Diseño de Sistemas Digitales

Práctica 2: Diseño de un Multiplicador Secuencial

Realizar el diseño, verificación e implementación de un multiplicador secuencial paramétrico de *N* bit por *N* bits.

**Especificaciones**:

1. La interfaz del multiplicador debe ser como se muestra en la Fig. 1



Fig. 1 Caja negra del bloque multiplicador secuencial.

1. El Sequential\_Multiplier calcula la multiplicación entre dos números enteros en sistema numérico complemento a 2 de *N-*bits (*Multiplier* X *Multiplicand*).
2. El algoritmo de multiplicación debe ser secuencial. Una señal de reloj debe marcar el ritmo del algoritmo de multiplicación: al menos se deben requerir *N* ciclos de reloj para obtener el resultado esperado (*Product*).
3. El tamaño de la salida *Product* debe ser de 2N bits.
4. El procesamiento debe iniciar cuando la señal de s*tart* = 1.
5. Cuando el resultado *Product* esté completo el puerto de salida *ready* debe ser activado. La siguiente Fig. 2 muestra un ejemplo de simulación de una multiplicación.

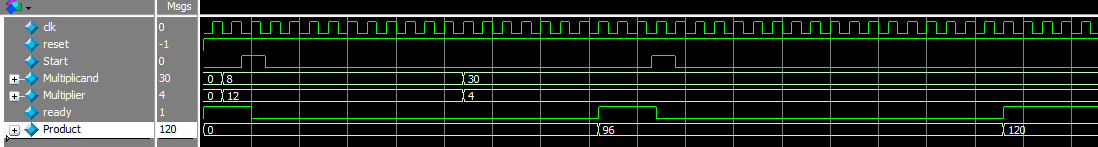


Fig. 2 Formas de onda de la simulación del multiplicador.

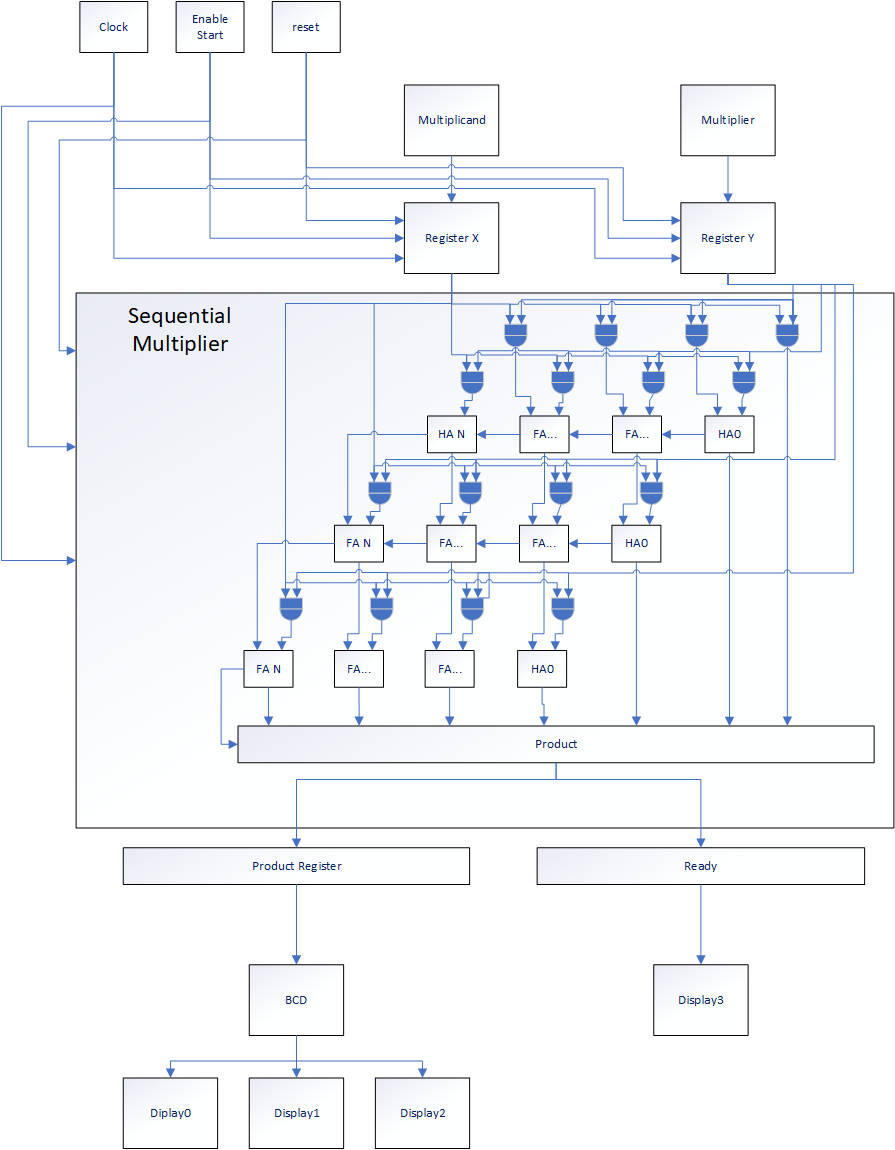
1. El diseño de mayor jerarquía del multiplicador secuencial debe estar modelado con Verilog a nivel estructural: debe instanciar los sub-modulos, los cuales deben ser bloques de uso frecuente claramente identificables: registros, contadores, registros de corrimiento, multiplexores, sumadores, decodificadores máquinas de estado, etc.
2. La verificación del multiplicador se realiza en el simulador considerando N = 8.
3. Para la implementación en la tarjeta DE10-Standard limitar el tamaño de las entradas a 5 bits e incluir un decodificador de binario a BCD para mostrar los resultados en el display en decimal signado.
4. En la implementación, no se deben mostrar en display los productos intermedios o productos parciales. El resultado solo debe actualizarse después de presionar *start* y se haya generado la señal de *ready*.

**Entregables**

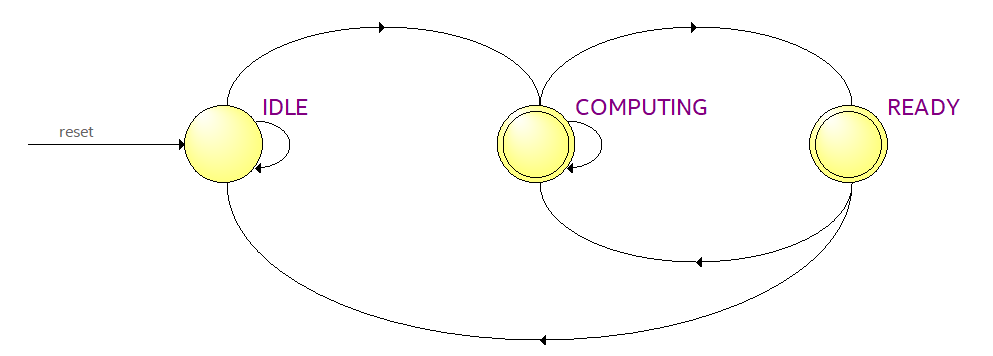
1. El proyecto comprimido del diseño top generado con la herramienta Quartus, subido en el link correspondiente.
2. Reporte subido en Canvas en la fecha de entrega definida. Usa este mismo documento para entregar tu reporte. El reporte debe contener los siguientes puntos.
3. Explicación del algoritmo de multiplicación secuencial que usaste y planteamiento de la solución al diseño [10%].

El método o algoritmo que se esta implementando para el desarrollo del multiplicador secuencial es el método “shift and add method” Si cualquier bit en el multiplicador (b) es 0, entonces el multiplicando (a) se suma con cero. Se utiliza un sumador que tiene la misma longitud que los operandos. La salida del sumador y el multiplicador se aumenta en un banco de registro. Después de cada adición, el contenido del banco de registros se desplaza a la derecha.

1. Diagrama de bloques (arquitectura) del diseño de mayor jerarquía y diagrama de estados de la FSM para controlar los módulos del multiplicador sequencial [15%].

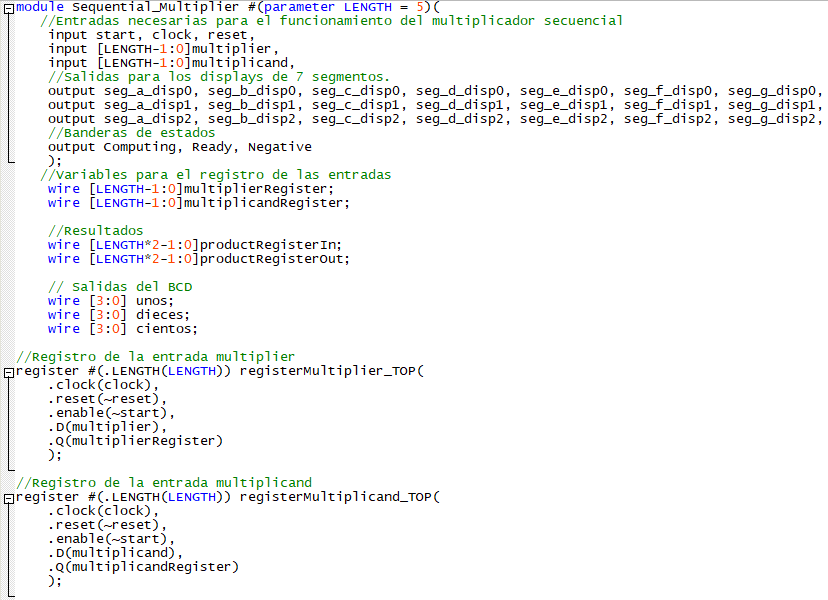
Arquitectura: 

Diaframa de Maquina de estados:



1. Modelos Verilog comentados, de cada uno de los bloques del sistema y del diseño de mayor jerarquía [25%].

Modulo TOP, Sequential\_Multiplier



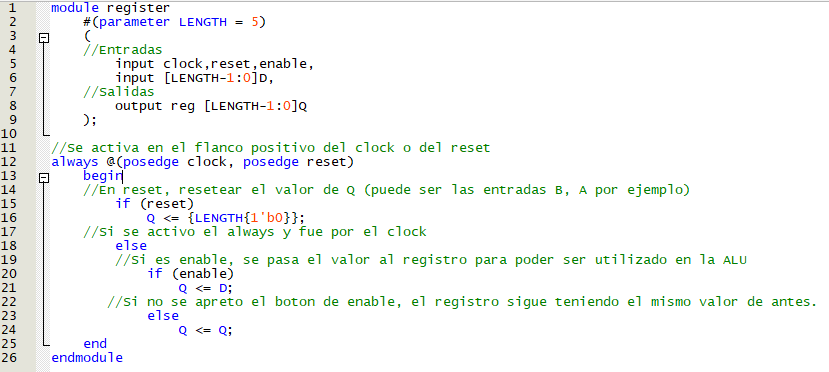
Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Modulo register:



Modulo FSM\_SecuentialMultiplier:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Modulo BinaryToBCD:

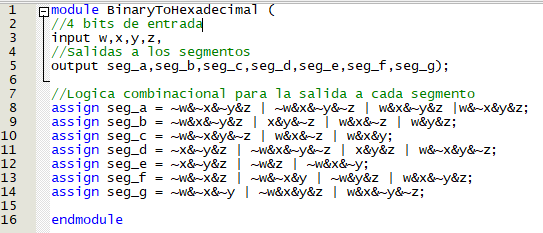
Texto

Descripción generada automáticamente

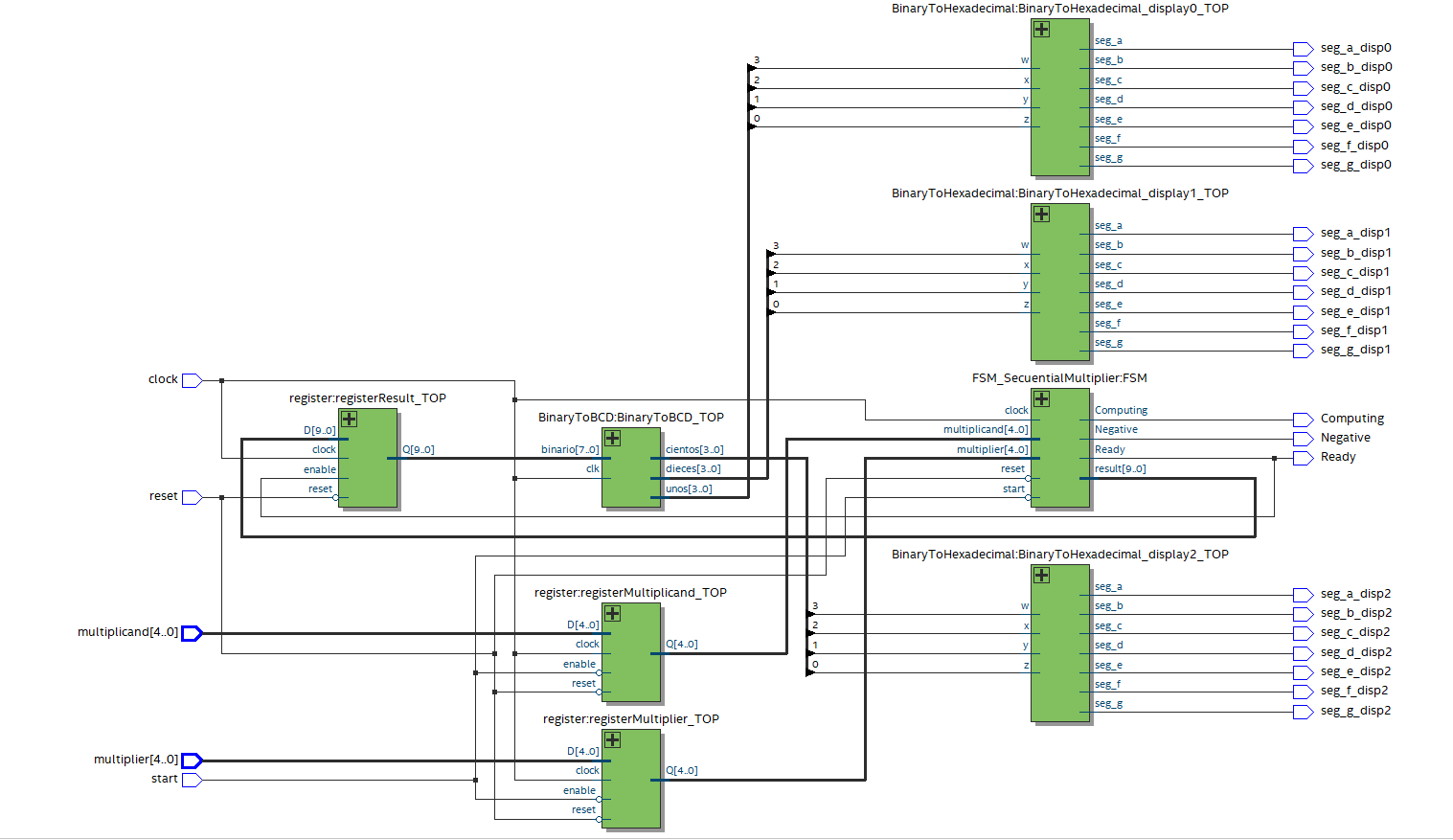
Texto

Descripción generada automáticamente

Modulo BinaryToHexadecimal:

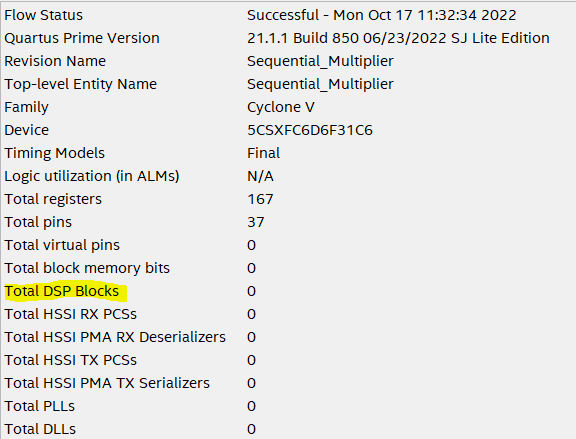


1. Diagrama esquemático que genera la herramienta Quartus Prime (Tools > Netlist Viewers > RTL Viewer) [5%].

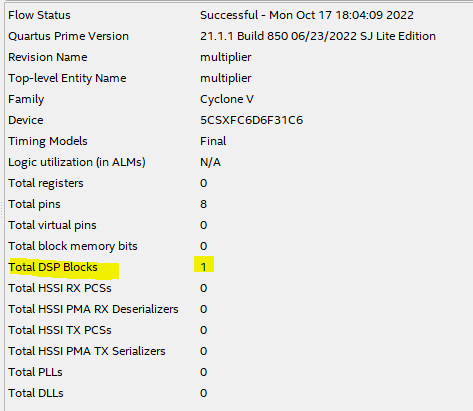


1. Reporte de recursos utilizados por el Sequential\_Multiplier para *N* = 8 bits, y su comparación con una versión de un multiplicador modelado directamente con el operador \* (*assign Product = Multiplicand \* Multiplier*) implementado en LUTs, no en bloques DSP de multiplicación. Comentar los resultados de los recursos utilizados [5%].

Recursos utilizados para el “Sequential\_Multiplier” donde N=8:



Recursos utilizados para un multiplicador modelado directamente de N=8



Como previamente se comento en clase, se cuenta con un numero limitado de DSPs para usa sola multiplicación donde N=8 vemos como solo se usa un DSP y en cambio en el “Secuential\_Multiplier” vemos como ningún DSP esta siendo utilizado

1. El *test-bench* y formas de onda de la simulación del *Sequential\_Multiplier*, solamente el *Sequential\_Multiplier*, sin display ni decodificador a decimal. Donde se muestren los siguientes casos: 1 x -1, -1 x -1, 15 x -16, 15 x 15, -16 x -16, -16 x 0. Incluye en esta simulación el procesamiento de señales intermedias de interés, por ejemplos, señales de control de la FSM, valores en los registros de corrimiento y productos intermedios en el acumulador [25%].

Tomar en cuenta que la bandera de Negativo está en lógica negativa para poder encender el signo de menos en el display de 7 segmentos, además de los botones de start y reset que también cuentan con lógica negativa.

Resultados para 1x-1:

Interfaz de usuario gráfica

Descripción generada automáticamente

Resultados para -1x-1:

Interfaz de usuario gráfica

Descripción generada automáticamente

Resultados para 15x-16:

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

Resultados para 15x15:

Interfaz de usuario gráfica

Descripción generada automáticamente

Resultados para -16x-16:

Interfaz de usuario gráfica

Descripción generada automáticamente

Resultados para -16x0:

Interfaz de usuario gráfica

Descripción generada automáticamente

1. Conclusiones y reflexiones sobre los aprendizajes [10%].

Al finalizar la practica unos de los principales a puntos a resaltar es el continuo aprendizaje en el lenguaje de modelado de HW Verilog y el desarrollo de los testbench

Otro punto muy importante a resaltar el concepto de FSM Maquinas de estados finitas, en esta práctica se introdujo este concepto al desarrollar una máquina de estados que estaría controlando el funcionamiento de nuestro multiplicador secuencial

Y finalmente comenzamos a aplicar optimización de código, el ejemplo esta en evitar usar el multiplicador que es una de las operaciones que mas demanda recursos e implementar en su lugar adders y corrimientos, el hecho si este es la mejor implementación que se puede lograr queda sujeta a la intención o finalidad de la aplicación que se busca desarrollar.

1. Referencias consultadas en formato estándar utilizando Zotero [5%].

[1] Stack Exchange. (2013, Agosto 28). Why are inferred latches bad?. [En linea]. Disponible en: <https://electronics.stackexchange.com/questions/38645/why-are-inferred-latches-bad>

[2] StackOverflow. (2014, Marzo 17). What is inferred latch and how it is created when it is missing else statement in if condition. Can anybody explain briefly?. [En linea]. Disponible en: <https://stackoverflow.com/questions/22459413/what-is-inferred-latch-and-how-it-is-created-when-it-is-missing-else-statement-i#:~:text=A%20latch%20is%20inferred%20within,sensitivity%20list%20and%20feedback%20loops>.

[3] StackOverflow. (2016, Abril 19). conditionally calling a module using case statement. [En linea]. Disponible en: <https://stackoverflow.com/questions/36661085/conditionally-calling-a-module-using-case-statement>

[4] ITESO Canvas. Finite State Machines. [En linea]. Disponible en: <https://iteso.instructure.com/courses/25896/files/4377462?module_item_id=1042875>

Presentación mostrando la implementación en la tarjeta DE10-Standard y un enlace a un video en algún *drive* donde se muestre el funcionamiento de la implementación para los casos indicados en el punto 6. Sin presentación no se toma en cuenta el reporte.

[Sequential Multiplier](https://drive.google.com/file/d/1S78CKckMEdU-JG4ziV-kkjY9i0V2_v7P/view?usp=sharing)