Proyecto Parte 2: RTL de un sistema de generación y comprobación de hashes

Estudiante: Juan Pablo Elizondo Calvo B52485
Profesor: Oscar Villalta
Microelectrónica IE-0411
Escuela de Ingeniería Eléctrica, Universidad de Costa Rica

***Resumen*—En este documento se mostrará el diseño y de desarrollo de un sistema que genera y comprueba hashes. Para esto se proponen 2 arquitecturas, en una de ellas se prioriza un área mínima y en el otro un máxima velocidad de procesamiento.**

I. Arquitectura de área mínima

I-A. Generalidades del sistema

Lo primero fue empezar creando una memoria donde se encuentran las entradas, aunque estas bien pueden ser generadas desde el testbench, la idea de esto es que el circuito puede leer entradas tanto del testbench como de una memoria física. Así se creo una memoria ROM (aunque en los archivos se llama RAM) que puede almacenar 4 registros de 96 bits, a los cuales luego se les concatenará el nonce para formar las 16 palabras de bits cada una. Esta memoria de 4 registros se creo de esta manera porque el circuito fue diseñado para leer 4 entradas distintas con su nonce y hallarle solución al hash de cada una. Aunque el cicuito tiene esta capacidad en las pruebas se decidió por encontrar solo una solución para una entrada, esto porque si no las pruebas serían demasiado largas, sin embargo esta opción puede habilitarse desde el driver, para esto la variable *num_entrada* tomaría el valor de 0 para una entrada, 1 para 2 entradas y así hasta 4 entradas, desde el driver puede además modificarse el target.

I-B. Generador nonce

Este módulo genera un nonce "aleatorio" cada ciclo de reloj. Para lograr la sensación de aleatoriedad simplemente se programaron 2 registros con números de 32 bits, luego a uno de ellos se le sumo su valor anterior más el valor del otro registro. El nonce generado entraba a la etapa llamada concatenador.

I-C. Concatenador

Esta etapa toma la entrada proveniente de la memoria ROM y la guarda en los bits superiores de un registro de 128 bits llamado *bloque_in*, luego en la parte baja de bloque_in se guarda el nonce generado en el módulo anteriormente mencionado. Este bloque es el que entra al módulo encargado de generar el hash.

I-D. Micro_ucr_hash

Este es el módulo principal y el más complejo del sistema, en este módulo hay una memoria llamada W que almacena bloque_in, luego se hacen todas las operaciones lógicas descritas en las especificaciones del proyecto durante 32 ciclos de reloj para generar un hash. Para controlar todo el flujo en este módulo se creo un contador de 6 bits, esto porque si bien 5 bits cubren las 32 iteraciones, tomaba un ciclo de reloj extra para almacenar bloque_in e inicializar otras variables, por lo que esta decisión fue muy conveniente, además se recibe un bloque_in únicamente cuando el contador es cero, esto ayuda a mejorar el algoritmo de randomización del nonce ya que se toma cada 33 ciclos de reloj aunque este cambie cada ciclo. El hash generado cada 33 ciclos (32 iteraciones más el ciclo de reset) es enviado a la etapa de comparación, además se envia el nonce que acompaña a este hash para que si se determina que es válido este valor no se pierda.

I-E. Comparador

Este etapa recibe el hash y el target y los compara, si se cumple la condición (los 2 primeros bytes del hash son menores que el target) el hash se toma como válido, lo que pone en uno la variable *valid* y hace que aumente el puntero en la memoria para leer la siguiente entrada o si ya no hay más entradas pone en uno la variable *fin* que está en el módulo *salida*, la cual siempre se mantiene en cero y está conectada a todos los módulos del sistema. Si el valid es 1 se deja pasar el nonce válido y su hash a la etapa de salida.

I-F. Salida

En el avance anterior se pensó en una estapa llamada Next que controlara cuando hay una próxima entrada, aquí esta función fue asimilada por los módulos Comparador y Salida por lo que no hizo falta implementarla en un módulo aparte, de hecho, para el RTL se hicieron algunas modificaciones a lo expuesto cuando se trabajo a alto nivel, sin embargo fueron necesarias dada la naturaleza del circuito. Volviendo al módulo salida, este controla la variable fin muestra las salidas del sistema, como el nonce válido y su hash.

II. Optimizando el área

II-A. Pruebas y síntesis

Las únicas señales generadas en el driver son el target, cantidad de entradas, el reloj y la señal de reset, las cuales

cambian al inicio de la simulación por lo que el resto es automático. Para la síntesis se utilizo qflow y la tecnología osu018, hubo varios inconvenientes en este punto para al final se pudo resolverlos:

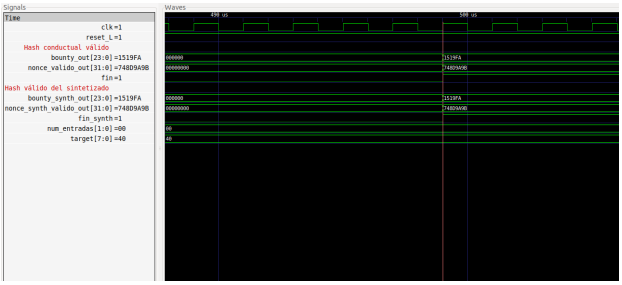

Figura 1. GTKw para la arquitectura de area minima

II-B. Métricas solicitadas

Se tienen las siguientes características obtenidas gracias al comando sta, al synth.log y a magic con el comando measure.

- Frecuencia máxima de operación: 377.49 MHz
- Número de celdas combinacionales: 3527
- Número de celdas secuenciales: 680
- Número de buffers: 297
- Número de inversores: 378
- Area: 0,17 um^2
- Tiempo para 1000 ejecuciones de la función hash: 87 us

III. Optimizando el desempeño

Para este caso se instanció un nuevo módulo hash, además el contador que estaba dentro de este ahora es un módulo idependiente. La idea es que el contador entre al módulo hash_1 y actue con normalidad, a su vez habrá otro módulo hash_2 que recibe el mismo reloj pero atrasado 2 ciclos de reloj, lo cual se logra con otro módulo de contador que recibe la salida del original y la retrasa. Habiamos dicho anteriormente que el modulo generador de nonces los genera cada ciclo de reloj, independientemente de si se usan o no, por lo tanto el módulo concatenador fue modificado para generar 2 salidas, cada una con la misma entrada en la parte alta pero con diferente nonce en la parte baja, así pues se tiene a los 2 módulos de hashes trabajando con entradas iguales iguales pero nonces diferentes, por lo que ahora el sistema es el doble de eficiente. Siguiendo este mismo sistema podríamos agregar tantos módulos de hash como se quisieran. Luego las salidas de los módulos hash van a un mux que las redirige a la etapa de comparación para ser comprobadas en diferentes instantes, con lo que no hubo necesidad de crear nuevos bloques de comparación, solo de hashes.

III-A. Pruebas

Aunque el circuito es en teoría el doble de eficiente, el tiempo que tardó en resolver una entrada fue curiosamente mayor que en el diseño anterior, sin embargo una sola prueba no es para nada concluyente y es resultado de la aleatoriedad del nonce, se podría decir que la suerte es un factor.

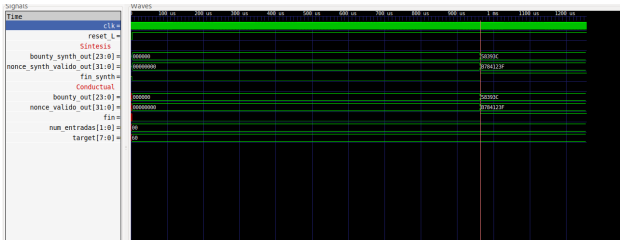

Figura 2. GTKw para la arquitectura de velocidad

III-B. Métricas

- Frecuencia máxima de operación: 356.23 MHz
- Número de celdas combinacionales: 6239
- Número de celdas secuenciales: 1230
- Número de buffers: 546
- Número de inversores: 79
- Area: 0,3 um^2
- Tiempo para 1000 ejecuciones de la función hash: 46 us

Para este caso el cálculo se dificulta debido a la paralelización (incluso dice que el circuito opera a una frecuencia menor pero hay que tomar esto encuenta), así que se hacen cálculo análogos y luego se divide entre 2.

Referencias