero 2015.

[27] F. A. Elmisery, A. H. Khalil, A. Salama, and H. F. Hammed, "A FPGA-Based HMM For A Discrete Arabic Speech Recognition System," in *ICM*, Cairo-Egipto, Diciembre 2003.

[28] K. Sujuan, Y. Hou, Z. Huang, and H. Li, "A HMM Speech Recognition System Based on FPGA," in *Congress on Image and Signal Processing*, China, Mayo 2008.

[29] Xilinx. (2009) Microblaze Processor Reference Guide. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/mb_ref_guide.pdf

[30] H. Geng, W. Liang, and M. Dong, "A Speech Recognition SoC Based on ARM7-TDMI Core and a MSAC Co-processor," in *SOC Conference, 2009. SOCC 2009. IEEE International*, China, Setiembre 2009.

[31] ARM. (2001) Arm7tdmi (Rev 3) Technical Reference Manual. [Online]. Available: http://www.atmel.com/images/ddi0029g_7tdmi_r3_trm.pdf

[32] I. Abbas, "Isolated Uttered Words Recognition Based on GMM/HMM Algorithms Using SoPC/Nios II Processor Build on Altera Cyclone II FPGA Chip," in *National Conference for Engineering Sciences*, Iraq, Noviembre 2012.

[33] Altera. (2015) Nios II Ethernet Standard Design Example. [Online]. Available: https://www.altera.com/support/support-resources/design-examples/intellectual-property/embedded/nios-ii/exm-net-std-de.html

[34] Y. R. Kumar, "Modified Viterbi Decoder for HMM Based Speech Recognition System," in *International Conference on Control, Instrumentation, Communication and Computational Technologies*, India, Julio 2014.

[35] K. Kpalma and J. Ronsin, "An Overview of Advances of Pattern Recognition Systems in Computer Vision," in *I-Tech Education and Publishing*, June 2007.

[36] Z. Ghahramani, "An introduce to hidden markov models and bayesian networks." in *International Journal of Pattern Recognition and Artificial Intelligence.*, 2001.

[37] Q. Wu, F. Merchant, and K. Castleman., *Microscope Image Processing.* Elsevier, 2008.

[38] Y. Qasim, P. Janga, S. Kumar, and H. Alesaimi. (S.F) APPLICATION SPECIFIC PROCESSORS. [Online]. Available: http://web.engr.oregonstate.edu/~qassimy/index_files/Final_ECE570_ASP_2012_Project_Report.pdf

[39] B. Gough, *An Introduction to GCC: for the GNU Compilers gcc and g++*. Network Theory Limited, 2004.

[40] K. Yaghmour, J. Masters, G. Yossef, and P. Gerum, *Building Embedded Linux System*, 2nd ed. O'Reilly Media, Inc., 2008.

[41] D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design: Modeling, Synthesis and Verification*. Springer Dordrecht Heidelberg London New York, 2010.

[42] Digilent. (2015) Nexys™4 Artix-7 FPGA Board. [Online]. Available: https://www.digilentinc.com/Products/Detail.cfm?Prod=NEXYS4

[43] X. A. Programable. (2015) Artix-7 FPGA Family. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/artix-7.html#documentation

[44] J. Hennessy and D. Patterson, *Computer architecture. A Quantitative Approach*, 5th ed. Morgan Kaufmann, 2012.

[45] S.A. (1998) MIPS Instruction Reference. [Online]. Available: http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

[46] Xataka. (2012) Imagination Technologies se queda con MIPS. [Online]. Available: http://www.xataka.com/otros/imagination-technologies-se-queda-con-mips

[47] I. Technologies. (2015) Lastest imagination news. [Online]. Available: http://imgtec.com/

[48] D. Lampret. (2014) OpenRISC Project Overview. [Online]. Available: http://openrisc.io/

[49] B. University of California. (2015) The RISC-V Instruction Set Architecture. [Online]. Available: http://riscv.org/

[50] Berkeley. (2015) Why not build on OpenRISC? [Online]. Available: http://https://blog.riscv.org/2014/10/why-not-build-on-openrisc/.org/

[51] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, 2nd ed. Prentice-Hall, 2002, vol. 1.

[52] N. Weste and D. Harris, *Principles of CMOS VLSI Design*, 3rd ed. Addison Wesley, 2004, vol. 1.

[53] A. T. A. for the Digital World. (2015) HARVARD VS VON NEUMANN. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka11516.html

[54] D. Rodríguez, "Diseño e implementación de una unidad aritmético-lógica de punto flotante para un procesador de aplicación específica," Tesis de Licenciatura, Escuela de Ingeniería Electrónica, ITCR, Cartago, Costa Rica, Noviembre 2013.

[55] A. Schliep, B. Steckemetz, B. Wichern, and B. Knab. (2015) GHMM library. [Online]. Available: http://ghmm.org/

[56] N. Louvet, J. Muller, and A. Panhaleux, "Newton-Raphson Algorithms for Floating-Point Division Using an FMA," in *21st IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP),*, Julio 2010.

[57] S. A. (2003) El algoritmo CORDIC. [Online]. Available: https://www.jcea.es/artic/cordic.htm

Apéndice A

Datos precargados en la memoria RAM

Para las instrucciones en coma flotante a diferencia de las instrucciones enteras, no existe una instrucción de carga inmediata, típicamente el sistema operativo se encarga de generar una excepción en la ejecución y generar esta constante. Sin embargo, debido que para el ASP desarrollado en este trabajo se compiló en bare metal fue necesario idear la forma de generar las constantes inmediatas necesarias para la ejecución del algoritmo. Se expone en la tablas A.1 y A.2 los datos precargados en la memoria RAM para la ejecución del algoritmo, para precisión simple y doble.

Tabla A.1: Datos precargados en la memoria RAM para inicializar el algoritmo, constantes en formato IEEE 754 con precisión simple.

Constante	Dirección	Grupo
0	0x0400	Constantes Base
1	0x0404	
-1	0x0408	
10000000	0x040C	Constantes División
1000000	0x0410	
100000	0x0414	
10000	0x0418	
1000	0x041C	
100	0x0420	
10	0x0424	
0.1	0x0428	
0.01	0x042C	
0.001	0x0430	
0.0001	0x0434	
0.00001	0x0438	
0.000001	0x043C	

0.0000001	0x0440	
0.5	0x0444	
2.823529	0x0448	
1.882353	0x044C	
2	0x0450	
e	0x0454	Constantes Logaritmo
0.367879	0x0458	
0.5	0x045C	
0.333333	0x0460	
0.25	0x0464	
$log(e)$	0x0468	
1.648721	0x046C	Constantes Cordic
1.284025	0x0470	
0.606530	0x0474	
0.778801	0x0478	
A[0][0]	0x047C	Matriz A Bosque
⋮	⋮	
A[9][9]	0x0608	
B[0][0]	0x060C	Matriz B Bosque
⋮	⋮	
B[9][31]	0x0B08	
pi[0]	0x0B0C	Matriz π Bosque
⋮	⋮	
pi[9]	0x0B30	
A[0][0]	0x0B34	Matriz A Motosierra
⋮	⋮	
A[9][9]	0x0CC0	
B[0][0]	0x0CC4	Matriz B Motosierra
⋮	⋮	
B[9][31]	0x11C0	
pi[0]	0x11C4	Matriz *pi* Motosierra
⋮	⋮	
pi[9]	0x11E8	
A[0][0]	0x11EC	Matriz A Disparo
⋮	⋮	
A[9][9]	0x1378	
B[0][0]	0x137C	Matriz B Disparo
⋮	⋮	
B[9][31]	0x1878	
pi[0]	0x187C	Matriz π Disparo

⋮	⋮	
pi[9]	0x18A0	
$-\infty$	0x18A4	Constante $-\infty$

Tabla A.2: Datos precargados en la memoria RAM para inicializar el algoritmo, constantes en formato IEEE 754 con precisión doble.

Constante	**Dirección**	**Grupo**
0	0x0400	Constantes Base
1	0x0408	
-1	0x0410	
10000000	0x0418	Constantes División
1000000	0x0420	
100000	0x0428	
10000	0x0430	
1000	0x0438	
100	0x0440	
10	0x0448	
0.1	0x0450	
0.01	0x0458	
0.001	0x0460	
0.0001	0x0468	
0.00001	0x0470	
0.000001	0x0478	
0.0000001	0x0480	
0.5	0x0488	
2.823529	0x0490	
1.882353	0x0498	
2	0x04A0	
e	0x04A8	Constantes Logaritmo
0.367879	0x04B0	
0.5	0x04B8	
0.333333	0x04C0	
0.25	0x04C8	
$log(e)$	0x04D0	
1.648721	0x04D8	Constantes Cordic
1.284025	0x04E0	
0.606530	0x04E8	
0.778801	0x04F0	
A[0][0]	0x04F8	Matriz A Bosque
⋮	⋮	
A[9][9]	0x0818	

B[0][0]	0x0820	Matriz B Bosque
⋮	⋮	
B[9][31]	0x1210	
pi[0]	0x1218	Matriz π Bosque
⋮	⋮	
pi[9]	0x1260	
A[0][0]	0x1268	Matriz A Motosierra
⋮	⋮	
A[9][9]	0x1580	
B[0][0]	0x1588	Matriz B Motosierra
⋮	⋮	
B[9][31]	0x1F80	
pi[0]	0x1F88	Matriz *pi* Motosierra
⋮	⋮	
pi[9]	0x1FD0	
A[0][0]	0x1FD8	Matriz A Disparo
⋮	⋮	
A[9][9]	0x22F0	
B[0][0]	0x22F8	Matriz B Disparo
⋮	⋮	
B[9][31]	0x2CF0	
pi[0]	0x2CF8	Matriz π Disparo
⋮	⋮	
pi[9]	0x2D40	
$-\infty$	0x2D48	Constante $-\infty$