PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO DE UN PROCESADOR CRIPTOGRÁFICO DE CURVAS ELÍPTICAS PARA EL DISPOSITIVO WISP

Tesis para obtener el título profesional de Ingeniero Electrónico

AUTOR

Igor Ivan Mendez Cabana

ASESOR:

Dr. Ing. Carlos Bernardino Silva Cárdenas

Lima, febrero, 2023

Informe de Similitud

Yo, ……CARLOS SILVA CARDENAS……,

docente de la Facultad de …CIENCIAS E INGENIERIA de la Pontificia Universidad Católica del Perú, asesor(a) de la tesis/el trabajo de investigación titulado:

DISEÑO DE UN PROCESADOR CRIPTOGRÁFICO DE CURVAS ELÍPTICAS PARA EL DISPOSITIVO WISP

del/de la autor(a)/ de los(as) autores(as) IGOR IVAN MENDEZ CABANA ,dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 35%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 13/02/2023. Es necesario indicar que el porcentaje es alto porque esta tesis toma información de lo trabajado en la tesis de bachiller del mismo alumno y de la cuál también he sido asesor; no hay forma de reducir el porcentaje pese a colocar la fuente real.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: …LIMA, 14 FEBREERO 2023…

Apellidos y nombres del asesor / de la asesora: Paterno Materno, Nombre1 Nombre 2 SILVA CARDENAS, CARLOS BERNARDINO	
DNI:08014721	Firma
ORCID: 0000-0003-4653-0915	

Resumen

El internet de las cosas (IoT) está creciendo a un ritmo acelerado y con ello las redes de sensores están tomando una mayor importancia. Los nuevos avances se enfocan en disminuir los costos, facilitar la implementación y la escalabilidad de estas redes. En este sentido, la tecnología RFID es una alternativa que brinda mejoras en estos aspectos. Esto se debe a que al no usar baterías para la implementación de los nodos permite que sean más baratos y brinda más capacidad de conectividad. La plataforma WISP (*Wireless Identification Sensing Platform*) es una etiqueta RFID programable que facilita el desarrollo de nodos RFID y que ha facilitado la investigación de nuevos protocolos de comunicación y de seguridad en RFID. Por otro lado, un problema que afecta la adopción de esta tecnología es el gran incremento de ciberataques a nodos IoT en los últimos años. Esto se debe principalmente a su baja seguridad ya que con sus limitaciones en recursos de hardware y energía se dificulta desarrollar criptografías en software óptimas.

En este trabajo se presenta la arquitectura de un procesador criptográfico de Curvas elípticas (ECC) de bajo consumo energético para un FPGA y que cumple con las limitaciones energéticas para ser utilizado con la etiqueta WISP. Además, el procesador propuesto soporta operaciones sobre GF(p) en curvas Weierstrass. Por otro lado, la operación de multiplicación modular se realiza utilizando el algoritmo *Multiple Word Radix-2 Montgomery Multiplication* (MWR2MM). De esta manera se puede implementar una arquitectura con forma de matriz sistólica lo que permite un alto nivel de paralelización y *pipelining*. Finalmente, se disminuyen las transiciones de señales y se eliminan los *glitches* que generan consumo energético innecesario.

Se realizó la simulación utilizando un campo de 192 bits en el FPGA igloo AGL1000V2. Como resultado se obtuvo una latencia de 4,157,358 ciclos de reloj. Además, a una frecuencia de 6MHz se obtuvo una potencia de 5.74 mW lo cual implica que, a medio metro de distancia de la antena, la etiqueta WISP necesitará 1.6 segundos para completar una operación de multiplicación de punto.

Palabras Clave: Bajo consumo energético, WISP, RFID, Criptografía de Curva Elíptica, FPGA

Dedicado a mis padres por impulsarme a enfrentar nuevos
retos y a superarme, a mi hermana por el apoyo
que me brinda día a día y a mis amigos
que alegran mi vida.

Índice General

Índice de figuras

Índice de tablas

Lista de abreviaturas

ASIC	Application-Specific Integrated Circuit
ASK	Amplitude-Shift Keying
ASM	Algorithmic State Machine
ASMD	Algorithmic State Machine with Datapath
DBLU	Doubling point Updating the input point
DP	Duplicación de Punto
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
EPC	Electronic Product Code
FIPS	Federal Information Processing Standards
FPGA	Field-Programmable Gate Array
IoT	Internet of Things
MCD	Máximo común Divisor
MM	Multiplicación Modular Montgomery
MP	Multiplicación de Punto
MWR2MM	Multiple Word Radix-2 Montgomery Multiplication
NIST	National Institute of Standards and Technology
PE	Process Element
RFID	Radio-frequency Identification
RTL	Register-Transfer Level
SP	Suma de Punto
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
UHF	Ultra High Frequency
WISP	Wireless Identification Sensing Platform
ZADDC	Conjugate co-Z addition
ZADDU	co-Z addition with update

Introducción

El internet de las cosas (IoT) es una gran red de dispositivos interconectados que permiten recopilar una gran cantidad de datos e interactuar entre ellos. Actualmente, es una de las tecnologías en mayor auge. Acorde con las últimas estadísticas presentada por Statista, se estima que la cantidad de dispositivos conectados a internet superará los 50 billones en el 2030 [1]. La velocidad e intensidad con la que esta tecnología avanza depende en gran medida de, en primer lugar, la eficiencia energética y, en segundo, la seguridad de las redes de sensores. Por ello, el uso de la tecnología RFID (*Radio Frequency Identification*) en estas redes de sensores es una propuesta cada vez más atractiva debido principalmente a que no requiere de baterías para funcionar. Por otro lado, gracias al avance en la microelectrónica, ahora es posible el uso de microcontroladores de propósito general de muy bajo consumo energético. Es a partir de esto que nace la etiqueta WISP (*Wireless Identification Sensing Platform*) la cual es la primera etiqueta RFID computacional, es decir, que incorpora un microcontrolador de propósito general que permite ejecutar diversos algoritmos disipando muy baja potencia [2].

Por otro lado, la seguridad en estas redes está tomando mayor importancia en los últimos años debido al alarmante aumento de ataques a redes IoT (*Internet of Things*). Esto se debe a que los dispositivos que conforman esta red presentan grandes limitaciones de hardware, dificultando de esta manera la implementación de criptografías fuertes. Además, al ser dispositivos que suelen estar en lugares de fácil acceso son más propensos a ataques de canal lateral [2].

En esta tesis se diseñará un procesador criptográfico de curvas elípticas orientado a disminuir la disipación de potencia para poder adecuarse a los requerimientos de una etiqueta WISP. Para ello se diseñará la arquitectura del procesador, luego se sintetizará y simulará en el FPGA igloo. Posteriormente, se optimizará el diseño a partir de un análisis de la síntesis y se realizará una verificación funcional del circuito. El desarrollo de este documento será de la siguiente manera. En el primer capítulo se muestra el marco problemático. En el segundo capítulo, se introducen los conceptos teóricos necesarios para comprender acerca de la etiqueta WISP y la Criptografía de Curvas Elípticas. En el capítulo tercero, se presenta en detalle el diseño de la arquitectura propuesta. En el capítulo cuatro, se realiza la verificación funcional y se hace un análisis de la síntesis del circuito. Finalmente, en el último capítulo, se muestran las conclusiones y recomendaciones.

Capítulo 1

Planteamiento del Problema

1.1. Contexto del problema

En los últimos años, ha surgido un gran interés en las redes de sensores debido al creciente auge del internet de las cosas. Como resultado, se han visto beneficiados sectores desde el campo industrial, que ha mejorado la medición de datos y control de procesos, hasta el de la salud donde promete mejorar la calidad de vida de las personas. Sin embargo, este continuo crecimiento se ve frenado por las limitaciones tecnológicas actuales. Uno de los problemas más cruciales es el consumo energético. Esto es debido a que el tiempo de vida de los dispositivos dentro de la red está limitado por la energía que puede entregar una batería [2]. Además, el uso de baterías incrementa el tamaño, peso y costo de cada sensor. Frente a este problema, las características de la tecnología RFID surge como una solución atractiva [3].

1.1.1. Redes de sensores RFID

La identificación por radiofrecuencia, o "RFID" por sus siglas en inglés, es una tecnología que se usa para identificar objetos de una manera eficiente y automatizada. Este sistema está compuesto de las "etiquetas", que son dispositivos que incorporan cada uno un número de identificación único, y los "lectores" que detectan e identifican a todas las "etiquetas" que se encuentran dentro de su rango de alcance. Lo novedoso de esta tecnología es que estas "etiquetas" no requieren de baterías para poder funcionar, sino que aprovechan la energía de las ondas de radiofrecuencia que emite el "lector". Por tanto, al no usar baterías, las etiquetas pueden llegar a ser muy baratas y pequeñas, e incluso se pueden fabricar en forma de "stickers" que se adhieren a los objetos que se quieren identificar. Utilizando este sistema, las empresas de retail, por ejemplo, pueden realizar el inventariado y seguimiento de sus productos de manera eficiente y rápida con solo establecer un "lector" en su almacén [2].

Debido a estas ventajas, han surgido varias propuestas que plantean aplicar la tecnología RFID en las redes de sensores [4]. En este nuevo modelo RFID, las etiquetas no solo responden al lector con su número de identificación, sino que agrega un campo de data adicional que puede ser usado para enviar los datos obtenidos por los sensores. Este nuevo esquema, permite usar etiquetas RFID como nodos dentro de las redes de sensores

incrementando así su tiempo de vida, llegando incluso a décadas. Además, su reducido peso y tamaño les da el potencial de poder conectar cualquier "cosa" a una red, reduciendo la brecha entre el mundo físico y el mundo virtual de las cosas [2]. Es por ello que muchos investigadores en Internet de las Cosas (IoT) consideran al RFID como los cimientos de la futura generación IoT [4].

1.1.2. Etiqueta WISP

WISP es la primera etiqueta RFID programable. Incorpora un microcontrolador de propósito general de muy bajo consumo energético y puertos de extensión que permiten agregar sensores externos u otros dispositivos de bajo consumo [5]. Este dispositivo fue desarrollado en los laboratorios de Intel como un proyecto de código abierto, lo cual incentivó que otros investigadores utilicen y modifiquen este dispositivo. La última versión es el WISP 5 (Figura 1.1) y tiene un alcance de hasta seis metros del lector [2].

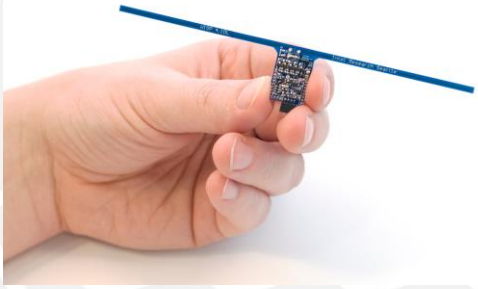

Figura 1.1. Etiqueta RFID UHF: WISP 5 [5]

En la última década, se ha creado una comunidad de desarrolladores que han empezado a utilizar este dispositivo en sus proyectos. Un ejemplo es WISPCam [6], que acopla una pequeña cámara a una etiqueta WISP para tomar una fotografía cuando detecta movimiento, permitiendo enviar imágenes cada 15 minutos. Otro ejemplo es NeuralWISP [7] que detecta pulsos neuronales de insectos aprovechando el bajo peso y tamaño de WISP. Por otro lado, este dispositivo ha facilitado la investigación dentro de la comunidad de ciberseguridad en RFID. Esto es debido a que WISP es completamente programable lo que permite probar y analizar distintos protocolos de seguridad y algoritmos criptográficos sin requerir un gran esfuerzo en desarrollo [8]. Debido a todo esto es que WISP ha fortalecido el desarrollo de la tecnología de redes de sensores a lo largo de estos años, creando nuevas aplicaciones innovadoras y mejorando la ciberseguridad de estas redes [2].

1.2. Descripción y Formulación del problema

Según las estadísticas proporcionadas por Symantec, corporación internacional en seguridad informática, los ataques a dispositivos IoT han aumentado un 600% entre 2016 y 2017 [9]. Este gran incremento se debe a la gran facilidad con la que estos dispositivos pueden ser vulnerados, lo que genera un agujero de seguridad que los atacantes aprovechan para penetrar una red y ganar acceso a información sensible. Es por eso por lo que en los últimos años la ciberseguridad se ha vuelto un factor relevante en el desarrollo de software y hardware de dispositivos IoT [2]. Por ejemplo, en el caso de aplicaciones médicas es necesario realizar una autenticación de los dispositivos que solicitan una conexión para evitar que terceros puedan leer o modificar la data, ya que podría llegar a tener consecuencias críticas [10].

Además, en los nodos de redes de sensores o en la etiqueta WISP se debe considerar el riesgo que implica que estos dispositivos sean de fácil acceso ya que están expuestos en ambientes públicos. Este hecho hace que sean más propensos a "ataques de canal lateral". Este tipo de ataques aprovechan el acceso al hardware para extraer información de la ejecución de criptosistemas y que luego es utilizada para obtener las llaves del sistema. La información extraída puede ser simplemente un trazo de disipación de potencia, emanaciones electromagnéticas, tiempos de respuesta, etc. [11]

1.2.1. Implementación de criptografía en WISP

Para evitar que atacantes puedan robar información, clonar o falsificar dispositivos se hace uso de funciones criptográficas y protocolos de seguridad. Sin embargo, debido a su alto costo computacional, requieren un gran coste energético lo que dificulta la implementación de estos algoritmos en dispositivos de bajos recursos como las redes de sensores. En el caso de WISP y otras etiquetas RFID las limitaciones son más extremas ya que no incorporan batería.

En [8], C. Pendl, M. Pelnar y M. Hutter diseñaron una librería criptográfica basado en la "Criptografía de Curvas Elípticas" (ECC) que se podía utilizar en la etiqueta WISP. Esta librería contiene funciones criptográficas que permiten implementar protocolos de seguridad utilizando una llave de 192 bits. Sin embargo, si bien el algoritmo se puede ejecutar con los pocos recursos que dispone WISP, la velocidad de operación es muy baja. Así, por ejemplo, si la etiqueta está a una distancia de 40 cm del lector, una operación criptográfica requiere

alrededor de 58 segundos para completarse y este tiempo es aún mayor cuando aumenta la distancia [8].

1.3. Importancia y Motivación

WISP ha suscitado gran relevancia debido a la facilidad y rapidez con la que permite prototipar dispositivos RFID. Esto ha impactado especialmente en grupos de investigación que utilizan prototipos para probar y testear nuevos protocolos de comunicación y seguridad en RFID [5]. Es por lo que es necesario brindar mejores opciones criptográficas que disminuyan el esfuerzo de desarrollo y que pueda ser usado por distintos desarrolladores en sus proyectos. Por esta razón, es necesario una solución criptográfica que, en primer lugar, pueda satisfacer todas las limitaciones de recursos que presenta WISP y, en segundo lugar, que pueda ejecutarse en pocos segundos para no interferir con la comunicación entre los nodos [2]. Todos estos objetivos se pueden lograr con una implementación en hardware, en un dispositivo FPGA, que permite un mejor rendimiento y a la vez la suficiente flexibilidad para poder ser modificado. Además, al estar implementado en RTL (*Register Transfer Level*) facilita su integración a un WISP ASIC.

1.4. Objetivos

Objetivo principal:

Diseñar en el FPGA Igloo nano un procesador criptográfico de curvas elípticas (ECC) de un consumo energético menor a 15 mW para que pueda ser utilizado con la etiqueta WISP.

Los objetivos específicos son los siguientes:

1. Diseñar una arquitectura para el algoritmo de Multiplicación de Punto enfocado en el bajo consumo de energía
2. Implementar la arquitectura diseñada en un FPGA utilizando Verilog bajo la técnica ASMD (Algorithmic State Machine with Datapath)
3. Realizar la verificación funcional del módulo criptográfico
4. Optimizar el consumo energético en el FPGA Igloo Nano

Capítulo 2

Marco teórico

En el presente capítulo se explicarán los conceptos básicos sobre la etiqueta WISP, la criptografía de curvas elípticas (ECC) y el consumo energético en un FPGA. En la primera sección se describe la estructura y funcionamiento de la etiqueta WISP. En la segunda se profundiza sobre la implementación de criptografía en dispositivos de bajos recursos y se detallan los fundamentos teóricos de la criptografía de curvas elípticas, así como de los algoritmos que lo componen. Por último, se analizan los distintos tipos de consumo energético en un FPGA y las estrategias para disminuirlas.

2.1. Etiqueta RFID UHF: WISP

La etiqueta WISP 5 (Figura 2.1) es una etiqueta RFID del tipo UHF pasiva, es decir, no usa batería y opera a una radiofrecuencia aproximada de 900 MHz. En [12] se le acuña el término etiqueta RFID computacional porque incorpora un microcontrolador de propósito general de muy baja energía de la familia MSP430. Trabaja a una frecuencia máxima de 6MHz a 3.3V y de 4MHz a 1.8V. Además, incorpora un acelerómetro 3D, un sensor de temperatura y dos leds, también brinda unos puertos de extensión que permiten conectar sensores externos [12].

Figura 2.1. Etiqueta RFID UHF: WISP 5 [12]

WISP fue diseñado para que sea compatible con los “lectores” comerciales de etiquetas UHF que suelen utilizar el protocolo *Electronic Product Code* (EPC) *Class 1 Generation 2* para la comunicación lector-etiqueta. Este estándar define la conexión y los protocolos en la capa física y de enlace [2].

2.1.1. Diagrama de bloques de WISP

La etiqueta WISP se compone básicamente de seis partes que se pueden apreciar en la Figura 2.2. En primer lugar, la antena recibe la señal de radiofrecuencia emitida por el lector y lo transforman a señales eléctricas. El siguiente bloque cumple dos funciones. Primero, se encarga de adaptar las impedancias de la antena para disminuir las pérdidas y, segundo, rectifica la señal recibida. A partir de esta señal, se obtiene la información enviada por el lector y también se obtiene la energía para alimentar a todo el circuito. En el bloque de comunicación, se demodula la señal extrayendo la envolvente de la señal ASK (*Amplitude Shift Keying*) recibida y luego es acondicionada para adquirir niveles lógicos que puedan ser interpretados por el microcontrolador. Los bloques de control de energía y microcontrolador (MSP430) se describirán en los siguientes subcapítulos. [2]

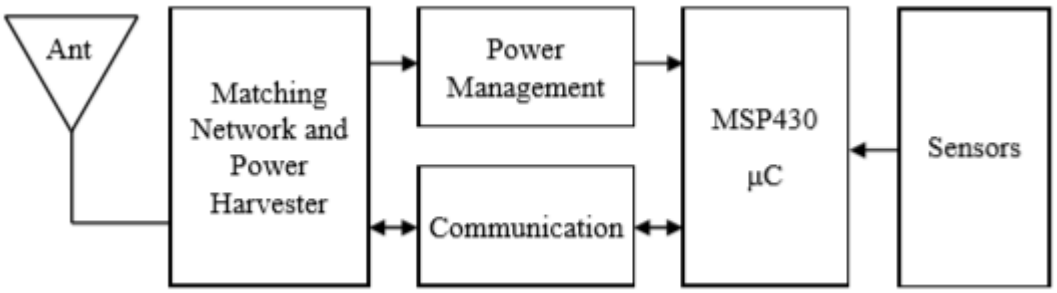

Figura 2.2. Diagrama de bloques de WISP [12]

2.1.2. Módulo control de energía

Para alimentar el circuito digital se necesita un voltaje regulado y estable. Sin embargo, debido a que la cantidad de potencia recibida por la "etiqueta" disminuye con el cuadrado de la distancia al "lector", cuando la etiqueta se aleja mucho, la potencia que recibe el regulador no es suficiente para mantener un voltaje estable a la salida y alimentar correctamente el circuito [2]. Es por este motivo, que se utiliza un condensador a la entrada del regulador que va almacenando energía hasta que el circuito supervisor ("SV" en la Figura 2.3.) detecte que alcanzó un valor adecuado para que pueda energizar el circuito y envía una interrupción al microcontrolador para habilitar el regulador. Una vez culminada la tarea, el microcontrolador entra en modo de bajo consumo y se mantiene a la espera de que el supervisor lo vuelva a activar para realizar otra tarea [2]. De esta forma, se pueden ejecutar algoritmos que requieren más energía del que se recibe del lector con la desventaja de que tardarán más tiempo en ejecutarse debido al tiempo que toma cargar el condensador. [5]

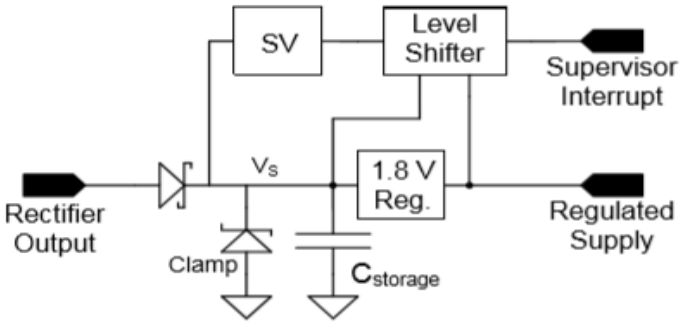

Figura 2.3. Circuito del módulo de control de energía [12]

En la Tabla 2.1 se muestran datos experimentales, extraídos de [5], donde se observa como varía la potencia recibida por la "etiqueta" a distintas distancias respecto al "lector" considerando una potencia de emisión de 1 W. Otro dato para considerar es que solo el protocolo de comunicación requiere 1.2mA a 1.8V, aproximadamente 2mW, por lo que sólo puede ejecutarse de manera continua hasta medio metro de distancia. Superado este límite, el microcontrolador ejecutará el algoritmo en periodos intermitentes de tiempo cuando el capacitor tenga suficiente energía, lo que aumentará el tiempo de ejecución del algoritmo. En general, entre mayor sea la distancia a la antena, el programa requerirá mayor tiempo para concluir [2].

Tabla 2.1. Disminución exponencial de la potencia respecto a la distancia

Distancia (m)	0.5	1	2	3	4	5
Potencia (uW)	2500	271	67.6	30.1	17.0	10.8

2.1.3. WISP firmware

El firmware está diseñado en tres capas, como se ve en la Figura 2.4. La capa más baja se encarga del control de energía para asegurarse de que el sistema no caerá en sub-voltajes. Para ello se comunica con el módulo de control de energía quien es el que determina en qué momentos el microcontrolador debe estar en estado activo o en bajo consumo. La segunda capa implementa el protocolo de comunicación "EPC class 1 gen 2" para poder recibir e interpretar los comandos enviados por el "lector". Por último, la capa de aplicación incorpora el programa desarrollado por el usuario que puede ser lectura de sensores o algoritmos simples. Este esquema permite que el desarrollador solo se enfoque en la capa de aplicación si tener que preocuparse por las capas inferiores [2].

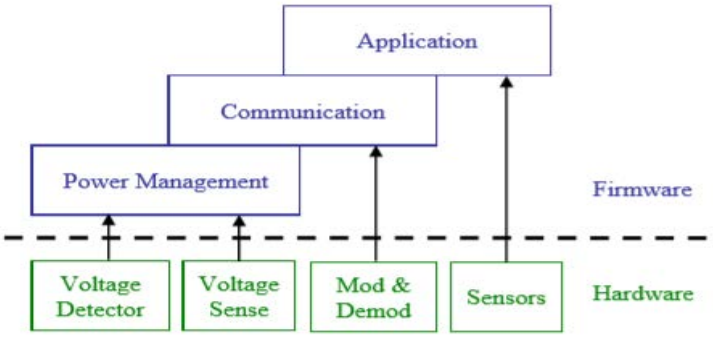

Figura 2.4. Diagrama de bloques de Firmware [12]

2.2. Estado del Arte en Criptografía

La implementación de funciones criptográficas en dispositivos de bajos recursos como el RFID es un gran reto ya que requiere un diseño que equilibre entre rendimiento, seguridad y costo [2]. A este tipo de criptografía se le conoce como "Criptografía ligera" y se enfoca en utilizar la mínima cantidad de recursos posibles [13]. Suelen implementarse en ASIC, ya que permite una mayor optimización de recursos lo que también implica un menor consumo de energía y área. Sin embargo, tiene como desventaja el no ser flexible, por lo que suelen ser mucho más complicado parchar las nuevas vulnerabilidades que suelen aparecer con el tiempo. Por otro lado, se encuentran las implementaciones en software que son muy flexibles ya que son reprogramables, pero no alcanza un nivel de optimización de recursos tal como una implementación en hardware. Por otra parte, el uso de FPGA es un punto medio entre optimización eficiente y flexibilidad que permite conseguir un bajo consumo energético con un menor tiempo de ejecución que una implementación en software [2].

Las funciones criptográficas se pueden dividir en tres grupos: algoritmo Hash, criptografía simétrica y criptografía asimétrica. Por un lado, los algoritmos Hash se utilizan conjuntamente con otros tipos de funciones criptográficas por lo que no son convenientes para dispositivos de bajos recursos [14]. Los algoritmos simétricos, por otro lado, son los más utilizados como criptografía ligera porque consumen menos recursos que su contraparte asimétrica. En el algoritmo simétrico cada integrante en la red comparte una misma llave que se usa para encriptar y desencriptar la información. Esto implica una vulnerabilidad para la red, ya que, al encontrar la llave de un nodo, todos los otros quedan expuestos. En cambio, en la criptografía asimétrica cada nodo tiene una llave privada que nunca es compartida y otra pública que es compartida al comenzar la comunicación. El principio de funcionamiento

consiste en que un paquete encriptado con la llave pública de un nodo sólo puede ser desencriptado con la llave privada del mismo nodo. De esta manera, aunque un nodo sea comprometido, se puede mantener la seguridad de la red. Por este motivo se están proponiendo cada vez más el uso de criptográficas asimétricas para implementar distintos protocolos para dispositivos de bajos recursos como RFID, redes de sensores o dispositivos IoT [13].

Respecto a la criptografía asimétrica, el algoritmo más usado es el RSA. Sin embargo, implementar este algoritmo requiere de muchos recursos de hardware. Desde hace unas décadas, el algoritmo ECC (*Elliptic Curve Cryptography*) ha llamado la atención, ya que proporciona una seguridad similar al RSA pero utilizando llaves de menor tamaño, lo cual implica un menor uso de recursos de memoria y hardware para implementarlo [15]. Es por ello por lo que actualmente se ha convertido en la mejor alternativa de criptografía asimétrica ligera, utilizándose en dispositivos de bajos recursos como IoT o incluso RFID [14].

2.3. Criptografía de curva elíptica

En el año 1980, Victor Miller y Neal Koblitz propusieron la criptografía de curva elíptica (ECC). Esta es una criptografía asimétrica o de llave pública que está basada en el problema del logaritmo discreto de las curvas elípticas sobre un campo finito (ECDLP). A partir de este algoritmo, se pueden conseguir varias aplicaciones de seguridad necesarias en una comunicación entre dos dispositivos como intercambio de llaves, firmas digitales, encriptación de datos y autenticación [15]. La mayor ventaja que ofrece ECC respecto a otras criptográficas asimétricas es que posee un tamaño de llave más pequeño y un mejor desempeño con un mismo nivel de seguridad. Esta característica lo vuelve un buen candidato para implementaciones en sistemas con bajos recursos [2].

2.3.1. Curva elíptica Sobre Campos finitos

Las curvas elípticas pueden ser definidas sobre cualquier tipo de campo, como el campo real, el complejo, etc. Para aplicaciones criptográficas, se definen sobre campos finitos que pueden ser de dos tipos: campos binarios (F_{2m}) y campos primos (F_p). Entre estos, los campos primos son más simples y por tanto más utilizados en los estándares de curvas elípticas [2].

2.3.2. Curvas elípticas sobre *F(p)*

Existen distintos tipos de curvas elípticas sobre campos primos que son utilizadas en criptografía, tales como la curva Montgomery, la curva Gallant-Lambert-Vanstone o la llamada Curve25519 que se ha introducido recientemente en los últimos años [15]. Sin embargo, la más utilizada y estudiada es la curva Weierstrass que define a la curva elíptica a partir de la ecuación corta de Weierstrass (*ecuación 2.1*) [2].

Si p es un número primo mayor a 3, una curva elíptica E sobre $F(p)$ es definida de la siguiente manera:

$$E: \quad y^2 = (x^3 + ax + b) \bmod p, \qquad (2.1)$$

donde $a,b,x,y \in F(p)$ y su discriminante $4a + 27b \neq 0$. Los parámetros a, b y p son importantes para determinar la seguridad de la curva. Es por ello que se han creado estándares que han sido validadas por organismos especializados en la seguridad informática [2]. Por ejemplo, el NIST, Instituto Nacional de Estándares y Tecnología de Estados Unidos, recomienda en FIPS 186-2 los estándares p-192, p-224, p-256, p-384 y p-521 [15]. Cabe recalcar que a un valor de campo 'p' mayor, la criptografía será más fuerte de romper, pero también requerirá un mayor costo computacional. Otro estándar de curvas elípticas populares son las curvas Brainpool [16].

2.2.3. Aritmética de Curva elíptica sobre F(p)

Se tienen dos puntos $P = (x_1, y_1)$ y $Q = (x_2, y_2)$ que pertenecen a la curva elíptica E de la *ecuación 2.1*. A partir de estos puntos se definen las siguientes operaciones [2]:

- **Inversa del punto P**

  Se define como el punto simétrico respecto al eje x.

  $$-P = (x_1, -y_1)$$

- **Suma de dos puntos**

  La suma de los puntos P y Q da como resultado la inversa del punto generado por la intersección entre la curva E y la línea que une P y Q.

  El nuevo punto $R = P + Q = (x_3, y_3)$ se puede calcular a partir de las siguientes fórmulas:

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1 \tag{2.2}$$

$$\lambda = \frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{2.3}$$

- **Duplicación de punto**

  La duplicación de *P* es la inversa del punto generado por la intersección entre la línea tangente a la curva *E* en el punto *P* y la curva *E*. El punto $R = 2P = (x_3, y_3)$ se puede calcular a partir de las siguientes ecuaciones:

$$x_3 = \lambda^2 - 2x_1, \quad y_3 = \lambda(x_1 - x_3) - y_1 \tag{2.4}$$

$$\lambda = \frac{(3x_1^2 + a)}{(2y_1)} \tag{2.5}$$

- **Multiplicación de punto o multiplicación escalar**

  La multiplicación $Q = kP$ entre un punto de la curva *P* y un escalar *k* se define como la suma del punto *P* consigo mismo *k* veces. Esta operación es la base de la Criptografía de Curvas Elípticas. Por ello, los procesadores criptográficos se diseñan para acelerar esta operación.

2.3.4. Algoritmo de multiplicación de punto

Para realizar las operaciones aritméticas de curva, las coordenadas de los puntos se pueden representar de dos formas: *affine* o *projective* [2]. El primero es la manera común de representar una coordenada, es decir, con un valor binario *(X,Y)*. Mientras que la segunda forma lo representa con un valor ternario *(X,Y,Z)* donde $Z \neq 0$ y tiene una correspondencia con las coordenadas affine $(\frac{X}{Z^c}, \frac{Y}{Z^d})$. Los métodos *projective* más comunes son el estándar (*c=1, d=1*), Jacobeans (*c=2, d=3*) y Lopez-Dahab (*c=1, d=2*) [15].

La mayoría de los algoritmos de multiplicación de punto hacen uso de la operación duplicación de punto (DP) para reducir la cantidad de sumas de punto (SP). Una opción para disminuir el uso de área en implementaciones para dispositivos de bajos recursos consiste en operar de manera iterativa con cada bit de *k* (que es el operador escalar en $Q = kP$). El algoritmo más utilizado debido a su simplicidad y eficiencia es el *Double-and-Add* (algoritmo 2.1). En este solo realiza una operación de SP y DP en cada iteración de cada bit de *k*.

Algoritmo 2.1. Cálculo de la Multiplicación de punto usando el algoritmo *Doube-and-Add* [15]

Input: Prime $p \in E(\mathbb{F}_q)$, $P = (x, y)$, where $x, y \in GF(p)$
$k \in \mathbb{Z}, 0 < k < \#E$, $k = (k_{l-1}, k_{l-2}, \ldots, k_0)_2$, $k_{l-1} = 1$
Output: $Q = (x', y')$
$Q = P$
for $i = l - 2$ **downto** 0 **do**
$Q = 2Q$
if $k_i = 1$ **then**
$Q = Q + P$
return Q

Sin embargo, este algoritmo es vulnerable a un tipo de ataque de canal lateral conocido como análisis simple de consumo energía. Con este ataque se puede predecir el valor de *k* analizando las variaciones en la potencia disipada por el dispositivo en cada iteración, ya que la potencia al realizar la operación SP será diferente que al realizar DP. De esta manera se puede inferir el valor de cada bit de *k* [2].

El algoritmo *Montgomery Ladder* (algoritmo 2.2) es una alternativa similar al *Double-and-Add* pero que es fuerte contra este tipo de ataque. Esto se debe a que en cada iteración siempre se realiza ambas operaciones SP y DP consiguiendo así un trazo de potencia regular y uniforme.

Algoritmo 2.2. Cálculo de la Multiplicación de punto usando el algoritmo *Montgomery Ladder* [15]

Input: Prime $p \in E(\mathbb{F}_q)$, $P = (x, y)$, where $x, y \in GF(p)$
$k \in \mathbb{Z}, 0 < k < \#E$, $k = (k_{l-1}, k_{l-2}, \ldots, k_0)_2$, $k_{l-1} = 1$
Output: $Q = (x', y')$
$Q = P$
$P = 2Q$
for $i = l - 2$ **downto** 0 **do**
if $k_i = 1$ **then**
$Q = Q + P$
$P = 2P$
else
$P = P + Q$
$Q = 2Q$
return Q

2.3.5. Algoritmo de suma y duplicación de punto

En las ecuaciones (2.2) y (2.3) presentadas anteriormente se define la suma de punto para coordenadas *affine*, y en las (2.4) y (2.5) se define la duplicación de punto. Estas ecuaciones

se pueden modificar para utilizar coordenadas *projective*. Según el trabajo realizado por Melodi [17], se demuestra que la suma de dos puntos y la duplicación de punto pueden ser aceleradas utilizando coordenadas *projective* que comparten el mismo valor de Z. Así introduce una nueva fórmula, ZADDU, a la que llama Suma de punto con actualización (Algoritmo 2.3). Esta operación calcula la suma de puntos utilizando menos operaciones y tiene la ventaja de que sin un costo extra también actualiza el valor de la coordenada Z de los puntos para que se mantengan iguales, una condición para continuar utilizando este algoritmo. También introduce la fórmula ZADDC, Suma de punto con conjugada (Algoritmo 2.4). Esta operación calcula la suma de puntos y sin un costo extra también calcula la diferencia de puntos manteniendo igual el valor de la coordenada Z. Como resultado, el algoritmo *Montgomery Ladder* puede optimizarse (algoritmo 2.5) si se remplazan las operaciones de suma y duplicación de punto por las fórmulas ZADDU y ZADDC.

Algoritmo 2.3. ZADDU: Algoritmo de suma de punto con actualización [17]

Require: $\boldsymbol{P} = (X_1 : Y_1 : Z)$ and $\boldsymbol{Q} = (X_2 : Y_2 : Z)$
Ensure: $(\boldsymbol{R}, \boldsymbol{P}) \leftarrow \text{ZADDU}(\boldsymbol{P}, \boldsymbol{Q})$ where $\boldsymbol{R} \leftarrow \boldsymbol{P}+\boldsymbol{Q} = (X_3 : Y_3 : Z_3)$ and $\boldsymbol{P} \leftarrow (\lambda^2 X_1 : \lambda^3 Y_1 : Z_3)$ with $Z_3 = \lambda Z_1$ for some $\lambda \neq 0$

function ZADDU($\boldsymbol{P}, \boldsymbol{Q}$)
$C \leftarrow (X_1 - X_2)^2$
$W_1 \leftarrow X_1 C$; $W_2 \leftarrow X_2 C$
$D \leftarrow (Y_1 - Y_2)^2$; $A_1 \leftarrow Y_1(W_1 - W_2)$
$X_3 \leftarrow D - W_1 - W_2$; $Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$; $Z_3 \leftarrow Z(X_1 - X_2)$
$X_1 \leftarrow W_1$; $Y_1 \leftarrow A_1$; $Z_1 \leftarrow Z_3$
end function

Algoritmo 2.4. ZADDC: Algoritmo de suma de punto con conjugada [17]

Require: $\boldsymbol{P} = (X_1 : Y_1 : Z)$ and $\boldsymbol{Q} = (X_2 : Y_2 : Z)$
Ensure: $(\boldsymbol{R}, \boldsymbol{S}) \leftarrow \text{ZADDC}(\boldsymbol{P}, \boldsymbol{Q})$ where $\boldsymbol{R} \leftarrow \boldsymbol{P}+\boldsymbol{Q} = (X_3 : Y_3 : Z_3)$ and $\boldsymbol{S} \leftarrow \boldsymbol{P}-\boldsymbol{Q} = (\overline{X_3} : \overline{Y_3} : Z_3)$

function ZADDC($\boldsymbol{P}, \boldsymbol{Q}$)
$C \leftarrow (X_1 - X_2)^2$
$W_1 \leftarrow X_1 C$; $W_2 \leftarrow X_2 C$
$D \leftarrow (Y_1 - Y_2)^2$; $A_1 \leftarrow Y_1(W_1 - W_2)$
$X_3 \leftarrow D - W_1 - W_2$; $Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$; $Z_3 \leftarrow Z(X_1 - X_2)$
$\overline{D} \leftarrow (Y_1 + Y_2)^2$
$\overline{X_3} \leftarrow \overline{D} - W_1 - W_2$; $\overline{Y_3} \leftarrow (Y_1 + Y_2)(W_1 - \overline{X_3}) - A_1$
end function

Algoritmo 2.5. Algoritmo *Montgomery Ladder* modificado con las fórmulas ZADDU y ZADDC [17]

Input: Prime $p \in E(\mathbb{F}_q)$, $P = (X, Y, Z)$,
where $X, Y, Z \in GF(p)$, $k \in \mathbb{Z}, 0 < k < \#E$,
$k = (k_{l-1}, k_{l-2}, \ldots, k_0)_2$, $k_{l-1} = 1$
Output: $Q = (X', Y', Z)$
$P, Q = DBLU(P)$
for $i = l - 2$ **downto** 0 **do**
  if $k_i = 1$ **then**
    $Q, P = ZADDC(P, Q)$
    $P, Q = ZADDU(Q, P)$
  else
    $P, Q = ZADDC(Q, P)$
    $Q, P = ZADDU(P, Q)$
return Q

2.4. Análisis del consumo energético en un FPGA

En los circuitos electrónicos se suelen analizar dos tipos de consumo energético: la potencia estática y la potencia dinámica. Sin embargo, en los dispositivos FPGA volátiles se deben tener en cuenta además otros dos tipos de consumo: la potencia de arranque y la potencia de configuración [2].

2.4.1. Potencia arranque y configuración

Los FPGA volátiles se componen de bloques que están basados en celdas SRAM, las cuales no mantienen su estado después de ser apagados. Por eso, requieren una memoria no volátil externa que almacene la configuración del diseño y que cargue al FPGA en cada arranque. Esto genera un pico de corriente durante algunos milisegundos debido a la potencia disipada al cargar la configuración de la memoria y al realizar la secuencia de iniciado en el FPGA.

Por otra parte, los FPGA no volátiles están basados en tecnología Flash por lo que tienen una significativa ventaja, ya que la configuración no necesita ser cargado de un dispositivo externo ya que esta funciona como una memoria no volátil. Así se evita los altos picos de corriente al arrancar. Además, debido a la tecnología FLASH, la secuencia de inicio disipa menos potencia que las celdas SRAM.

2.4.2. Potencia estática

La potencia estática hace referencia a la que se disipa cuando el FPGA está energizado, pero no está realizando ninguna operación. Esta pérdida se debe a la corriente de polarización y es resultado de las fugas de corriente en los transistores. Además, está fuertemente influenciado por la tecnología del dispositivo, como la tecnología del transistor o el proceso de fabricación. Los factores que se pueden controlar para disminuir la disipación de potencia estática son el valor del voltaje de alimentación y la temperatura del ambiente. Los FPGA basados en SRAM están compuestos de celdas de SRAM que se componen de seis transistores por celda que resulta en una sustancial fuga de corriente. En comparación, los FPGA basados en Flash consisten en solo un transistor por celda con hasta mil veces menor corriente de fuga. Por ello, este tipo de dispositivo se vuelve perfecto para aplicaciones de bajo consumo energético [2].

2.4.3. Potencia dinámica

La potencia dinámica se debe principalmente, a que durante las transiciones de estado de los transistores se genera un corto circuito por un corto lapso de tiempo que produce picos de corriente. Esta perdida se puede modelar utilizando la fórmula $P = CV^2F$, donde P es la potencia disipada, V es el voltaje de operación, F la frecuencia equivalente y C la capacitancia de la carga. Entre los últimos tres factores, el voltaje de operación y la capacitancia dependen del dispositivo utilizado, mientras que la frecuencia equivalente depende completamente del diseño. Es por esto que las técnicas para reducir el consumo dinámico se enfocan en reducir la frecuencia equivalente. El propósito principal es reducir las transiciones de estados del transistor [2].

a) Reducción de *Glitches*

Los *glitches* son transiciones de señal indeseadas que ocurren dentro de un bloque combinacional antes que la señal de salida se establezca. Esto ocurre porque cada nodo tiene un retardo distinto lo que resulta en varios cambios de estado antes de que la salida se termine de establecer. Un ejemplo de una fuente de *glitches* es un selector de un multiplexor que cambia varias veces durante un periodo de reloj o nodos con retardos desbalanceados dentro del bloque combinacional. Estos *glitches* se pueden propagar por otros bloques combinacionales y afectar drásticamente la disipación de potencia. Algunas técnicas para reducir los *glitches* se muestra a continuación.

- **Rearreglar la lógica**:

  Esta técnica consiste en mover la lógica para que las posibles fuentes de *glitches* o las señales que tengan cambios rápidos durante un periodo de reloj se encuentren en la parte más baja del flujo de datos. De esta manera se reduce la propagación de *glitches*.

- **Implementar pipeline**:

  Esta técnica consiste en introducir registros a lo largo de la lógica combinacional. Esto aumenta la latencia del bloque, pero aumenta la velocidad y reduce la propagación de *glitches*. Si bien introducir registros extras causa un incremento en la potencia, los *glitches* se reducen drásticamente.

b) *Gating clock*

La señal de reloj es una gran fuente de disipación de potencia debido a la alta frecuencia de transición y al gran *fanout* que tiene. La técnica de *Gating clock* consiste en deshabilitar la señal de reloj de los módulos que no están siendo utilizados. De esta manera la frecuencia en ese módulo se vuelve cero y por ello el consumo dinámico también. La señal de reloj se suele desactivar utilizando una compuerta AND en la entrada de la señal de reloj del bloque. Sin embargo, es necesario que esta compuerta genere un retraso muy bajo en la señal porque puede ocasionar problemas de sincronización con otros bloques (*Clock skew*) y es por ello que los FPGA suelen disponer de compuertas especiales que minimizan el retraso de la señal y pueden ser utilizados para deshabilitar el reloj [2].

2.5. FPGA Igloo Nano

La familia de FPGA IGLOO de Microsemi [18] ofrece los dispositivos de menor consumo energético en el mercado. Estos FPGA están basados en tecnología FLASH (no volátil) y operan con voltajes entre 1.2 y 1.5 V. La principal ventaja de este dispositivo es que permite el uso de modos energía con los que se puede reducir el consumo de energía a costa de deshabilitar ciertas funciones y módulos. En el modo *ultra-low-power*, que es el de menor consumo energético, alcanza una disipación de potencia de hasta 5 uW mientras mantiene los datos de los registros y la memoria SRAM. Este modo se activa a través de un pin especial que puede ser controlado por un dispositivo externo [2].

Capítulo 3

Diseño del procesador criptográfico

En este capítulo se presentará la arquitectura del procesador criptográfico diseñado para el dispositivo WISP. En este trabajo se usarán técnicas de optimización de diseño como ASM (máquina de estados algorítmicas) y de reducción de *glitches* para disminuir la disipación de potencia dinámica. Para la multiplicación modular se utilizará el algoritmo *Multiple-Word Radix-2 Montgomery* que permite una implementación en arreglo sistólico, alcanzando así una mayor paralelización.

3.1. Esquema general del diseño

La arquitectura del diseño general se muestra en la figura 3.1. Este esquema se puede dividir en cuatro partes: Bloque de control, Datapath, Controlador de Bus y Memoria RAM.

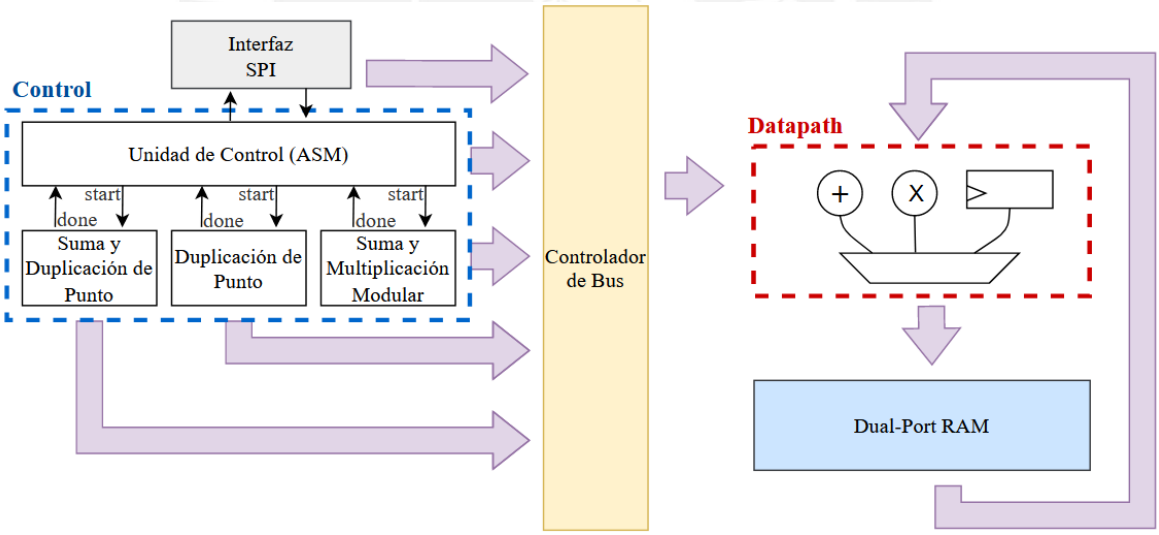

Figura 3.1. Diagrama general del procesador ECC

Por un lado, se tiene el "bloque de control" que ejecuta el algoritmo de multiplicación de punto en la unidad de control principal. En este mismo bloque se incluyen pequeñas unidades de control con las operaciones más utilizadas las cuales son la "Suma y Duplicación de punto", la "Duplicación de Punto", y la "Suma y Multiplicación Modular". Por otro lado, se tiene el "bloque *datapath"* que es donde se realizan las operaciones aritméticas modulares que incluye un sumador/restador modular, un multiplicador modular y también un registro de

desplazamiento. Asimismo, se introduce un "Controlador de Bus" que permite compartir el uso del *datapath* entre las distintas unidades de control y la interfaz SPI. Por último, se integra una memoria RAM Dual-Port para almacenar las variables utilizadas durante las operaciones. En la *Figura 3.1* se muestra la comunicación con un bloque SPI externo, pero no pertenece a la arquitectura del procesador criptográfico propuesto.

3.2. Mapeo de Registros

Las variables utilizadas entre operaciones se almacenan en la memoria RAM. En total se requieren 11 registros de *n* bits de ancho, donde *n* es el número de bits de la llave. Los tres primeros registros almacenan los operandos de la multiplicación de punto: el valor escalar *k* y el punto *(x, y)*. Los siguientes tres registros almacenan los valores de un segundo punto temporal $Q(X_2,Y_2,Z)$ que es necesario para algunas operaciones. Además, se agregan cinco registros temporales que serán utilizados para diversas operaciones. En la Tabla 3.1 se muestra el mapeo en memoria de cada registro.

Tabla 3.1. Mapeo de direcciones de registros

Dirección	Valor	Dirección	Valor	Dirección	Valor
0	K	4	Y_2	8	T3
1	X	5	Z	9	T4
2	Y	6	T1	10	T5
3	X_2	7	T2		

3.3. Diseño de la unidad de control

El algoritmo ejecutado en la unidad de control principal puede ser dividido en tres etapas:

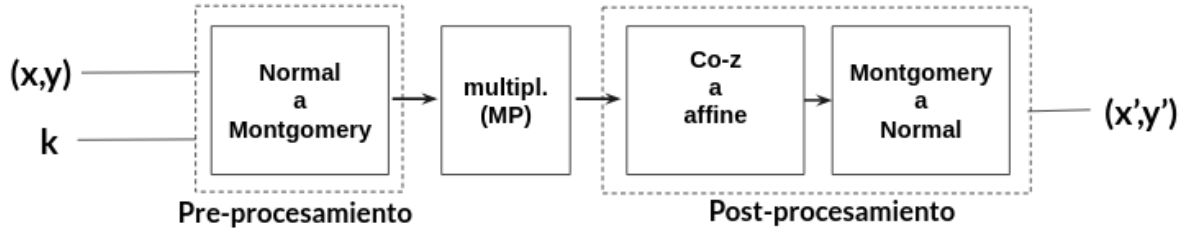

Figura 3.2. Secuencia de algoritmos en la unidad de control principal

La primera etapa se encargan del pre-procesamiento de los datos recibidos del microcontrolador *(x,y)* y *k*. Esto se realiza en dos pasos:

1. Se pasa el punto *P(x,y)* y *k* al dominio Montgomery: Normal a Montgomery
2. Se convierte el punto *P* de coordenadas *affine* a "coordenadas Z" (Co-Z)

   $(x,y) \rightarrow (X,Y,Z)$. Donde $X=x$, $Y=y$, $Z=1$

   Como se observa, la transformación de *affine* a Co-Z no requiere ninguna operación ya que *x* e *y* no cambian. Solo se debe considerar un valor de *Z* inicializado como *Z=1*.

En la segunda etapa, ya se disponen de los valores en las coordenadas y dominio adecuados, por lo que se puede realizar la multiplicación de punto. Luego, en la tercera etapa, el resultado de la multiplicación debe ser devuelto a coordenadas *affine* y al dominio normal.

1. Se vuelve el punto *P* de "coordenadas Z" a coordenadas *affine*
2. Se pasa el punto *P(x,y)* al dominio Normal: Montgomery a Normal

3.3.1. Primera Etapa: Normal a Montgomery

Para hacer uso de la "multiplicación modular Montgomery" es necesario que todos los números estén en el dominio Montgomery. Es por ello que se requiere que las coordenadas sean convertidas de normal *(x, y)* a Montgomery *(x', y')* con las siguientes ecuaciones.

$$x' = xR \bmod M = xR^2R^{-1} \bmod M = MM(x, R^2)$$

$$y' = yR \bmod M = yR^2R^{-1} \bmod M = MM(y, R^2)$$

Donde *MM* es la "multiplicación modular Montgomery", $R=(n^2)$ es una constante pre-almacenada en memoria, y *n* es la longitud de palabra. Como se demuestra en las ecuaciones anteriores, se puede realizar la conversión de dominios utilizando solo una "multiplicación modular Montgomery". Además, por el axioma de distribución ($xR+yR = (x+y)R$), los números en el dominio Montgomery se pueden sumar y restar como si estuvieran en el

dominio normal, mientras que la multiplicación debe realizarse con la "multiplicación modular Montgomery".

3.3.2. Segunda Etapa: Multiplicación de punto

La multiplicación de punto se implementa a partir del algoritmo *Montgomery Ladder* modificado (algoritmo 5). Esta es la operación principal y la que exige la mayor parte del tiempo de procesamiento.

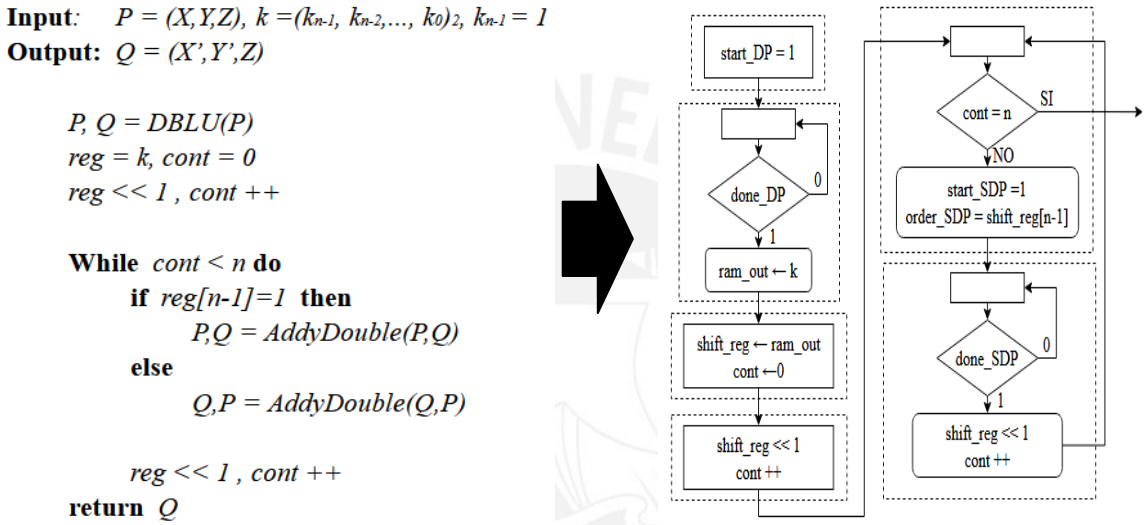

Figura 3.3. a) algoritmo Montgomery Ladder modificado y re-arreglado b) ASM de la Multiplicación de punto

En lado izquierdo de la Figura 3.3 se muestra el algoritmo modificado y re-arreglado para que pueda ser ejecutado con las operaciones disponibles en el *datapath*. Una de las modificaciones consiste en unir la operación de suma y duplicación de punto (*AddyDouble*) para que pueda ser ejecutado por un módulo independiente. Por otro lado, en la parte derecha se observa la implementación del algoritmo utilizando la técnica *Algorithmic State Machine* (ASM). Esta implementación comprende seis estados lógicos. En el primero, se activa el módulo de duplicación de punto (DBLU). En el siguiente estado se espera que la operación anterior termine para luego leer el valor de *k* almacenado en la RAM. En el tercer estado, se carga el valor de *k* en el registro de desplazamiento del *datapath.* Este registro dispone de un contador que aumenta cada vez que se realiza una operación de desplazamiento lo que permite saber cuándo se han recorrido todos los bits en el registro. En el quinto estado, se evalúa este contador para determinar si ya se han recorrió todos los bits *k* en cuyo caso se termina el algoritmo. En caso contrario, se activa el bloque de "suma y duplicación de

punto", que ejecuta las operaciones ZADDC y ZADDU en distinto orden dependiendo del valor de la señal "order_SDP". En el último estado, se espera la señal "done_SDP" que indica que la operación anterior concluyó y vuelve a iniciar el bucle.

3.3.3. Tercera Etapa 1: Coordenadas Z a *affine*

El primer paso de la tercera etapa consiste en transformar el resultado *Q(X,Y,Z),* obtenido en la segunda etapa, a coordenadas *affine.*

$$(X, Y, Z) \rightarrow (x, y) \quad donde \quad x = XZ^{-2}, y = YZ^{-3}$$

Esta operación se puede reordenar como se muestra en el algoritmo 3.1. De esta forma, solo se requieren de dos tipos de operadores: la "multiplicación modular Montgomery" y un "inversor modular Montgomery".

Algoritmo 3.1. Algoritmo para transformar coordenadas Z a *Affine*

Requiere: *P=(X,Y,Z)*
Calcula: *A=(X,Y)*

T1 = Z -1
T2 = T1 T1*
X = X T2*
T3 = T1 T2*
Y = Y T3*

a) Inversor modular Montgomery

Una forma eficiente de implementar esta operación es utilizando el teorema de Fermat.

$$a^{-1} \bmod M = a^{M-2} \bmod M, \qquad si \ mcd(a, M) = 1$$

Como en criptográfica de Curvas Elípticas el valor de *M* siempre es un número primo, se cumple que no tiene ningún divisor común con otro número. Por lo tanto, se puede usar el teorema de Fermat en la implementación. De esta manera la inversión modular se reduce a una operación exponencial que puede implementarse utilizando múltiples "multiplicaciones modulares Montgomery". La operación exponencial también se puede optimizar utilizando el algoritmo de cuadrados y multiplicaciones repetitivas [19], la cual se muestra en la parte izquierda de la Figura 3.4.

Calcula: $T1 = Z^{-1} mod\ M = Z^{M-2} mod\ M$

reg = M-2, cont = 0
T1 = Z
reg << 1 , cont ++

While *cont < n* **do**
Z = MM(Z,Z)
if *reg[n-1]=1* **then**
T1 = MM(T1,Z)

reg << 1 , cont ++
return *T1*

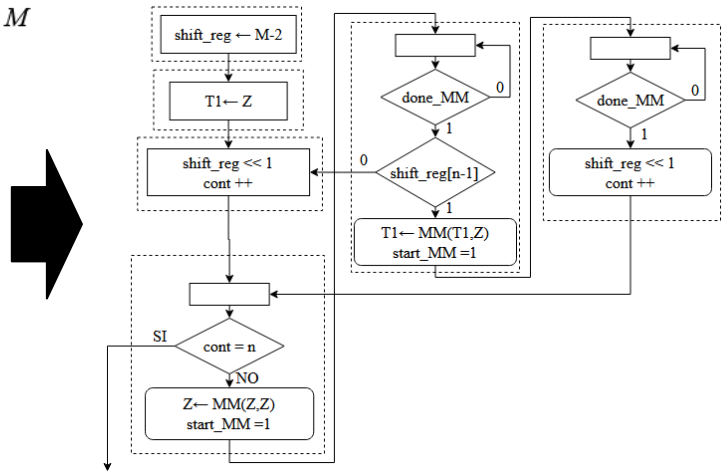

Figura 3.4. a) algoritmo de cuadrados y multiplicaciones repetitivas b) ASM de inversor modular

La implementación ASM de este algoritmo utiliza seis estados. Los tres primeros inicializan el registro temporal T1 y el registro de desplazamiento, a la vez que se realiza una operación de desplazamiento. Este registro dispone de un contador que aumenta cada vez que se realiza una operación de desplazamiento lo que permite saber cuándo se han recorrido todos los bits en el registro. En el cuarto estado, se evalúa este contador para determinar si ya se recorrió todos los bits en el registro en cuyo caso se termina el algoritmo. En caso contrario, se realiza una multiplicación modular Montgomery y se espera que la operación termine. Si el bit más significativo del registro de desplazamiento es '1', se ejecuta otra multiplicación modular Montgomery. En ambos casos se realiza un desplazamiento del registro y se reinicia el bucle.

3.3.4. Tercera Etapa 2: Montgomery a Normal

El segundo paso de la tercera etapa transforma las coordenadas *(x', y')* del dominio Montgomery al normal haciendo uso de las siguientes fórmulas.

$$x = x\ mod\ M = xR\ R^{-1}\ mod\ M\ = MM(xR, 1) = MM(x', 1)$$

$$y = y\ mod\ M = yR\ R^{-1}\ mod\ M = MM(yR, 1) = MM(y', 1)$$

El proceso es similar al seguido para la conversión de Normal a Montgomery, pero en vez de multiplicar por R^2 se multiplica por la unidad. Una vez concluida la transformación, la unidad de control activa la señal "done_uc" que le indica a la interfaz SPI que los registros *X* e *Y* ya pueden ser enviados al microcontrolador.

3.4. Diseño de las unidades de control complementarias

Además de la unidad de control principal, se han implementado otras tres unidades de control complementarias que realizan operaciones complejas: la "duplicación de punto", la "suma y duplicación de punto", y la "suma y multiplicación modular". Estos módulos son utilizados por la unidad de control principal, el cual inicia la operación al activar la señal "start" respectiva, y luego espera hasta que reciba la señal "done" que indica que la operación se completó. En las siguientes secciones se detalla la implementación de cada uno de estos módulos.

3.4.1. Operación de duplicación de punto

La operación de duplicación de punto solo se utiliza al inicio del algoritmo *Montgomery Ladder* modificado (Figura 3.3.a), justo después del cambio de coordenadas de *affine* a Coordenadas Z. Por ello, se puede asumir que $Z=1$ al inicio de esta operación, lo que permite reducir la fórmula DBLU propuesta por Melodi [17] a la mostrada en algoritmo 3.2.

Algoritmo 3.2. Algoritmo duplicación modular

Require: $P = (X, Y, 1)$
Calcula: $2P = (X\_2, Y\_2, Z)$

1.	$X\_2 \leftarrow Y * Y$	
2.	$Y\_2 \leftarrow X\_2 * X\_2$	
3.	$T3 \leftarrow X * X$	$, X \leftarrow X + X\_2$
4.	$X \leftarrow X * X$	
5.	$T3 \leftarrow n\_3 * T3$	$, X \leftarrow X - T3$
6.		$, X \leftarrow X - Y\_2$
7.	$X \leftarrow n\_2 * X$	$, T3 \leftarrow T3 + a$
8.	$T1 \leftarrow n\_8 * Y\_2$	
9.	$T2 \leftarrow T3 * T3$	
10.	$Z \leftarrow n\_2 * X$	
11.	$Z \leftarrow n\_2 * Y$	$, X\_2 \leftarrow T2 - Z$
12.	$Y \leftarrow n\_8 * Y\_2$	$, T2 \leftarrow X - X\_2$
13.	$Y\_2 \leftarrow T3 * T2$	
14.		$, Y\_2 \leftarrow Y\_2 - T1$

Como se observa, en cada línea de este algoritmo se realiza una secuencia de "multiplicación modular Montgomery" y "suma/resta modular". Para reducir la complejidad de la implementación, se ha diseñado otra unidad de control que realiza estas dos operaciones de forma consecutiva (Sección 3.3.3). De esta manera, el algoritmo se puede implementar utilizando quince estados ASM, donde cada estado representa una línea del algoritmo 3.2

incluyendo un estado inicial de reposo. Por otro lado, se observa que se utilizan tres registros temporales durante la operación.

3.4.2. Operación de suma y multiplicación modular

Como las operaciones de suma y duplicación modular son recurrentes durante la implementación de los algoritmos 3.2 y 3.3, se ha implementado una unidad de control que realiza estas dos operaciones en secuencia. Primero, se inicializa con las direcciones de memoria de los operandos y del registro donde se almacenará el resultado de cada operación. Además, ya que algunas partes en los algoritmos solo realizan la multiplicación o solo la Suma/Resta modular, se han incluido unas señales que habilitan cada una de estas operaciones: mul_enable y add_enable. También permite seleccionar entre la operación de suma o resta utilizando la señal add_sub. Por último, una vez inicializado el módulo, para iniciar las operaciones se activa la señal “start” y se espera hasta que la señal “done” indique que se completó la operación.

3.4.3. Operación de suma y duplicación de punto

Otra operación utilizada en el algoritmo *Montgomery Ladder* modificado (Figura 3.3.a) es la “suma y duplicación de punto” que se ejecuta en cada bucle. Esta operación está compuesta por las operaciones ZADDU y ZADDC[17] ejecutadas secuencialmente. El orden de la operación y la dirección de los registros está determinado por la señal “order_SDP”, para corresponder a la lógica del algoritmo 2.5. La implementación de esta operación es similar al de la operación de duplicación de punto. En este caso se puede implementar utilizando treinta estados ASM donde cada estado representa una línea en del algoritmo 3.3 incluyendo un estado de inicio. Por otro lado, en total se requieren cinco registros temporales para esta operación.

Algoritmo 3.3. Algoritmo de suma y duplicación modular

Require: $P = (X1, Y1, Z)$ $Q = (X2, Y2, Z)$
Calcula: $R, S = (2P, P+Q)$

1.		$, K \leftarrow X1 - X2$
2.	$T1 \leftarrow K * K$	$, K \leftarrow Y1 - Y2$
3.	$T3 \leftarrow X1 * T1$	$, T2 \leftarrow Y1 - Y2$
4.	$T1 \leftarrow X2 * T1$	
5.	$T4 \leftarrow K * K$	
6.		$, K \leftarrow T3 - T1$
7.	$T5 \leftarrow Y1 * K$	$, K \leftarrow T4 - T3$
8.		$, K \leftarrow K - T1$
9.		$, T4 \leftarrow T3 - K$
10.	$T2 \leftarrow T2 * T4$	$, T4 \leftarrow X1 - X2$
11.		$, X1 \leftarrow K$
12.	$Z \leftarrow Z * T4$	$, K \leftarrow Y1 + Y2$
13.	$K \leftarrow K * K$	$, Y1 \leftarrow Y1 + Y2$
14.		$, K \leftarrow K - T3$
15.		$, X2 \leftarrow K - T1$
16.		$, K \leftarrow T3 - X2$
17.	$K \leftarrow Y1 * K$	$, Y1 \leftarrow T2 - T5$
18.		$, Y2 \leftarrow K - T5$
19.		$, K \leftarrow X1 - X2$
20.	$T1 \leftarrow K * K$	$, K \leftarrow Y1 - Y2$
21.	$T3 \leftarrow X1 * T1$	$, Y2 \leftarrow Y1 - Y2$
22.	$T2 \leftarrow X2 * T1$	$, T1 \leftarrow X1 - X2$
23.		$, X1 \leftarrow T3$
24.	$K \leftarrow K * K$	$, X2 \leftarrow X1 - T2$
25.	$Y1 \leftarrow Y1 * X2$	$, K \leftarrow K - X1$
26.		$, X2 \leftarrow K - T2$
27.		$, K \leftarrow X1 - X2$
28.	$Y2 \leftarrow Y2 * K$	
29.	$Z \leftarrow Z * T1$	$, Y2 \leftarrow Y2 - Y1$

3.5. Diseño del Controlador de Bus

Como se describió anteriormente, hay cuatro unidades de control que realizan operaciones aritméticas, para ello cada una necesita acceso al *datapath* en cierto momento. De esta manera, para gestionar un acceso sin colisiones se usa un multiplexor que da acceso al bus de las señales de control del *datapath*, como se muestra en la Figura 3.1. Este multiplexor es controlado por la unidad de control principal que es quien el acceso a estas señales.

3.6. Diseño del *datapath*

En el *datapath* se realizan las operaciones aritméticas y está compuesto principalmente por un multiplicador modular Montgomery, un Sumador/Restador modular, y un registro de desplazamiento con contador como se muestra a continuación.

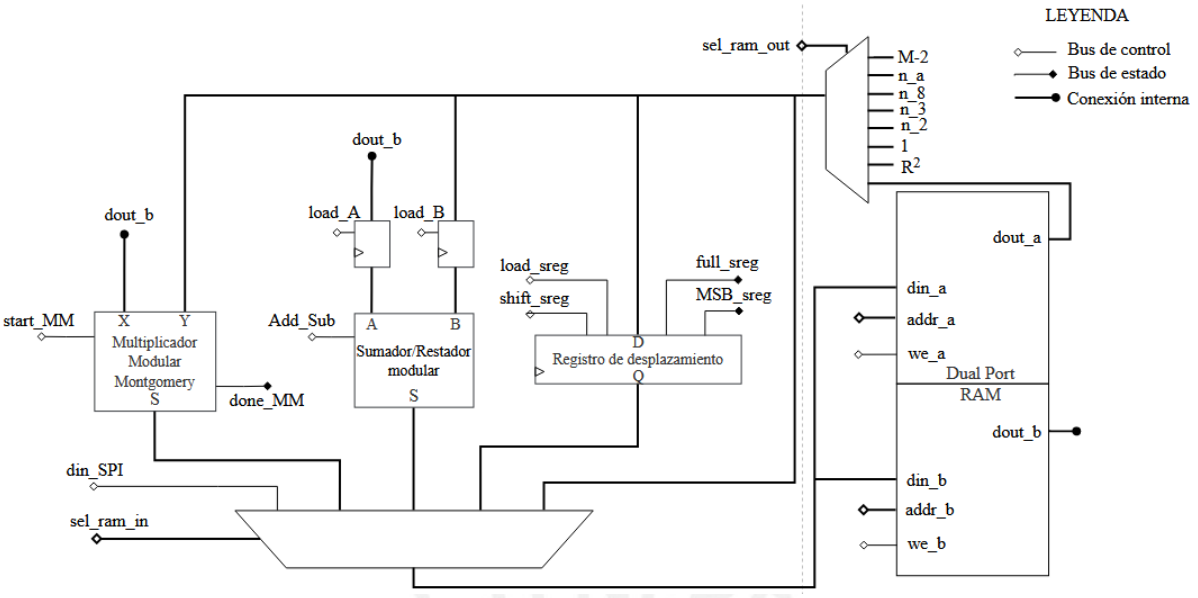

Figura 3.5. Arquitectura del datapath y memoria RAM Dual-Port

En la Figura 3.5 se muestra la arquitectura del *datapath*. Como se observa en la "leyenda", las flechas con un rombo blanco hacen referencia a las señales que componen al bus de control, mientras que las de flecha con rombo negro, representan a las salidas del *datapath* que indican a las unidades control el estado de los módulos aritméticos. Por otro lado, también se muestran las conexiones con la memoria RAM la cual da acceso a los registros que se utilizan durante las operaciones. La memoria RAM que se utiliza es Dual-Port, es decir, permite que se realicen dos accesos de lectura o escritura a la vez. En algunos casos las operaciones se realizan con alguna constante, es por ello que se ha implementado un multiplexor a la salida de la RAM, que permite seleccionar entre diversos números predefinidos como $R^2 = (n^2)^2$ (donde n es el tamaño de la palabra), a (uno de los parámetros de la curva elíptica que es determinada por el estándar utilizado), n_2, n_3, n_8 (representación de los números 2, 3 y 8 en el dominio Montgomery).

Asimismo, la entrada del Sumador/Restador se registra para evitar la disipación de potencia debido a *glitches*, que son transiciones indeseadas de las señales que no aportan a la funcionalidad del algoritmo. Con respecto al multiplicador modular sus entradas ya son registradas internamente. En las siguientes secciones, se detallará la implementación de cada uno de los módulos principales del *datapath*.

3.6.1. Registro de desplazamiento

El registro de desplazamiento es utilizado en los algoritmos de las Figuras 3.3 y 3.4. En estas operaciones se hace uso del desplazamiento para recorrer todos los bits del valor cargado en el registro. El bit que se utiliza para las operaciones es el bit más significativo "MSB_sreg". Además, se ha implementado un contador interno que aumenta cada vez que se realiza una operación de desplazamiento y que se reinicia a cero cada vez que se carga un nuevo dato. De esta manera, cuando se haya recorrido todos los bits del registro se activa la señal "full_sreg".

3.6.2. Sumador-Restador modular

El sumador-restador modular se puede implementar de manera eficiente utilizando el "método de Omura" [20]. Con este método, la suma modular se implementa como lo indica la ecuación 3.1 y la resta modular como la ecuación 3.2. La ventaja de este método es que permite la implementación de ambas operaciones utilizando el mismo circuito, siempre y cuando los números negativos sean representados en complemento a dos.

$$S = A + B \ (mod\, p) = \begin{cases} A + B - p, & A + B \geq 2^m \\ A + B & otherwise \end{cases} \quad \text{..... (3.1)}$$

$$S = A - B \ (mod\, p) = \begin{cases} A - B + p, & if\ A - B < 0 \\ A - B & otherwise \end{cases} \quad \text{..... (3.2)}$$

El circuito es implementado utilizando sumadores *Ripple Carry* ya que a pesar de que tienen un retardo alto, son los que disipan menos potencia entre las topologías de sumadores más comunes [21]. En la figura 3.6 se muestra el circuito sumador/retador modular. La señal 'Add_Sub' permite convertir el sumador modular en un restador modular, al invertir el valor de B e introducir un bit de acarreo de inicio. Además, el multiplexor de salida determina si el resultado es *A+B* o *A+B-p*, para el caso de suma modular, y su respectivo complemento para la resta modular.

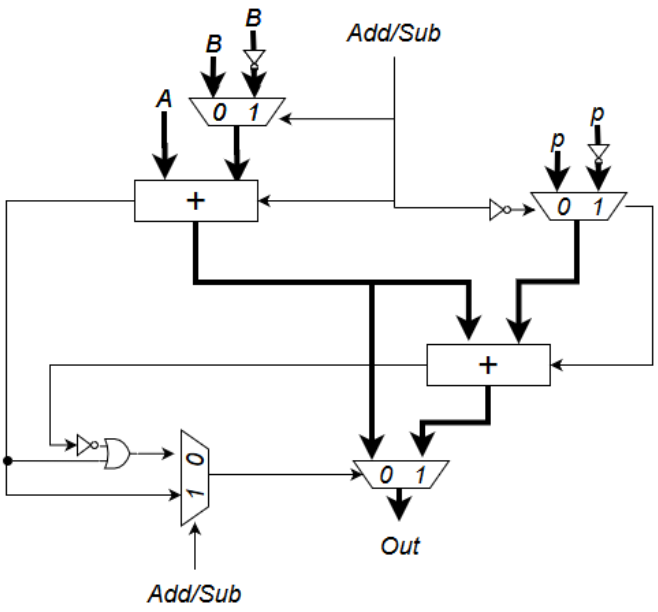

Figura 3.6. Circuito sumador/restador modular

3.6.3. Circuito multiplicador modular Montgomery

Una implementación eficiente del multiplicador modular es importante especialmente en sistemas criptográficos donde se operan números de gran tamaño. El algoritmo de "multiplicación modular Montgomery" es una manera eficiente de implementar esta operación, ya que remplaza la división del módulo M por divisiones por 2 que no exigen muchos recursos. Para poder utilizar este algoritmo, se exige una condición inicial que consiste en transformar los operandos al dominio Montgomery, como se explicó en secciones pasadas. Luego, la multiplicación Montgomery se define de la siguiente manera:

$$S = MM(X, Y) = X\,Y\,R^{-1} \mod M,$$

Donde X, Y son números de n bits de longitud, $R=2^n$ y M es un entero en el rango de $2^{n-1} < M < 2^n$, tal que $MCD(R, M) = 1$. Como en aplicaciones criptográficas el número M suele ser primo, se cumple la última condición. Una manera de implementar esta ecuación es utilizando el algoritmo "Radix-2 Montgomery multiplication" (algoritmo 3.4). En este algoritmo se computa sumas parciales de operaciones de cada bit de X con Y. Esto permite una implementación muy eficiente porque utiliza operaciones simples como multiplicaciones de palabra con bit, división por 2 que se puede implementar como un desplazamiento de bits y sumas.

Algoritmo 3.4. Algoritmo Radix-2 Montgomery Multiplication

$S_0 = 0$
for $i = 0$ to $m - 1$
    if $(S_i + x_i Y)$ is even
        then $S_{i+1} := (S_i + x_i Y)/2$
        else $S_{i+1} := (S_i + x_i Y + M)/2$
if $S_m \geq M$ then $S_m := S_m - M$

En [22], Tenca y Koç introducen una modificación de este algoritmo que reduce a *Y* en palabras cortas para cada operación. De esta manera se consigue una implementación que requiere menos área además que se reduce *fanout* lo cual disminuye la potencia dinámica. En este nuevo algoritmo, el operando *Y* es escaneado palabra por palabra y el operando *X* es escaneado bit por bit como se muestra en el Algoritmo 3.5. Este algoritmo computa sumas parciales *S* por cada bit de *X*, escaneando palabras de *Y* y *M*. Una vez se completan todas las palabras de *Y*, se toma otro bit de *X* y se repite el escaneo. Asimismo, la variable de acarreo *C* debe ser de dos bits para que no se desborde.

Algoritmo 3.5. Algoritmo MWR2MM para la multiplicación Modular Montgomery

Input: odd $M, n = \lfloor \log_2 M \rfloor + 1$, word size w, $e = \lceil \frac{n+1}{w} \rceil$, $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $Y = \sum_{j=0}^{e-1} Y^{(j)} \cdot 2^{w \cdot j}$, $M = \sum_{j=0}^{e-1} M^{(j)} \cdot 2^{w \cdot j}$, with $0 \leq X, Y < M$
Output: $Z = \sum_{j=0}^{e-1} S^{(j)} \cdot 2^{w \cdot j} = \mathrm{MP}(X, Y, M) \equiv X \cdot Y \cdot 2^{-n} \pmod{M}$, $0 \leq Z < 2M$

1: $S = 0$
2: **for** $i = 0$ **to** $n - 1$ **do**
3: $\quad q_i = (x_i \cdot Y_0^{(0)}) \oplus S_0^{(0)}$
4: $\quad (C^{(1)}, S^{(0)}) = x_i \cdot Y^{(0)} + q_i \cdot M^{(0)} + S^{(0)}$
5: $\quad$ **for** $j = 1$ **to** e **do**
6: $\quad\quad (C^{(j+1)}, S^{(j)}) = C^{(j)} + x_i \cdot Y^{(j)} + q_i \cdot M^{(j)} + S^{(j)}$
7: $\quad\quad S^{(j-1)} = (S_0^{(j)}, S_{w-1\ldots1}^{(j-1)})$
8: $\quad S^{(e)} = (0, S_{w-1\ldots1}^{(e)})$
9: **return** $Z = S$

a) Paralelización de MWR2MM

Huang et al. [23] analizaron la dependencia entre las operaciones dentro de cada iteración del algoritmo 3.5, para encontrar las que restringen la ejecución en paralelo. Notaron que es posible paralelizar instrucciones en diferentes iteraciones de la variable *i*. En la Fig. 3.16 se muestra la gráfica de dependencia creada por Huang et al. [23], donde cada circulo representa

una operación atómica. La tarea inicial D ejecuta las líneas 3 y 4 del algoritmo 3.5, mientras que la tarea E ejecuta las líneas 6 y 7, que son las que están dentro del bucle. Se observa a partir de esta gráfica que se puede obtener un alto grado de paralelización y pipelining.

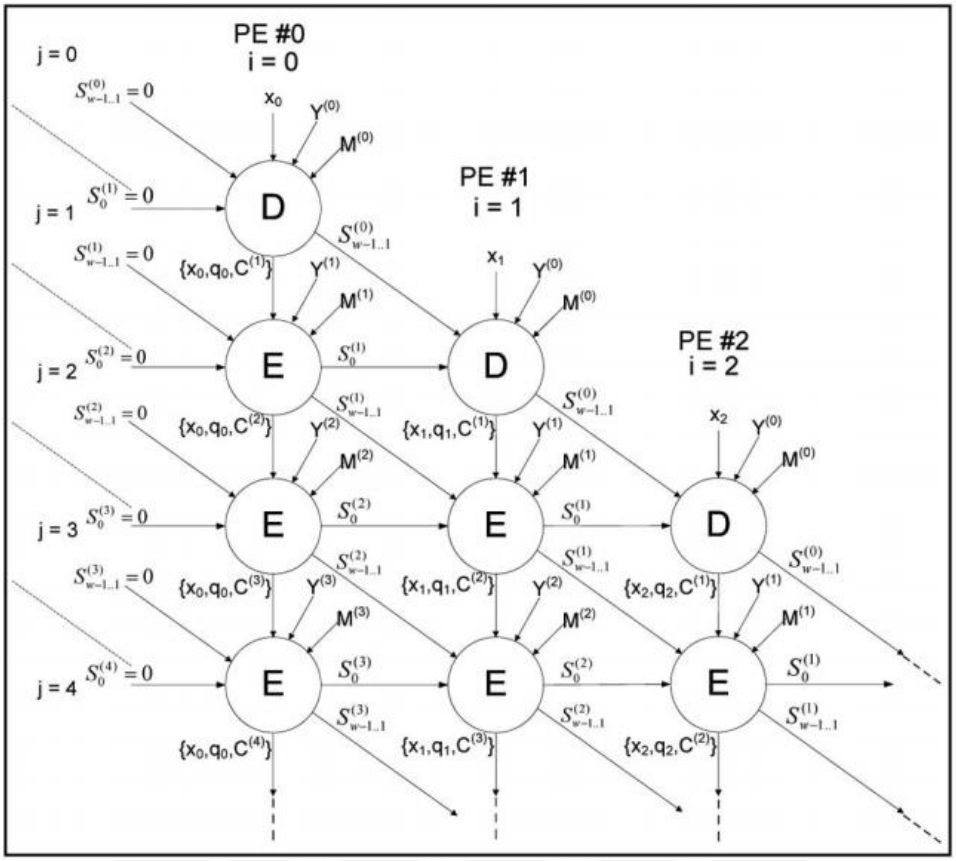

Figura 3.7. Gráfica de dependencia de datos del algoritmo MWR2MM [23]

Cada columna representa las tareas que pueden ser realizadas por un elemento de procesamiento (PE). Los datos generados por una PE se pasan a la siguiente en forma de pipeline. La novedad que introduce Huang et al. [23] es que para acelerar la ejecución precomputa en cada nodo los dos posibles resultados parciales que se resuelven en el siguiente ciclo. De esta manera se puede pasar parte de los datos al siguiente PE y enviar el resto cuando se resuelva en el siguiente ciclo. Esto permite que se consiga un retardo entre cada PE de solo un ciclo de reloj sin causar una gran penalidad en área. En consecuencia, se puede llegar a ejecutar una multiplicación Montgomery de números de *n* bits de longitud en hasta n ciclos de reloj en el mejor de los casos.

b) Diseño de la arquitectura de MWR2MM

En la Figura 3.17 se muestra la arquitectura propuesta para implementar el algoritmo MWR2MM. Primero, se muestra la unidad de control que recibe la señal externa “start_MM” para iniciar la operación y activa la señal “done_MM” cuando se finaliza. Este bloque se encarga de controlar y sincronizar todos los demás bloques. Para ello, utiliza dos contadores que llevan el registro de las variables *i* y *j*. Por otro lado, ya que según el diagrama de dependencia (Figura 3.16), cada PE recibe la data en palabras de *w* bits, la mejor manera de almacenar los valores *Y* y *M* es en rotadores con desplazamiento de *w* bits, mientras que *X* se almacena en un registro con *p* bits de desplazamiento, donde *p* es la cantidad de PE que componen el arreglo sistólico. En cada ciclo de reloj el arreglo sistólico recibe los siguientes *w* bits Y_j y M_j, y cada vez que se concluya una columna PE se enviarán los siguientes *p* bits de *X*.

Si la $e > p$, donde *e* es la cantidad de palabras $(e=n/w)$ y *p* es la cantidad de PE, entonces se necesitará agregar una cola de *w* bits de palabra para sincronizar el último PE con el primero. En este caso, la longitud de la cola L será de $e - p$ elementos.

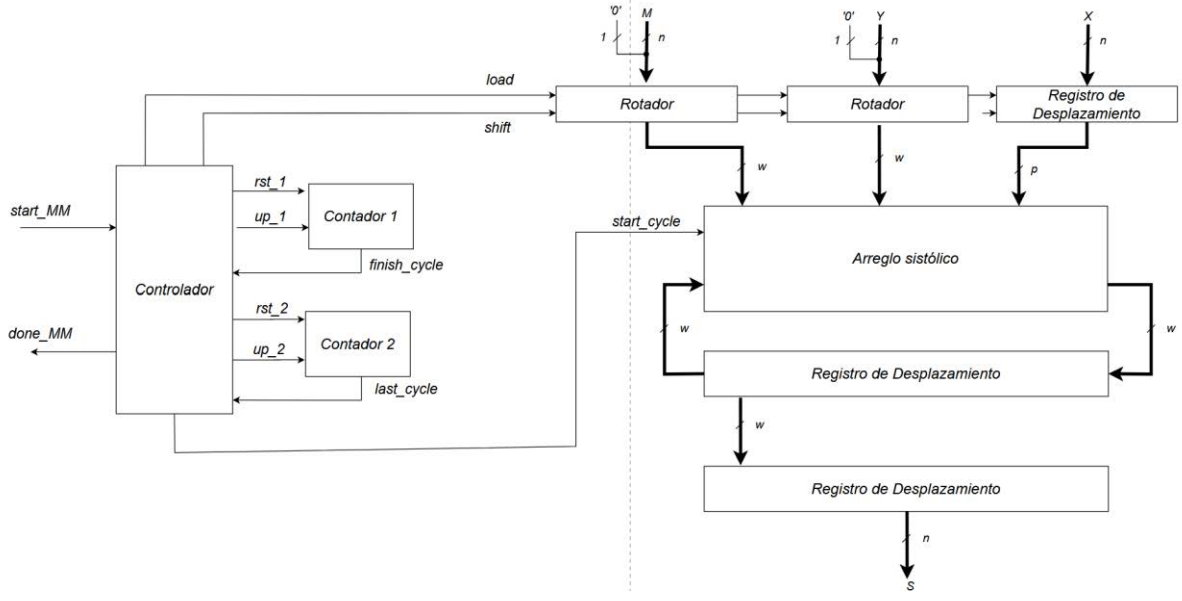

Figura 3.8. Arquitectura propuesta para el algoritmo MWR2MM

Por último, en la figura 3.18 se muestra la arquitectura interna del arreglo sistólico. Esta arquitectura debe permitir que se siga el diagrama de dependencias. Esto se consigue

fácilmente conectando los PE en cadena. Cada elemento puede ejecutar una tarea D o E. Esto se define con la señal “start_cycle” que se recibe al inicio de la operación. Los registros en cadena permiten que cada bloque PE reciba la señal ‘init’ un ciclo después del anterior. De misma manera los valores Y_j y M_j se van propagando a cada bloque. Finalmente, la implementación de los nodos PE se han implementado de la misma manera como lo propone Huang et al en. [23].

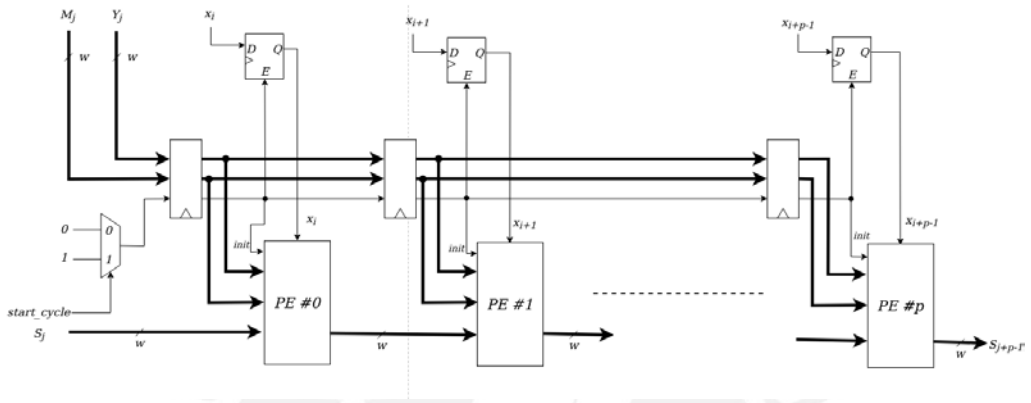

Figura 3.9. Arquitectura propuesta para el arreglo sistólico

Capítulo 4

Resultados

La arquitectura del procesador criptográfico presentado en el capítulo anterior se ha implementado enfocándose en obtener una baja disipación de potencia. Para ello, se trabajó en balancear los retardos de tiempo que se pudieran producir en los circuitos combinacionales para así evitar *glitches,* también en evitar las transiciones innecesarias producidas por un mal control de las señales. En las siguientes secciones se mostrará los resultados de la síntesis y de la simulación. Luego se compararán los resultados obtenidos con trabajos similares.

4.1. Resultados de la síntesis

La implementación se realizó en el FPGA igloo AGL1000V2 de Microsemi que utiliza tecnología flash. Este dispositivo dispone de 24576 elementos lógicos y 114 kbits de RAM. Para la síntesis e implementación se consideraron los siguientes puntos:

- El código se desarrolló en Verilog [24] y no se utilizó ningún bloque IP externo.
- La implementación se desarrolló utilizando el software Libero SoC 11.9.
- Los bloques fueron simulados utilizando la herramienta ModelSim.
- Los resultados son reportados luego del *Post-Place & Route*.
- Los resultados de potencia fueron obtenidos utilizando la herramienta SmartPower.
- Para la síntesis y simulación se eligió una curva de 192 bits que es el tamaño mínimo que recomienda el NIST para mantener una criptografía segura. Los parámetros de la curva que se usaron son los siguientes:
  - *Field: 0xe95e4a5f737059dc60dfc7ad95b3d8139515620f*
  - *Generador: (X, Y) = (0xb199b13b9b34efc1397e64baeb05acc265ff2378, 0xadd6718b7c7c1961f0991b842443772152c9e0ad)*
  - '*a*'*: 0x e95e4a5f737059dc60dfc7ad95b3d8139515620c*

Por otro lado, los resultados de la síntesis del módulo de "multiplicación modular Montgomery" van a depender de *p* (número de nodos PE) y *w* (tamaño de palabra de cada nodo). En la Tabla 4.1 se muestra los resultados de área, frecuencia máxima y potencia de las

implementaciones del procesador con distintos valores de *p* y *w*. La potencia y frecuencia máxima se obtienen usando las herramientas "SmartPower" y "Timing Analysis" respectivamente, que son complementos del software Libero SoC 11.9. Los valores de potencia se han calculado a una frecuencia de 6 MHz porque es la frecuencia máxima de la etiqueta WISP.

Tabla 4.1. Resultado de área, frecuencia máxima y potencia de la implementación

p (cantidad de nodos)	**w (ancho de palabra)**	**Número de unidades lógicas**	**Frecuencia Máxima (MHz)**	**Potencia Total a 6 MHz (mW)**
p =2	w = 2	14 971	12.043	5.754
	w = 4	14 954	11.990	5.859
	w = 8	15 358	11.809	5.895
p =4	w = 2	14 976	12.283	5.745
	w = 4	14 993	11.669	5.876
	w = 8	15 419	11.751	6.007
p =8	w = 2	15 214	12.026	5.814
	w = 4	15 798	11.953	5.738
	w = 8	15 728	11.835	6.074
p =16	w = 2	15 708	11.982	6.087
	w = 4	15 805	12.505	6.09
	w = 8	17 060	12.055	6.274

En esta tabla se observa que a medida que aumenta el tamaño de la palabra *w* y el número de nodos *p*, los valores de área y potencia suelen aumentar, pero no son cambios significativos. Es por eso que para los siguientes análisis se tomará una implementación considerando *p = 8* y *w = 4*. Esta configuración alcanza un bajo nivel de potencia y además

al tener valores de p y w altos se alcanza una menor latencia. En la Tabla 4.2 se muestra un análisis más completo de esta configuración. Para obtener los valores de velocidad de transmisión de datos (*throughput*) y la eficiencia se utilizan las ecuaciones 4.1 y 4.2.

$$Throughput = \left(\frac{1}{máxima\ frecuencia\ de\ operación} * latencia\ por\ ciclos\ de\ reloj\right)^{-1} \quad (4.1)$$

$$Eficiencia = \frac{Throughput}{Area} \quad (4.2)$$

Tabla 4.2. Resultados de la implementación con p = 8 y w = 4

Elementos Lógicos	Latencia (ciclos de reloj)	Potencia (mW)	Frecuencia máxima (MHz)	Throughput (op/seg)	Eficiencia (op/(slices.seg))
16246	4,157,358	5.738	11.953	2.875	176.975 x10^{-6}

4.2. Resultados de la simulación

La simulación se realiza utilizando la herramienta ModelSim. Se mostrarán las etapas del proceso de multiplicación de punto. Para ello se utilizan como valores de entrada los siguientes:

(X, Y) = (0xb199b13b9b34efc1397e64baeb05acc265ff2378,
0xadd6718b7c7c1961f0991b842443772152c9e0ad)
k = 0x88da3e0eacc70f73b57f20811f6ef579f9deeafe388e2deb

a) Primera Etapa: Normal a Montgomery

En esta etapa se realiza la transformación de X e Y al dominio Montgomery utilizando el módulo de Multiplicación Modular (MM). En la figura 4.1 se muestra en naranja los registros donde se almacenan los valores X e Y, y en verde las señales del módulo de multiplicación modular, donde *Y_datapath_MM* y *X_datapath_MM* son los operadores y *S_datapath_MM* es el resultado de la multiplicación. Se observa que al momento de activarse la señal *start_MM*, los valores del registro X y la constante R^2 se posicionan en las entradas *X_datapath_MM* y *Y_datapath_MM* para realizar la multiplicación que corresponde a la transformación $X=MM(x,R^2)$. De igual manera, en la siguiente multiplicación se realiza la transformación $Y=MM(y,R^2)$.

Figura 4.1. Simulación de la transformación de X e Y al dominio Montgomery

b) Segunda Etapa: Multiplicación de punto

En esta etapa se ejecuta el algoritmo *Montgomery Ladder modificado* de la Figura 3.3. En la siguiente Figura 4.2 se muestra en naranja los registros, en verde las señales del módulo *AddyDouble*, y en amarillo la señal del contador del registro de desplazamiento. En esta figura se observan n=191 iteraciones de la operación *AddyDouble()*. Durante este bucle los registros *X_reg, Y_reg, X2_reg y Y2_reg*, que representan los valores de *P=(X,Y)* y *Q=(X_2,Y_2),* se van actualizando tras cada operación completada de *addyDouble()*. Después de alcanzar n=191 iteraciones, la multiplicación de punto se completa y se pasa a la siguiente etapa.

c) Tercera Etapa 1: Coordenadas Z a *affine*

En esta etapa se vuelven a *affine* las coordenadas obtenidas como resultado de la multiplicación de punto. En la siguiente simulación (Figura 4.3) se muestra en naranja los registros, y en verde las señales del módulo de multiplicación modular, donde *Y_datapath_MM* y *X_datapath_MM* son los operadores y *S_datapath_MM* es el resultado de la multiplicación. La transformación de coordenadas se realiza utilizando el algoritmo 3.1. En la siguiente figura se muestran una a una las multiplicaciones modulares realizadas y como los respectivos registros se van actualizando hasta conseguir *X* e *Y* en coordenadas *affine.*

d) Tercera Etapa 2: Montgomery a Normal

Finalmente se vuelven las coordenadas al campo normal utilizando el módulo de Multiplicación Modular (MM). En la simulación (Figura 4.4) se muestra en naranja los registros donde se almacenan los valores *X* e *Y*, y en verde las señales del módulo de multiplicación modular, donde *Y_datapath_MM* y *X_datapath_MM* son los operadores y *S_datapath_MM* es el resultado de la multiplicación. Se observa que, de manera muy similar a la primera etapa, se consigue restaurar los valores *X* e *Y* a coordenadas normales tras dos multiplicaciones modulares. Después de realizar estas operaciones, se obtiene como resultado

(X, Y) = (0x9f24dc553e9daf8ef6d8744df2716de29b525e90,
0x8b312193942bbbcf67866487cbebd4c789d1127d)

Este resultado fue validado utilizando la librería criptográfica PyChrypto [25].

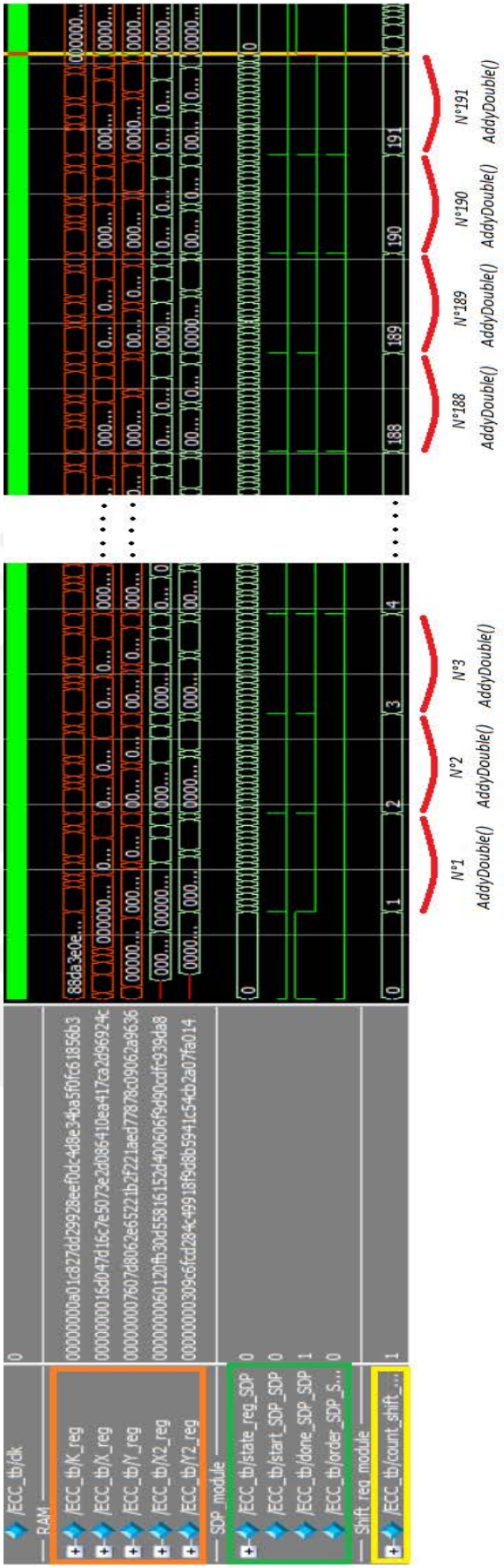

Figura 4.2. Simulación del algoritmo Montgomery Ladder

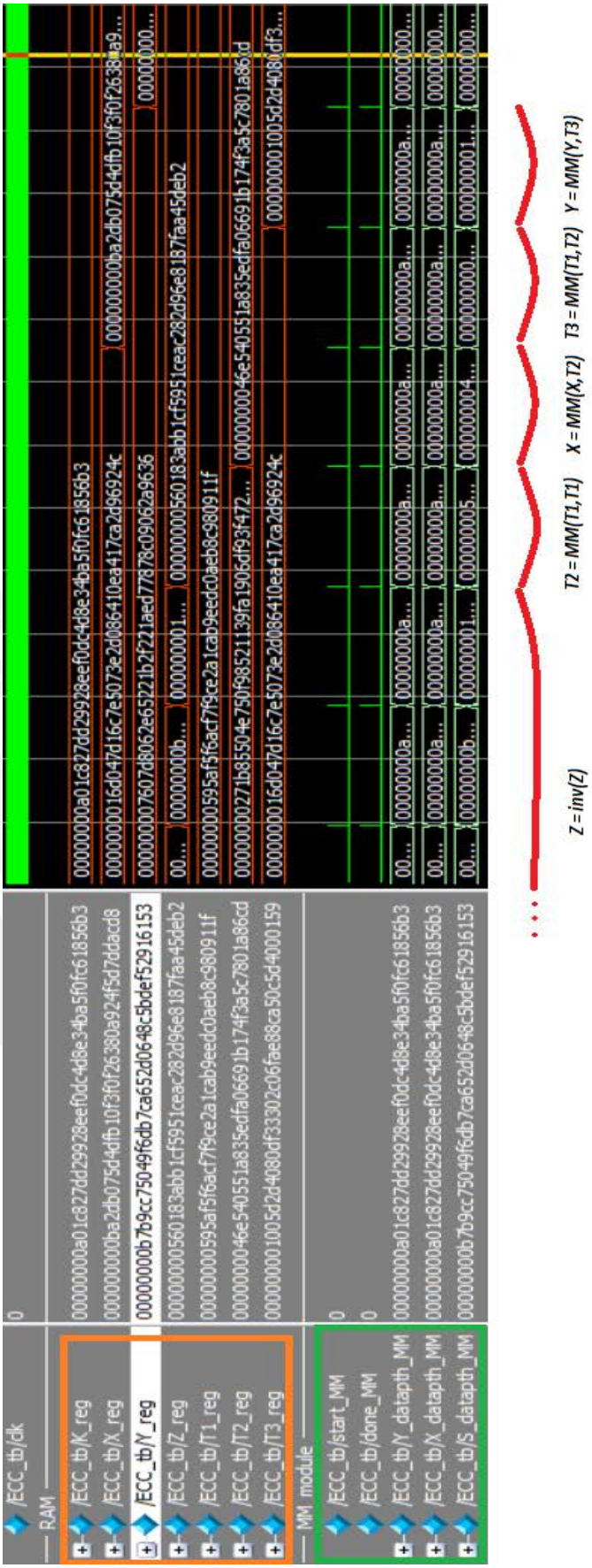

Figura 4.3. Simulación de transformación de coordenadas Z a affine

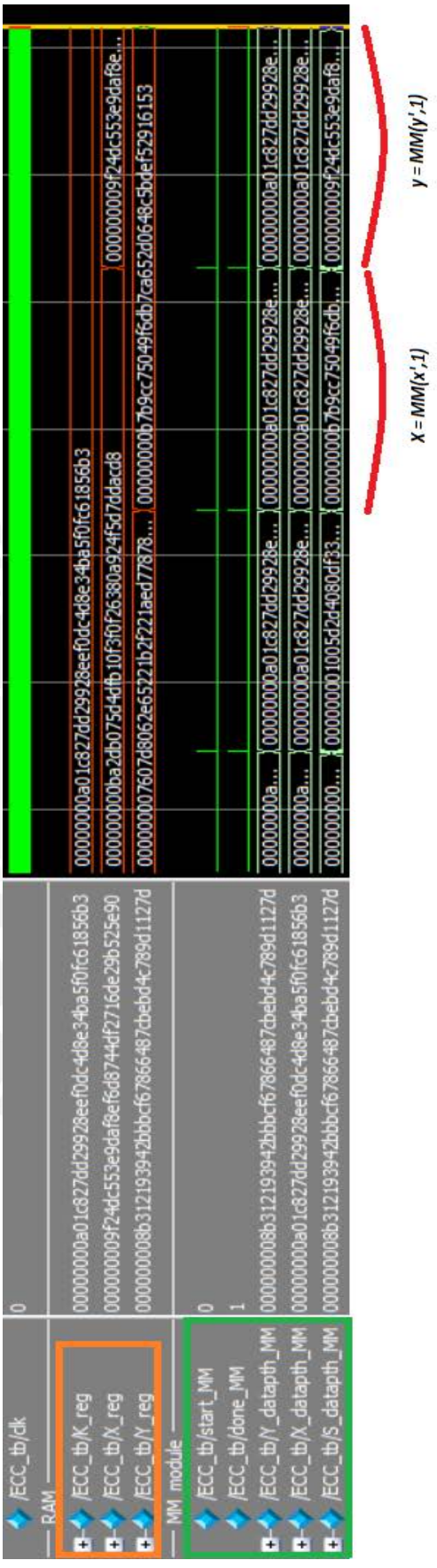

Figura 4.4. Simulación de la transformación de X e Y al dominio Normal

4.3. Comparación con otros trabajos

Existen múltiples publicaciones de implementaciones de Procesadores de curvas elípticas de bajos recursos en FPGA [14]. La mayoría se enfocan solo en reducir el área y son pocas las que muestran resultados de potencia. En la Tabla 4.3 se muestra tres implementaciones. La primera es propuesta por A. Salman [26]. Esta arquitectura se implementa en el FPGA Igloo 2 y ha sido diseñada para ser una implementación de bajos recursos alcanzando una potencia de 12 mW. La segunda es una implementación en software y fue diseñada específicamente para la etiqueta WISP. Se considera que disipa una potencia de 3.74 mW, que es la potencia que consume el microcontrolador a 6 MHz. Finalmente, la tercera implementación es la propuesta en este trabajo. Como se puede apreciar nuestro diseño alcanza un 47.8% de la potencia y 71.9% de la energía de la arquitectura propuesta por A. Salman [26]. Además, es un 153.4% de la potencia y 62% de la energía propuesta por C. Pendl, M. Pelnar y M.Hutter [8]. Por otro lado, se observa que la frecuencia y el área obtenidos son mayores que el del diseño de A. Salman. Sin embargo, ya que nuestro trabajo está enfocado en conseguir una baja potencia, se trabajará con frecuencias bajas, por lo que esto no representa un problema. Además, con la configuración utilizada solo se utilizan el 66.7% de la cantidad de elementos lógicos del igloo AGL1000V2.

Tabla 4.3. Análisis de potencia del procesador criptográfico a 6 MHz

Autor	Dispositivo	Elementos lógicos usados	Máxima frecuencia (MHz)	Latencia (ciclos de reloj)	Potencia (mW) @6MHz	Energía (mJ) @6MHz
A. Salman [26]	Igloo 2	6489	81	2,763,2523.	12	5.527
C. Pendl, M. Pelnar y M. Hutter [8]	MSP430 F233	-	6	10,289,883	3.74	6.414
Arquitectura propuesta en este trabajo	Igloo	16249	11.953	4,157,358	5.738	3.976

4.4. Resultados en la etiqueta WISP

A partir de los resultados obtenidos se puede estimar el comportamiento de nuestro procesador de curvas elípticas junto con la etiqueta WISP. En primer lugar, se debe tener en cuenta que la frecuencia máxima que alcanza WISP es de 6 MHz y la potencia recibida varía dependiendo a la distancia de la etiqueta al lector (ver tabla 2.1). En segundo lugar, se debe considerar que al igual que el microcontrolador de WISP, el FPGA tendrá periodos en los cuales estará activo y otros en los que estará en estado de baja energía. Este control debe estar a cargo del microcontrolador de WISP. Para ello, se aprovechan los modos de energía que dispone el FPGA Igloo Nano.

La potencia disipada por el diseño propuesto es de 5.74 mW y tiene una latencia de 692.9 ms a 6 MHz (4,157,358/6,000,000 segundos). Para que el dispositivo WISP pueda realizar la operación criptográfica a distancias mayores a unos centímetros donde no recibe la potencia suficiente, se debe disminuir el tiempo que el FPGA pasa en estado activo. A medio metro, por ejemplo, el FPGA estará activo solo (2.5 mW/5.74 mW) * 100 % = 43.55 % del tiempo. Por lo tanto, el tiempo que tomará completar una multiplicación de punto ya no será 692.9 ms, sino (692.9 ms/ 43.55 %) = 1.591 s. En la tabla 4.4 se muestran estos cálculos para distintas distancias.

Tabla 4.4. Tiempo de ejecución según la distancia a la antena

Distancia (m)	Potencia Recibida (uW)	Tiempo que pasa en estado Activo (DC%)	Tiempo de ejecución (s)
0.5	2500	43.55	1.59
1	271	4.72	14.68
2	67.6	1.18	58.84
3	30.1	0.52	132.13
4	17.0	0.30	233.96
5	10.8	0.19	368.26

Conclusiones

En este trabajo se presenta una arquitectura para un procesador criptográfico de Curvas Elípticas de bajo consumo energético. Para conseguir este objetivo, este diseño se implementa en un FPGA de tecnología Flash y se reduce el número de transiciones de señales, así como se evitan los *glitches* que puedan producir una disipación innecesaria de potencia. A partir de los resultados obtenidos llegamos a las siguientes conclusiones:

- La implementación del Procesador Criptográfico de Curvas Elípticas cumple con el objetivo general planteado, ya que este diseño disipa 5.74 mW que es casi la tercera parte de la potencia planteada en los objetivos que es de 15 mW.
- Como se explicó en el capítulo 2, las tareas que se ejecutan en WISP se miden por la energía que requieren para ejecutarse, ya que esto determina el tamaño del capacitor, así como el tiempo de carga y descarga. Por eso, es importante conseguir un diseño con bajo consumo de energía. Nuestro diseño alcanza un 71.9% de energía respecto a la propuesta de A. Salman [26] y un 62% de la implementación en software [8].
- En comparación con la librería criptográfica para WISP propuesta por C. Pendl, M. Pelnar y M. Hutter [8] que alcanza una máxima distancia de 40 cm y tiene un tiempo de ejecución de 56.2 segundos, el diseño que se propone en este trabajo toma aproximadamente un segundo y medio para ejecutar una multiplicación de punto a una distancia similar. Esto es 35 veces más rápido.

Recomendaciones y Observaciones

- El procesador de Curvas Elípticas presentado en este trabajo está basado en la ecuación Weierstrass para curvas elípticas. Por lo tanto, se puede implementar cualquier estándar basado en este tipo de curva. Esto abre la posibilidad de probar e investigar los nuevos estándares y protocolos de seguridad para RFID de una manera más rápida y fácil.
- En un futuro, se recomienda verificar el funcionamiento de la implementación física del procesador criptográfico integrado con la tarjeta WISP. Para ello se debe modificar el firmware de WISP que realice los protocolos de seguridad durante la comunicación ECP Gen2. Además, se recomienda evaluar experimentalmente la distancia máxima y el consumo energético que se puede lograr con la implementación física.
- En [5], se presenta un diseño en RTL del Core WISP y su implementación en ASIC. Tomando este trabajo como base, se plantea en un futuro integrar el procesador de curvas elípticas al Core WISP para poder conseguir una implementación en ASIC de WISP que soporte criptografía de curva elíptica. Una implementación en ASIC disminuiría drásticamente el consumo energético y mejoraría la frecuencia máxima del procesador.

Referencias

[1] L. Sujay, "Number of internet of things (IoT) connected devices worldwide in 2018, 2025 and 2030," *Statista*, 2021. https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/ (accessed Feb. 08, 2021).

[2] I. Mendez, "Estudio del diseño de un procesador criptográfico de Curvas Elípticas para el dispositivo WISP", tesis para optar el grado de Bachiller, Facultad de Ciencias e Ingeniería, Pontificia Universidad Católica del Perú, Lima, Perú, 2020.

[3] X. Jia, Q. Feng, T. Fan, and Q. Lei, "RFID technology and its applications in Internet of Things (IoT)," in *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, Apr. 2012, pp. 1282–1285. doi: 10.1109/CECNet.2012.6201508.

[4] S. S. Anjum *et al.*, "Energy Management in RFID-Sensor Networks: Taxonomy and Challenges," *IEEE Internet Things J*, vol. 6, no. 1, pp. 250–266, 2019, doi: 10.1109/JIOT.2017.2728000.

[5] D. J. Yeager, "Development and Application of Wirelessly-Powered Sensor Nodes," M.S. thesis, M.S. thesis, Dept. Elect. Eng., Washington Univ. Washington, USA, 2009.

[6] S. Naderiparizi, A. N. Parks, Z. Kapetanovic, B. Ransford, and J. R. Smith, "WISPCam: A battery-free RFID camera," *2015 IEEE International Conference on RFID, RFID 2015*, pp. 166–173, 2015, doi: 10.1109/RFID.2015.7113088.

[7] D. J. Yeager, J. Holleman, R. Prasad, J. R. Smith, and B. P. Otis, "NeuralWISP: A wirelessly powered neural interface with 1-m range," in *IEEE Transactions on Biomedical Circuits and Systems*, 2009, pp. 379–387. doi: 10.1109/TBCAS.2009.2031628.

[8] C. Pendl, M. Pelnar, and M. Hutter, "Elliptic Curve Cryptography on the WISP UHF RFID Tag," in *RFID. Security and Privacy*, 2012, pp. 32–47.

[9] Symantec, "2018 Internet Security Threat Report," 2018. Accessed: Jul. 22,2020. [Online]. Available: https://docs.broadcom.com/doc/istr-23-executive-summary-en.

[10] A. Ibrahim and G. Dalkiliç, "An Advanced Encryption Standard Powered Mutual Authentication Protocol Based on Elliptic Curve Cryptography for RFID, Proven on WISP," *J Sens*, vol. 2017, 2017, doi: 10.1155/2017/2367312.

[11] S. Gabsi, V. Beroulle, Y. Kieffer, H. M. Dao, Y. Kortli, and B. Hamdi, “Survey: Vulnerability Analysis of Low-Cost ECC-Based RFID Protocols against Wireless and Side-Channel Attacks,” *Sensors*, vol. 21, no. 17, p. 5824, Aug. 2021, doi: 10.3390/s21175824.

[12] M. Philipose, J. R. Smith, B. Jiang, A. Mamishev, S. Roy, and K. Sundara-Rajan, “Battery-free wireless identification and sensing,” *IEEE Pervasive Comput*, vol. 4, no. 1, pp. 37–45, 2005, doi: 10.1109/MPRV.2005.7.

[13] E. R. Naru, H. Saini, and M. Sharma, “A recent review on lightweight cryptography in IoT,” in *Proceedings of the International Conference on IoT in Social, Mobile, Analytics and Cloud, I-SMAC 2017*, 2017, pp. 887–890. doi: 10.1109/I-SMAC.2017.8058307.

[14] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, “Elliptic Curve Lightweight Cryptography: A Survey,” *IEEE Access*, vol. 6, pp. 72514–72550, 2018, doi: 10.1109/ACCESS.2018.2881444.

[15] B. Rashidi, “A Survey on Hardware Implementations of Elliptic Curve Cryptosystems,” *ArXiv*, pp. 1–61, 2017. [Online]. Available: http://arxiv.org/abs/1710.08336.

[16] M. Lochter and J. Merkle, “Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation,” Mar. 2010. doi: 10.17487/rfc5639.

[17] R. R. Goundar, M. Joye, and A. Miyaji, “Co-Z addition formulæ and binary ladders on elliptic curves,” in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES), ser. LNCS, vol. 6225. Springer,* 2010, pp. 65–79. doi: 10.1007/978-3-642-15031-9_5.

[18] Microchip Technology Inc., “IGLOO® FPGAs”. Accessed: Dec. 15,2022 [Online]. Available: https://www.microchip.com/en-us/products/fpga-and-plds/fpga/igloo-fpgas#

[19] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, “Math Background,” in *Handbook of Applied Cryptography*, CRC press, 1997. doi: 10.5860/choice.34-4512.

[20] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle, “Hardware implementation of an elliptic curve processor over GF(p),” in *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors. ASAP 2003*, 2003, pp. 433–443. doi: 10.1109/ASAP.2003.1212866.

[21] A. Misra, S. Birla, and N. Singh, “Comparative Analysis of Various Adder Architectures on 28nm CMOS Technology,” in *2020 7th International Conference on*

Computing for Sustainable Global Development (INDIACom), Mar. 2020, pp. 73–76. doi: 10.23919/INDIACom49435.2020.9083681.

[22] A. F. Tenca and Ç. K. Koç, "A scalable architecture for montgomery multiplication," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 1999, pp. 94--108. doi: 10.1007/3-540-48059-5_10.

[23] M. Huang, K. Gaj, and T. El-ghazawi, "New Hardware Architectures for Montgomery Modular Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 923–936, 2011, doi: 10.1109/TC.2010.247.

[24] S. Palnitka, *Verilog HDL: A Guide to Digital Design and Synthesis*, Second Edi. California, USA: Prentice Hall PTR, 2003. doi: 10.1201/9781315219547.

[25] D. Litzenberger, "pycrypto 2.6.1: Python Cryptography Toolkit", 2013. Accessed: Nov. 10, 2022. [Online]. Available: https://pypi.org/project/pycrypto/

[26] A. Salman, A. Ferozpuri, E. Homsirikamol, P. Yalla, J. P. Kaps, and K. Gaj, "A scalable ECC processor implementation for high-speed and lightweight with side-channel countermeasures," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2017, pp. 1–8. doi: 10.1109/RECONFIG.2017.8279769.

[27] I. Mendez, C. Silva, "A Low-Power Elliptic Curve Processor for WISP" in 2021 19th IEEE International New Circuits and Systems Conference (NEWCAS), 2021, pp. 1-4, doi: 10.1109/NEWCAS50681.2021.9462796.