Proyecto #2 Microelectrónica

Axel Gabriel Hernández Masís
Carné: B73690
José Pablo Camacho Cerdas
Carné: B71463

I. Introducción

Una función hash es aquella la cual asigna datos de tamaño arbitrario a valores de tamaño fijo, es decir un algoritmo el cual toma un bus de datos y lo transforma en una nueva serie de caracteres con una longitud de salida fija.

II. Desarrollo de la arquitectura

En la segunda parte del proyecto se realizó una descripción de los diseños propuestos tanto en optimización de área como de desempeño a nivel de RTL con el lenguaje Verilog, también se realizó una simulación de estos mediante la herramienta GTKwave y por último la sintetización de estos archivos a nivel de compuertas para distintas etapas del flujo de diseño para obtener así alguna métricas importantes. A continuación se presentarán el desarrollo de ambas arquitecturas así como sus resultados y comportamiento a nivel de señales.

III. Optimización en área

Para esta optimización se utilizó la misma arquitectura propuesta en la primera parte del proyecto, se presenta a continuación:

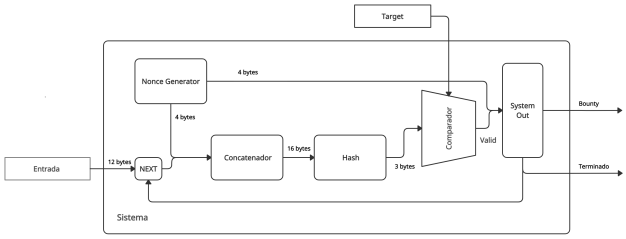

Figura 1: Arquitectura optimizada en área. Creación propia

Es importante anotar que se realizaron algunos cambios en la interconexión de bloques con el fin de sincronizar los bloques para obtener las salidas esperadas, a continuación se presenta una descripción de los bloques, sus conexiones y su función.

III-A. Next

Este bloque recibe una señal **finished** que da paso a la siguiente entrada lo que representa que se dio una salida esperada y se cumplió la condición con el target. Además recibe el valor de entrada, y el valor anterior para retener o dejar pasar la siguiente entrada, si el **finished** está en 1, pasa la siguiente entrada.

III-B. Nonce Generator

El generador de nonce es un contador el cual es reiniciado cuando el sistema recibe una señal de **success** la cual representa una salida positiva en el sistema cumpliendo la condición del target. Este generador de nonce se compone de un contador el cual aumenta gracias a una señal de **next** la cual representa que el bloque generado por la función hash no cumple la condición esperada.

III-C. Concatenador

El bloque concatenador reciben la salida del next que es la entrada inicial(la que esté entrando en el momento) y el nonce para generar la entrada a la función micro_hash

III-D. Hash

El bloque micro hash recibe la salida del concatenador que comprende la entrada externa del sistema junto a el nonce generado. Esta función realiza la lógica mediante la cual se obtiene un **valid_hash** de salida y el bloque **h** los cuales corresponden a la salida calculada por la función hash. Es importante anotar que cada vez que esta función reciba un **next** o un **finished**, la función hash se reiniciará y comenzará a calcular un nuevo bloque.

III-E. Comparador

El bloque comparador recibe como entrada la salida del hash, su valid y el target. Con esto si se cumple la condición en que el valor de **h** es mayor que la del target y el valid está activado, se activa la salida de valid la cual representa una correcta comparación, de otra forma se activa la salida next que representa una incorrecta comparación y lleva esta señal al generador de nonce para que realice la comparación con otro nonce.

III-F. System Out

Este bloque recibe el valid que es dado por el comparador, además del nonce asociado a esa operación que realizó la micro_hash. Si esta señal de valid se encuentra activa, se activa la señal **finished** la cual representa la correcta comparación y una salida exitosa para el sistema así como se activa salida del nonce, la cual corresponde al nonce generado con el cual se cumplió la condición. De otra forma no se activa la señal de finished, pero el sistema seguirá con el cálculo de un nuevo bloque **h** gracias a la señal next.

IV. Resultados para optimización en área

A continuación se presentan la simulación del sistema con optimización en área, a nivel de señales provistas desde un probador mediante el tool **GTKwave**:

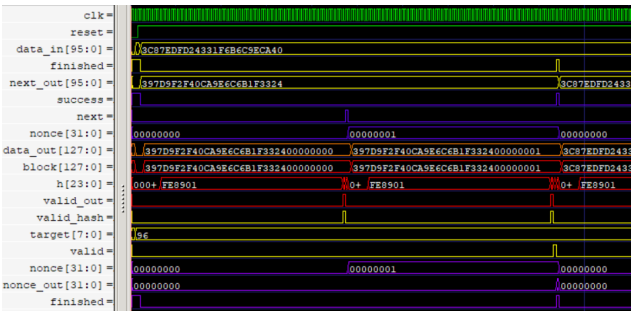

Figura 2: Comportamiento de sistema con optimización de área. Creación propia

De la figura 2 podemos ver el flujo de datos para el sistema y como este se comunica entre si, primero podemos ver como la señal **data_in** a pesar de cambiar, el sistema va a guardar la señal con la que comenzó a calcular su salida respectiva en un registor **next_out** el cual va a actualizarse solo si se da un cálculo exitoso en el sistema. También podemos notar que en la señal **nonce** comienza en cero y genera un nuevo nonce(aumenta el contador) cuando se activa **next** se activa o se resetea cuando la señal **finished** se encuentra en alto.

También de la figura anterior se puede apreciar como la señal **block** es la cual recibe el bloque **hash** y su salida respectivo se da cuando el **valid_out** se activa, es decir en el ciclo que esta señal de valid se activa el bloque **h** es el generado por la función hash. El bloque de comparador recibe esta señal **h** y el **valid_hash**, revisa que para este valor se cumpla con la condición en la cual dos de sus componentes sea mayores al valor del **target**, el cual en este ejemplo está seteado en 96 hexadecimal, si esta condición se cumple se activará un **valid** a la salida del comparador. Por último el bloque de system out va a recibir el valor del valid del comparador así como el bloque de nonce el cual respecta al nonce relacionado a la salida del hash actual y si este valid se encuentra en alto, se da una salida de **finished** en alto lo que representa un reseteo del sistema así como se deja entrar la salida la cual estaba esperando a que se cumpliera la condición.

Entre algunos puntos importantes de la figura 2 se puede notar que cuando no se da la comparación correcta, se levanta la señal **next**. Esta señal va a resetear **internamente** el sistema y comenzará a realizar un nuevo cálculo con el nuevo nonce generado. También podemos ver que la señal de **finished** representa un tipo de reset general ya que cuando esta se activa todas las señales comienzan desde cero, dejando paso a la nueva entrada. Es importante anotar que este diseño va a realizar el cálculo y generación de nonce hasta que se cumpla la condición, cuando esta se cumpla el sistema tomará la entrada que viene del exterior del mismo en ese momento.

IV-A. Métricas y desempeño del circuito

Para obtener las métricas del circuito se utilizó el tool **Qflow** mediante el cual se realizaron las etapas de **síntesis**, **placement** y **STA** para así obtener métricas relacionadas al flujo de diseño. En la siguiente tabla se presentan algunas de estas métricas:

Componente	Cantidad
Celdas Combinacionales	4597
Celdas Secuenciales	780
Buffers e Inversores	662
Total de Cables	4755
Total de Celdas	6039

Tabla I: Datos para optimización en área. Creación propia.

También mediante la etapa de placement se obtuvo el dato del área total del circuito:

$$\boxed{\text{Area = 2,61 mm}}$$

Seguido a esta etapa de placement, se realizó la etapa de Static Timimg Analysis con la cual se obtuvo la frecuencia máxima de operación para el circuito:

$$\boxed{\text{f}_{max} = 147,185MHz}$$

Por último se realizó el cálculo de tiempo basado en el funcionamiento de esta arquitectura, para esto se tomó el tiempo que le toma al sistema desde que recibe una entrada, genera su nonce y calcula su valor mediante la micro_hash. Este tiempo es el que le toma al sistema completo en realizar esta operación, para este se calculan la cantidad de ciclos de reloj y se multiplican por 1000 (caso del enunciado) y a su vez esta cantidad se multiplica por el periodo mínimo del circuito T_{min} el cual está asociado a la frecuencia máxima de operación. A continuación se presenta la cantidad de ciclos que toma el sistema en realizar una operación:

$$\boxed{\text{Cantidad}_{ciclos} = 74}$$

Ahora esta cantidad de ciclos se multiplica por 1000 (iteraciones para obtener un nonce válido) y a su vez por el periodo mínimo: el cual corresponde a:

$$\boxed{\text{T}_{minimo} = \frac{1}{f_{max}} = 6,76ns}$$

$$\boxed{\text{Tiempo}_{calculo} = 74 * T_{min} * 1000 = 502,77\mu s}$$

V. Optimización en desempeño

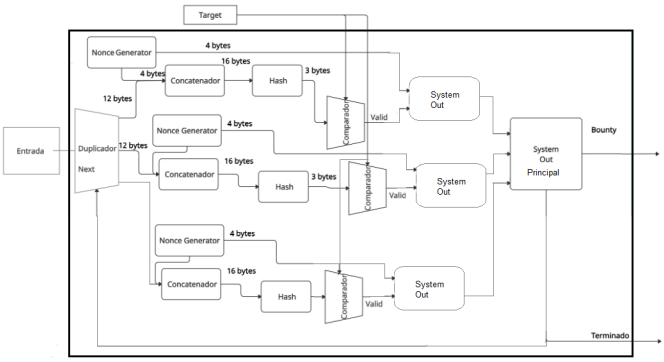

Figura 3: Bloque Desempeño. Creación propia

Como se observa en la imagen anterior comparándolo con la sección de área, este posee la diferencia de utilizar 3 concatenadores, 3 nonce generator, 3 comparadores, 3 hash y 3 valid. Estos bloques realizan las operaciones en paralelo, con el fin de reducir el tiempo en el que se encuentra un dato correcto, estos canales se unen en **system out** que retroalimenta el sistema dando paso a otro valor de entrada.

Los cambios realizados fueron pequeños, uno de estos fue en el generador de nonce, el cual posee una lógica en la cual el nonce generado por el primer generador es 0, para el segundo es 1 y para el tercero es 2, después de esto en caso de que no se de una correcta comparación en ninguno de estos, el contador va a aumentar en 3 para cada uno de estos nonce lo que dará como resultado que cada uno de los nonce sean diferentes y que vayan de 3 en 3 sin repetirse para aprovechar al máximo estos. También se genera un bloque extra llamado **System Out Principal** el cual recibe los 3 valid provenientes de los bloques system out y si alguno de estos, posee una señal activa el sistema se va a reiniciar mediante la señal de **finished**.

VI. Resultados para optimización en desempeño

A continuación se presentan la simulación del sistema con optimización en área, a nivel de señales provistas desde un probador mediante el tool **GTKwave**:

VI-A. Next

Se utiliza el mismo bloque que en el Next de área respetando las mismas especificaciones.

VI-B. Nonce Generator

Como se observa en la figura 3 se observa como este diseño posee 3 nonce que operan de manera independiente.Su logica es la misma que la del Nonce de área con la diferencia que cada uno de estos posee un desfase de un valor y dos entre ellos, además suman de 3 en 3 dando así 3 nonce de salida distintos.

VI-C. Concatenadores

Los concatenadores sigeun la lógica del concatenador de area, unen la salida de los nonce y de la entrada, con lo cual se tienen 3 valores distintos que son la entrada a los bloques microhash.

VI-D. Hash

Los bloques micro hash poseen la misma lógica de la micro hash de area, la diferencia es que al poseer 3 distintas, existen 3 salidas en paralelo que pueden ser comparadas de manera directa con el target.

VI-E. Comparadores

Los comparadores verifican el valor de salida con el target y envían una señal de valido a la salida del sistema.

VI-F. System Out

Estos bloques poseen la misma lógica que el System Out de área, ya que reciben el valid y dice que nonce es apto para un target mayor al valor de salida de h.

VI-G. System Out Principal

Este sistema toma las salidas de los 3 nonce como entradas, verifica que cuando un nonce es valido de cualquiera de los 3 system Out y le envía la señal a la entrada, haciendo que pase una nueva entrada y se reinicien todos los datos y señales.

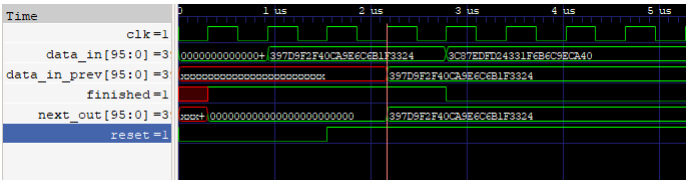

Figura 4: Next Desempeño. Creación propia

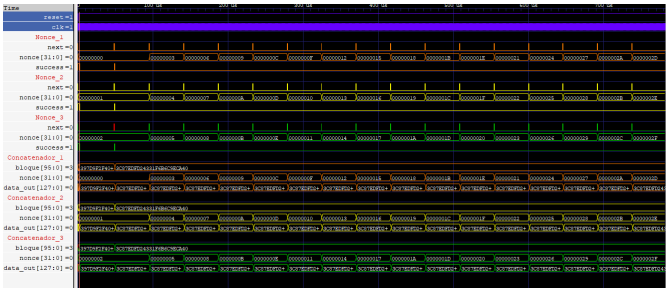

Figura 5: Nonce y Concatenador. Creación propia

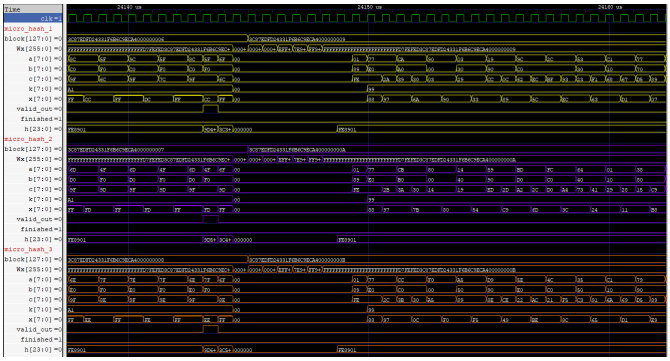

Figura 6: Micro hash. Creación propia

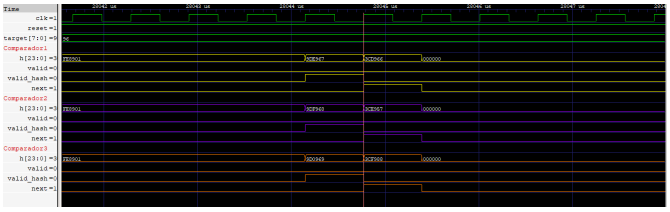

Figura 7: Comparador. Creación propia

VI-H. System Out

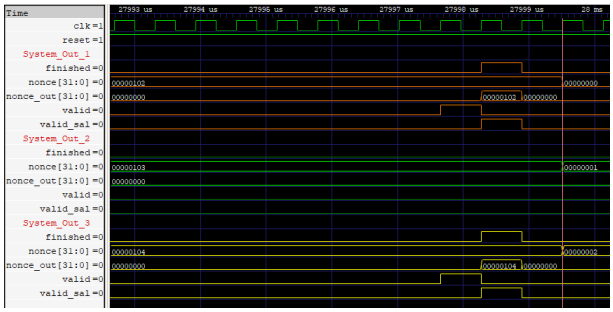

Figura 8: System Out. Creación propia

VI-I. System Out principal

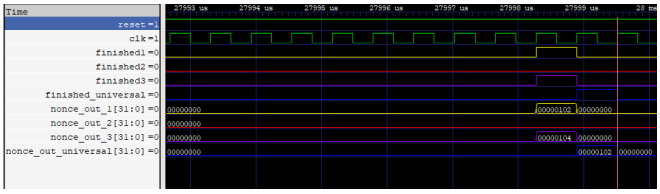

Figura 9: System Out principal. Creación propia

VI-J. Metricas y desempeño del circuito

Componente	Cantidad
Celdas Combinacionales	13211
Celdas Secuenciales	1989
Buffers e Inversores	1880
Total de Cables	13978
Total de Celdas	17080

Tabla II: Datos para optimización en rendimiento. Creación propia.

Igual que en la sección anteriror mediante la etapa de placement se obtuvo el dato del área total del circuito:

$$\boxed{\text{Area = 7,17 mm}}$$

Con la etapa de Static Timimg Analysis se obtuvo la frecuencia máxima de operación para el circuito:

$$\boxed{\text{f}_{max} = 151,045MHz}$$

Para obtener el tiempo de funcionamiento de la arquitectura,se realizan los mismo cálculos de la sección anterior. Con respecto a su cantidad de ciclos posee:

$$\boxed{\text{Cantidad}_{ciclos} = 74}$$

Con respecto a su tiempo mínimo es de:

$$\boxed{\text{T}_{minimo} = \frac{1}{f_{max}} = 6,62ns}$$

Con esto se puede determinar el tiempo de calculo:

$$\boxed{\text{Tiempo}_{calculo} = 74 * T_{min} * 1000 = 489,92\mu s}$$

Sin embargo se sabe que el circuito posee 3 hash en paralelo con lo cual el tiempo de calculo total es el tiempo de calculo dividido entre 3, ya que la cantidad de ciclos se reducirá en un factor de 3.

$$\boxed{\text{Tiempo}_{calculo} = (489,9202\mu s) * \frac{1}{3} = 163,3067\mu s}$$

VII. Métricas y resultados globales para ambas arquitecturas

A continuación se presenta un cuadro comparativo con la cantidad de celdas que posee cada arquitectura:

Componente	Opt. en Área	Opt. en Desempeño
Celdas Combinacionales	4597	13211
Celdas Secuenciales	780	1989
Buffers e Inversores	662	1880
Total de Cables	4755	13978
Total de Celdas	6039	17080

Tabla III: Comparación cantidad de celdas para ambas arquitecturas. Creación propia.

Podemos notar un resultado esperado en la comparación de la cantidad de celdas para cada arquitectura, como era de esperar la arquitectura que se basa en rendimiento y posee 3 sistemas en paralelo, consume una mayor cantidad de celdas con respecto a la arquitectura optimizada en área la cual posee hasta 3 veces menor cantidad de celdas y es por esto que si comparamos el área de ambos diseños se obtiene la siguiente relación:

$$Area_{opt_{rendimiento}} = 2,74 * Area_{opt_{area}}$$

Con respecto a la frecuencia de operación, se puede ver que entre ambos existe solamente una diferencia de 4MHz, esto es esperado ya que la arquitectura basada en rendimiento posee una mayor frecuencia de trabajo, sin

embargo siguen teniendo frecuencias similares. El cambio más importante se da en el tiempo de cálculo ya que podemos notar que en el tiempo que la optimización en área calcula una salida para un nonce específico, la optimización lo hace con 3 nonce. Lo anterior genera una disminución en el tiempo de ejecución para obtener la salida esperada, esto nos da como resultado una arquitectura más rápida siempre teniendo en cuenta que para esta se consume más área lo que va de la mano con un precio más alto en el mercado y mayores consumos de potencia.

VIII. Conclusiones

El tipo de arquitectura a escoger depende de la aplicación, presupuesto y objetivos que se deseen cubrir, ya que estos aspectos son muy importantes a la hora de escoger algún diseño ya sea enfocado a área, rendimiento, potencia, etc. Si se requiere un sistema que no se enfoque en la velocidad sino en el poder realizar la ejecución efectiva de este con la cantidad mínima de área, por los datos obtenidos, el sistema enfocado en área es el adecuado. Por otra parte, si se requiere un dispositivo con una salida de datos más rápida sin tomar en cuenta el área, el sistema de rendimiento es el que posee un mejor desempeño. Gracias a la síntesis, implementación y análisis de tiempo estático provistos por Qflow, se obtuvieron obtener métricas realistas sobre ambas arquitecturas implementadas.

IX. Código fuente

El código fuente relacionado al presente proyecto se encuentra en el repositorio: **https://github.com/jose-543/Proyecto2-Microelectronica.git**, este mismo repositorio posee un archivo README el cual presenta las instrucciones para compilar y ejecutar el código así como los archivos de tipo **log** generados por la herramienta **Qflow** en donde se pueden encontrar más detalles sobre las distintas etapas del flujo de diseño.