

**Multiplicador secuencial.**

**Nombre:**

Jorge Mizael Rodríguez Gutiérrez

Luis Roberto Lomelí Plascencia

**Expediente:**

IE698323

IE700093

**Materia:**

Diseño y verificación

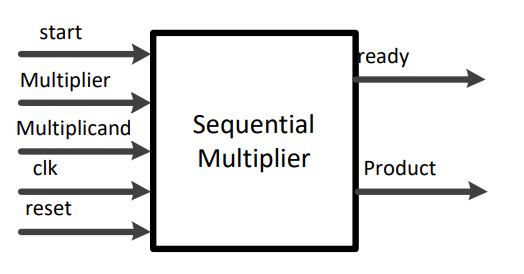
**Fecha:**

13/Marzo/2019



**Planteamiento del problema**

Realizar el diseño, verificación e implementación de un multiplicador secuencial (MS), el cual tiene la siguiente interfaz.

****

**Descripción funcional**

El MS calcula la multiplicación entre dos numero enteros a complemento a 2. El MS es un módulo secuencial que requiere N ciclos para el procesamiento del resultado, donde N es número de bits de los datos de entrada. El procesamiento inicia cuando Start = 1, y termina cuando ready = 1. El resultado, Product, se entregará a 2N bits.

**Restricciones**

1. El control se debe realizar con una máquina de estados que utilice tipos enumerados.
2. Se debe implementar la arquitectura que se realizó en clase, pero se pueden agregar optimizaciones al diseño.
3. El MS debe ser capaz de realizar múltiples cálculos cada vez que Start =1.
4. No se puede presionar el botón de reset entre peticiones de cálculo.
5. Se debe utilizar un archivo de macroinstrucciones en la simulación de modelsim donde se muestren las señales internas al módulo, en particular el acumulador del producto.
6. Se deben seguir todas las prácticas y recomendaciones de diseños vistas en clase.
7. La implementación tiene que hacer uso de un PLL para la generación de reloj, junto con el PLL se puede usar el divisor de frecuencia implementado en clase.

**Entregables**

**Un reporte que contenga:**

* Descripción de la arquitectura propuesta, la cual debe incluir la descripción funcional de cada módulo que compone el diseño.
* Simulación en modelsim.
* Resultados de síntesis en términos de logic elements (LEs), y frecuencia máxima de operación del diseño, sin incluir.
* Realizar una comparativa de LE y frecuencia de operación entre el MS puramente comportamental y el desarrollado en esta práctica. Para esta comparación se tiene que omitir el PLL y Signal Tap, es decir, solo se debe tomar en cuenta la implementación de ambas versiones de multiplicador.
* Capturas de señales internas con el siganalTap, en particular el valor del acumulador interno.
* Implementación en FPGA. Archivos de implementación con SystemVerilog.
* Todas las fuentes deben ser comentadas
* Repositorio privado en git hub.
* Test bench.

**Evaluación**

**Estos son los elementos por evaluar y sus respectivos pesos:**

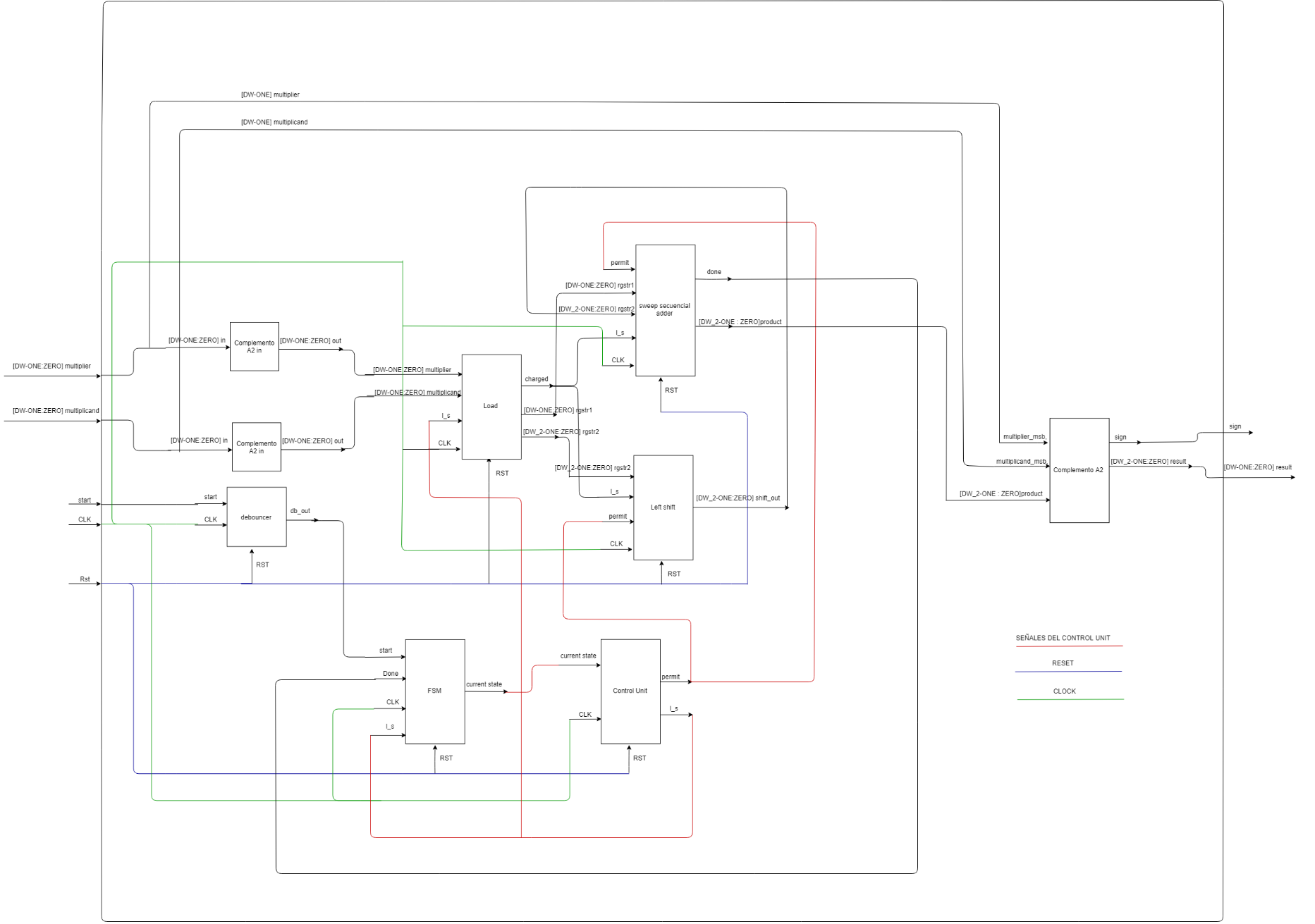
1. Funcionalidad. Se considera que el MS funciona solo si se compila sin errores en Quartus y Modelsim. Si la descripción no se puede compilar, o la simulación no termina o se aborta, se obtendrán cero puntos en este apartado.
2. Reporte. Se evaluará claridad y sencillez en la redacción, ortografía, que se presenten las decisiones clave que tuvieron que tomar para resolver el problema, que contenga la información solicitada y que los datos de los integrantes estén completos y sean correctos.
3. Preguntas. Durante la revisión de la práctica, el profesor hará varias peguntas a ambos integrantes del equipo. Cada pregunta va dirigida a un integrante en particular (el otro integrante no puede intervenir). Las preguntas son sobre la práctica y sobre los conceptos necesarios para realizarla. Se cuestionarán también las razones por las que decidió implementar una solución específica. Ambos integrantes deberán estar enterados de todos los aspectos de la práctica. Los puntos obtenidos dependerán de la proporción de preguntas contestadas correctamente. La nota obtenida en este apartado es individual.

**Recomendaciones:**

Para la simulación se puede omitir el circuito generador de frecuencia.

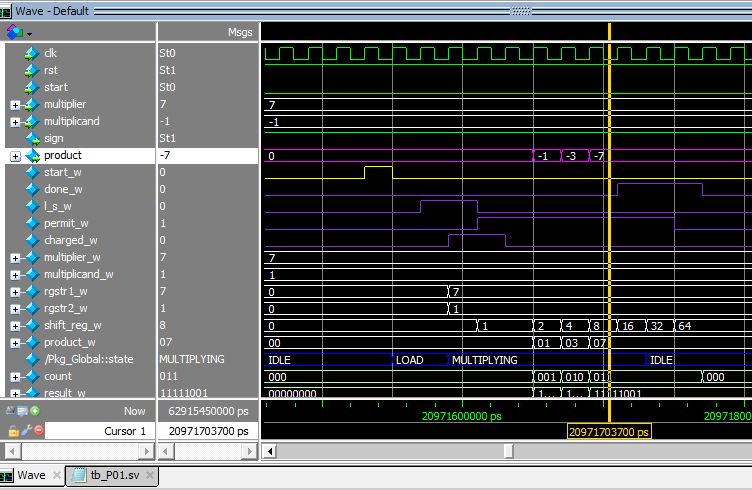
**DESCRIPCIÓN FUNCIONAL**

|  |  |
| --- | --- |
| Modulo | Descripción |
| LOAD | Este módulo se encarga de leer los factores de entrada y almacenarlos en un registro.  Al término de esta operación se indica con la señal de “charge” que los valores están listos para ser utilizados. |
| LEFT SHIFT | Una vez cargados los valores de las entradas, el algoritmo indica que uno de los factores ya sea el multiplicando o el multiplicador debe ser sumado al producto y se le debe hacer un corrimiento a la izquierda al factor elegido. En este módulo solo se produce el corrimiento necesario y este se aplica N veces donde “N” es el número de bits que mide cualquiera de las entradas. |
| SWEEP ADDER | Este módulo se encarga de sumar la entrada de uno de los factores con el producto. Cuando la entrada esta lista, se incrementa un contador en cada ciclo de reloj.  El multiplicando se debe sumar al producto solo si el multiplicador en la posición factor[contador] == 1’b1;  Por otro lado, si el contador cuenta hasta la cantidad del tamaño de bits de las entradas (los factores) entonces se manda una señal de “done” que indica que el resultado está listo. |
| A2 | Los parámetros de entrada al módulo complemento a2 son los bits más significativos de cada factor con ellos conocemos el signo de cada entrada y podemos determinar si el resultado de la operación será positivo o negativo.  El resultado de la operación xor define el signo del producto. Y define si aplicar la inversión de cada y sumar 1 a la salida. |
| FSM | Dentro de 3 posibles estados en los que puede estar el multiplicador secuencial este módulo coordina las operaciones de los módulos necesarios para la multiplicación como lo son; “load”, “sweep adder”, “left shift” y “a2”.  IDLE: Mantente preparado para la señal de inicio.  LOAD: Carga los valores de la entrada.  MULTIPLYING: Realiza la multiplicación. |
| CONTROL | Este módulo recibe retroalimentación con señales de estado (status) por parte de la máquina de estados o (FSM) y es también quien manda las señales de que rigen el comportamiento de la Finite State Machine. |

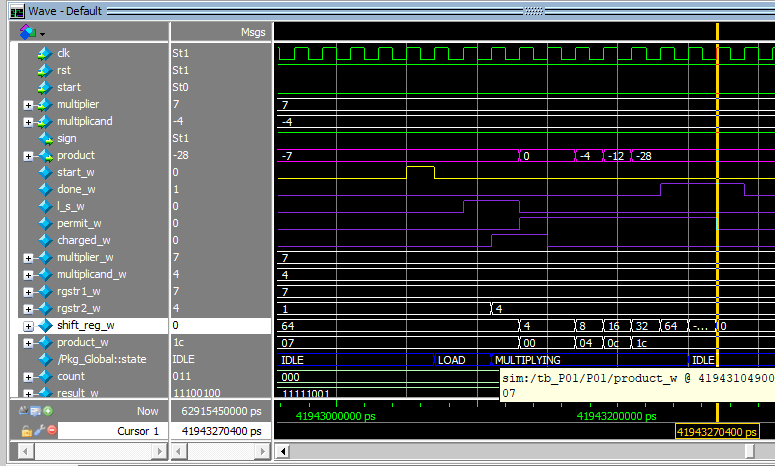
**Diagrama RTL de nuestro diseño**

**Simulación en Modelsim**

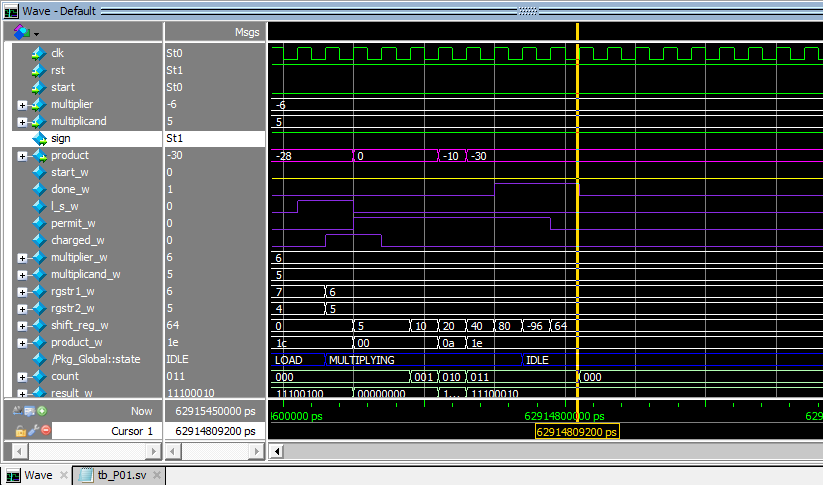
En las siguientes imágenes podemos observar el comportamiento de nuestra practica ante diferentes entradas, en la figura 1, 2, y 3 podemos ver la simulación mediante modelsim



**Figura 1:** simulación de modelsim ante entradas de 7 y -1



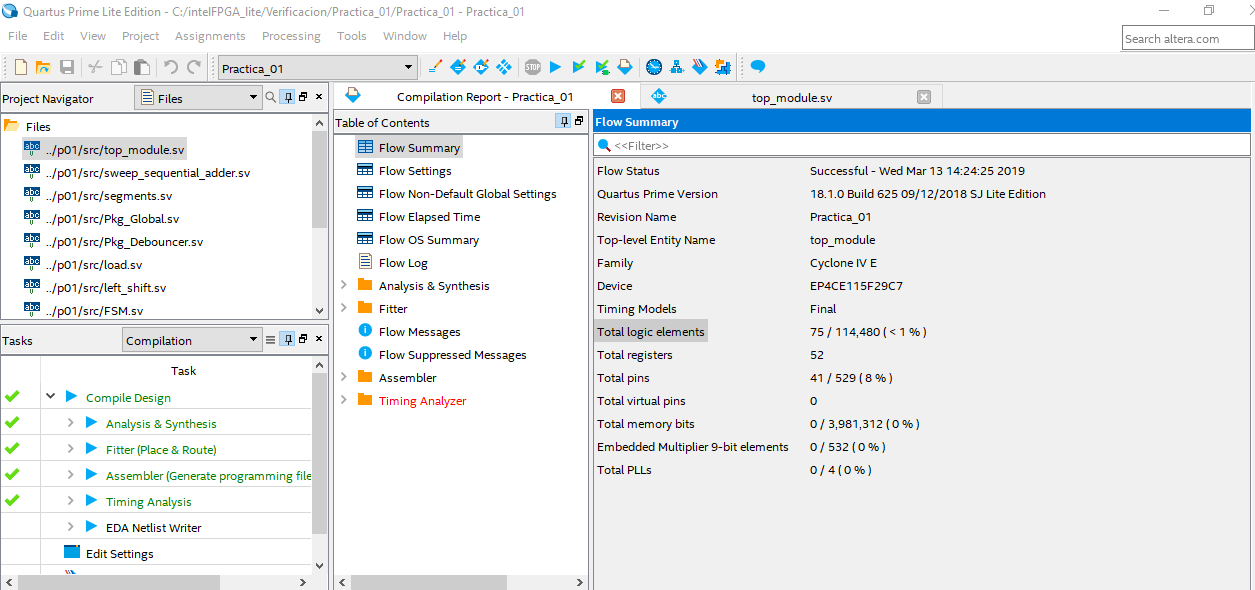
**Figura 2:** simulación de modelsim ante entradas de 7 y -4



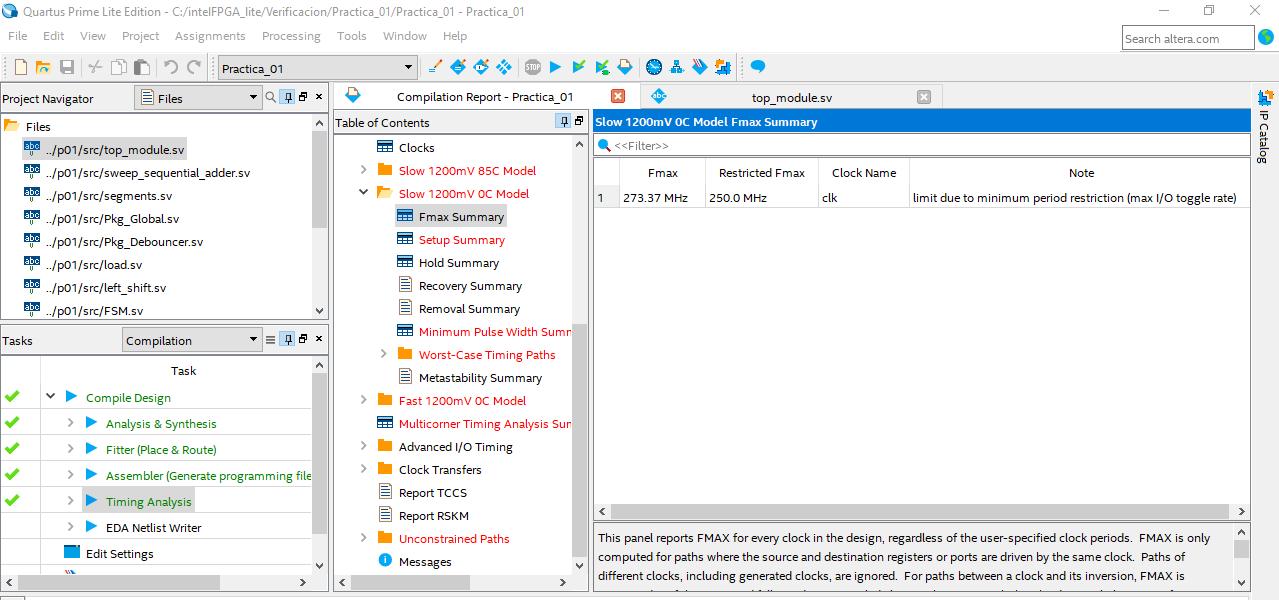
**Figura 3:** simulación de modelsim ante entradas de -6 y 5

**Descripción de las imágenes:** En la imagen si analizamos de manera cuidadosa podemos ver la transición de estados en el cual al entrar al estado multiplying vemos la transición que hay en la señal de producto que es la importante ya que esa es la que almacena el resultado para sumarlo con el siguiente.

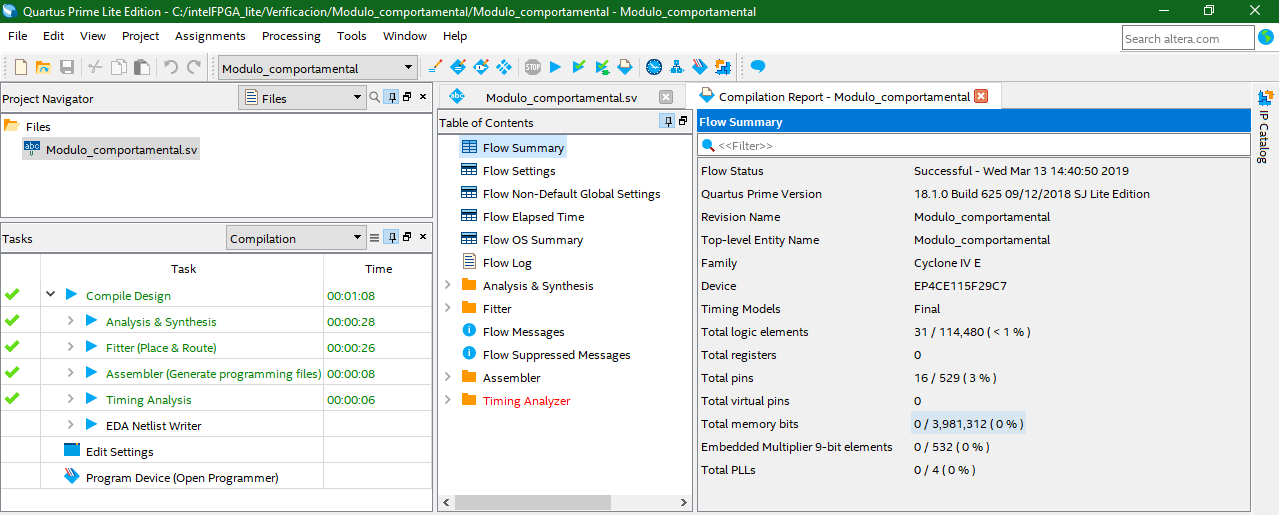
Cantidad de logic elements

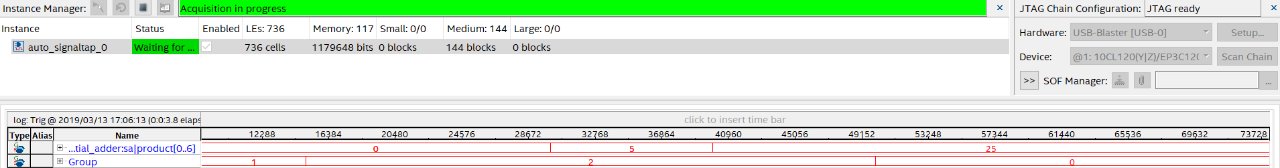


Frecuencia máxima de operación



Cantidad de logic elements del multiplicador comportamental



Capturas del signal tap

**Conclusiones**

**Luis Roberto:**

Una de las partes que me tomó más tiempo y más trabajo fue la manera de depurar el diseño del multiplicador secuencial, en este caso me costó entender que es mucho más fácil analizar cómo se comporta un sistema mediante las señales que encienden y/o apagan en una simulación, que tratar de encontrar el error viendo el código de System Verilog.

Ahora poco antes de entender como depurar por medio de la simulación en ModelSim me vi limitado a estar diseñando en papel algo que fuese lo más cercano a un módulo funcional del multiplicador, pero esto también tiene sus desventajas, como por ejemplo antes que nada cuando resuelves manualmente el problema que estas diseñando, identificas operaciones que puedes ir dividiendo en bloques de operación individuales o podrías también diseñar un bloque con la solución entera.

La primera solución que encontramos estaba limitada a un solo bloque que resolvía todo el problema, sin embargo, el requisito de la separación de módulos nos metió en la necesidad de dividir nuestra solución que ya era funcional, y me parece que esto es un poco más difícil que si diseñas la solución completa por bloques desde un inicio.

Esto concluye en que el ejercicio que hicimos tuvo un nivel mayor de dificultad por la división de un bloque en pequeños submódulos, y lo cual está altamente ligado a nuestra más grande complicación en el diseño del multiplicador que fue sincronizar las señales de control y de estatus.

**Jorge Mizael**

Cosas que tuvieron relevancia para mi en esta practica fue la manera de diseñar, ya que en un inicio buscamos un algoritmo que cumpliese con las multiplicaciones en paralelo, ya que logramos comprender el funcionamiento del algoritmo, diseñamos nuestro diagrama de bloques para mostrarle al maestro como era el funcionamiento de nuestro algoritmo.

Después diseñamos la practica en un módulo, como siguiente paso pasamos a dividirlo en bloques, entiendo que el objetivo de pasarlo a módulos más pequeños y paramétricos tiene la finalidad de reutilizarlos, pero desde mi punto de vista es más sencillo realizar la practica en un solo modulo, ya que encontrar errores siento que es más sencillo por otro lado el representarlo en módulos mas pequeños complica las cosas, ya que tienes que considerar mas la sincronía de la señales con más atención, tanto es así que para encontrar errores en la práctica es más sencillo usar modelsim para ver el comportamiento de las señales, analizar como quieres que se comporta y que hace, para rastrear de manera mas sencilla que es lo que se tiene que modificar en código de verilog.

URL

Diagrama de bloques

<https://drive.google.com/file/d/1GLv3T_kstHNf18boYKFuaCI5jJeBnRpM/view?usp=sharing>

nota: no se ve muy bien la imagen en el link, pero en su correo le mande la invitación para revisar mas a fondo el diagrama de bloques.

github

<https://github.com/lrolomeli/verificacion_2019_mizael_luis>