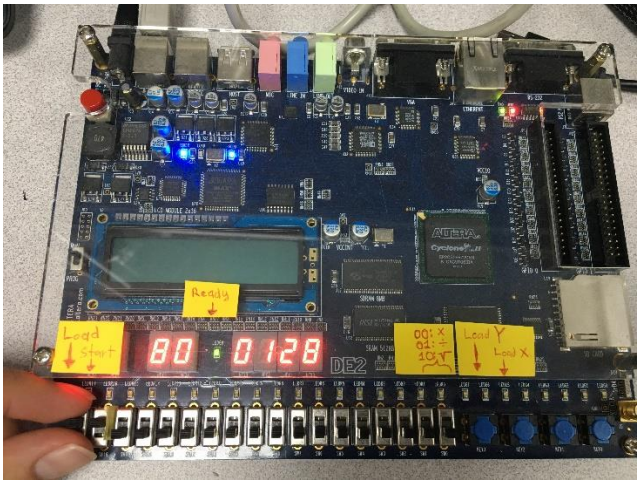
ITESO
Universidad Jesuita de Guadalajara

DISEÑO, VERIFICACIÓN Y VALIDACIÓN DE SISTEMAS DIGITALES

Otoño 2019

Práctica 2: Multiplicador, Divisor y Raíz Cuadrada



Profesor: Dr. José Luis Pizano Escalante

Alumno: Adrián Ramos Pérez

Expediente: IE692494

Table of Contents

DESCRIPCIÓN FUNCIONAL

Esta práctica tiene como objetivo el desarrollar un módulo aritmético que calcule la multiplicación, división y raíz cuadrada de un número entero con signo.

La interfaz del Multiplicador, Divisor y Raíz cuadrada (MDR) se muestra en la siguiente figura y sus señales se describen en la tabla 1:

Entradas	
Señal	Descripción
Data	Es el puerto de entrada de datos de 16 bits.
Start	Cuando esta señal es igual a 1 lógico, el modulo comienza a trabajar.
Load	Cuando es igual a 1 lógico, el valor que se encuentra en la señal ***Data*** se carga dentro del MDR.
Op	Selecciona la operación a realizar: 0: Multiplicación 1: División 2: Raíz cuadrada
clk	Señal de reloj.
reset	Señal de reinicio.
Salidas	
result	Este puerto entrega el resultado de la operación ejecutada, tiene un ancho de 16 bits.
ready	Cuando es igual a 1 lógico indica que el resultado en el puerto **Result** es válido.
Remainder	Es el puerto donde se muestra el residuo de la operaciones de división y raíz cuadrada.
Load X	Cuando es igual a 1 lógico indica que se debe colocar el valor de **X** a cargar en el puerto **Data**.
Load Y	Cuando es igual a 1 lógico indica que se debe colocar el valor de **Y** a cargar en el puerto **Data**.
error	Indica un error en el resultado.

El módulo por diseñar e implementar realiza la multiplicación, división o raíz cuadrada de dos números de 16 bits **X** y **Y** en forma secuencial, X y Y son números con signo a complemento a dos.

Los datos **X** y **Y** son cargados al MDR de la siguiente forma:

Cuando se presiona **Start**, tal que **Start** == 1, al siguiente pulso de reloj la salida **Loax X** debe ponerse en uno, indicando que se debe colocar **X** en el puerto **Data**, una vez colocado se presiona **Load**, tal que cuando **Load** == 1, se carga **X** en el MDR.

Después de cargar **X**, la señal de **Load Y** se debe poner en uno para indicar que se debe cargr el dato **Y**, seguido de esto se coloca el valor **Y** en el puerto **Data**, y se presiona **Load**. Una vez cargados los datos **X** y **Y**, se comienza el cálculo de la operación que indique el puerto **Op**. Durante el proceso de cálculo, los resultados parciales no deben mostrarse en el puerto **Result**, y cuando se alcance el valor resultante de **X** y **Y**, la señal **ready** debe ponerse en uno lógico.

Los resultados parciales deben mostrarse en 16 leds de la tarjeta DE2-115, además el resultado final se mostrará en los displays de 7 segmentos con un led que indique cuando el resultado es negativo, todo esto una vez que **ready** ==1.

Cuando la operación a ejecutar es raíz cuadrada, el único dato a cargar es el valor de **X**.

La bandera de error se activa cuando se detecte los siguientes casos:

- La multiplicación sobrepase los bits de salida, por ejemplo X=0xFFFF por Y=0xFFFF.
- División entre cero.
- Cálculo de la raíz cuadrada de un número negativo.

En el caso de existir sobre flujo la salida del MDR se debe saturar, es decir, Resultado = 0xFFFF.

RESTRICCIONES

1. Los diferentes algoritmos a implementar se deben hacer en forma secuencial.
2. El algoritmo de multiplicación a implementar debe ser el de Booth.
3. El algoritmo de división es a libre elección, como recomendación se puede usar el algoritmo de CORDIC.
4. El algoritmo de raíz cuadrada debe ser el expuesto en clase.
5. El diseño y la implementación solo puede contener un sumador/restador.
6. El diseño y la implementación no se puede hacer uso de multiplicadores.
7. Todas las entradas deben ser registradas.
8. El reloj del módulo debe ser al meos 5Mhz.
9. La implementación en FPGA debe hacer uso de un PLL para la generación de reloj, el cual se puede combinar con un divisor de contadores síncronos para obtener la frecuencia deseada.
10. El PLL y el divisor síncrono tiene que estar encapsulado en un módulo llamado generador de reloj.
11. El MDR debe ser capaz de procesar operaciones.
12. En los displays de 7 segmentos no se deben mostrar los resultados parciales, solo el final cuando ready == 1.
13. Todo aquello que no esté definido en la especificación de la práctica queda a libertad del diseñador.
14. La implementación tiene que hacer uso de un package para la definición de tipos de datos enumerados, tipos definidos. Debe hacer uso de interfaces y todos aquellos nuevos elementos del lenguaje de descripción de hardware SystemVerilog.

PROPUESTA Y DESCRIPCIÓN DEL DISEÑO (MICROARQUITECTURA MODULAR Y GENERAL)

Multiplicación.

Se describió una arquitectura modular para cada operación. La de la multiplicación queda con los 13 módulos como en la imagen que se muestra a continuación, junto con la descripción de cada módulo y para qué se utiliza:

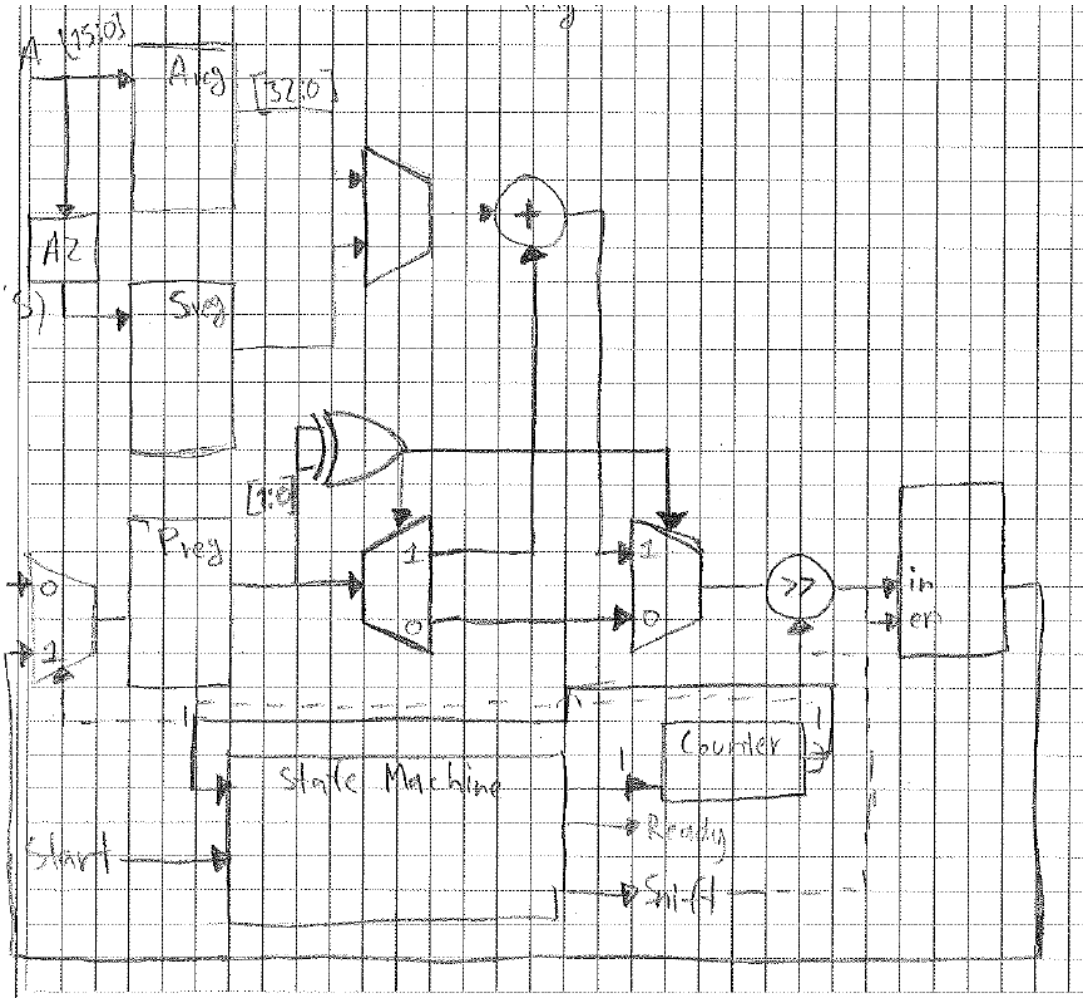


Ilustración 1. Microarquitectura del módulo de multiplicación.

- **4 PIPOs/Registros:** Para almacenar multiplicando, multiplicador y el complemento A2 para las operaciones según el algoritmo de Booth. (A, S y P).
- **1 Contador.** Para poder indicar el número de ciclos que se va a realizar la operación.
- **1 Shift Module.** Que permitan hacer un shift al registro P.
- **3 Multiplexores 2 a 1.** Cuyo selector será el bit menos significativo del multiplicador, y dejará pasar la correspondiente iteración del multiplicando o una cadena de ceros.

- **1 Demultiplexor.** Para decodificar multiplicando y multiplicador de 8 bits cada uno al correspondiente número binario positivo o negativo, y poder tomar el signo para el resultado de la multiplicación.
- **1 Sumador.** Para poder sumar lo que viene de la iteración correspondiente con lo que ya se ha acumulado hasta el momento.
- **1 XOR:** Para servir de control del selector del mux y demux en base a los dos bits menos significativos de P.
- **1 Máquina de estados Moore/Unidad de Control.** Para controlar el flujo de la operación, como indica el diagrama de estados que se muestra a continuación.

ALGORITMO DE BOOTH PARA MULTIPLICACIÓN:

1. Llenar A con el multiplicando, agregar mismo número de bits +1 de 0s. Llenas con complemento A2 de A e igual mismo número de bits +1 de 0s. Llenar P con n 0s en la parte alta y el multiplicador en parte baja. Agregar un 0.
2. Comparar los 2 bits menos significativos, si son iguales solo shift P >>1, si:
    - 10: P = P+S. P>>1.
    - 01: P = P+A. P>>1.
3. Repetir n ciclos.
4. Descartar el bit menos significativo.

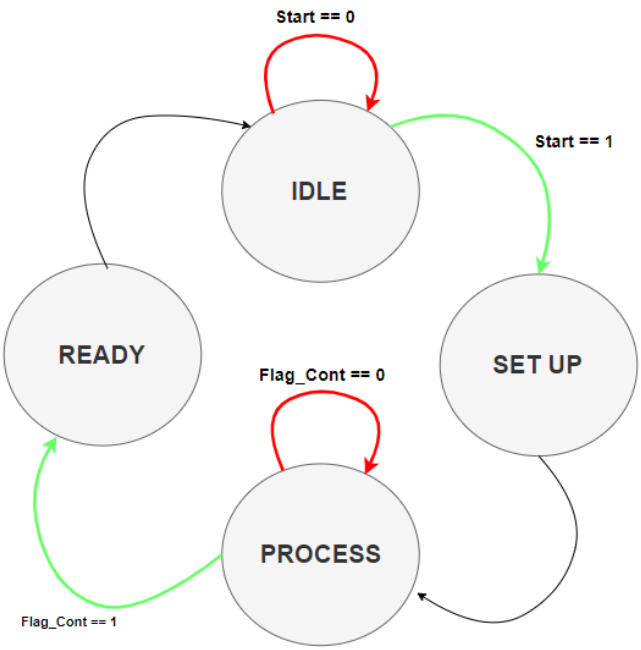


Ilustración 2. Máquina de estados del módulo de multiplicación.

División

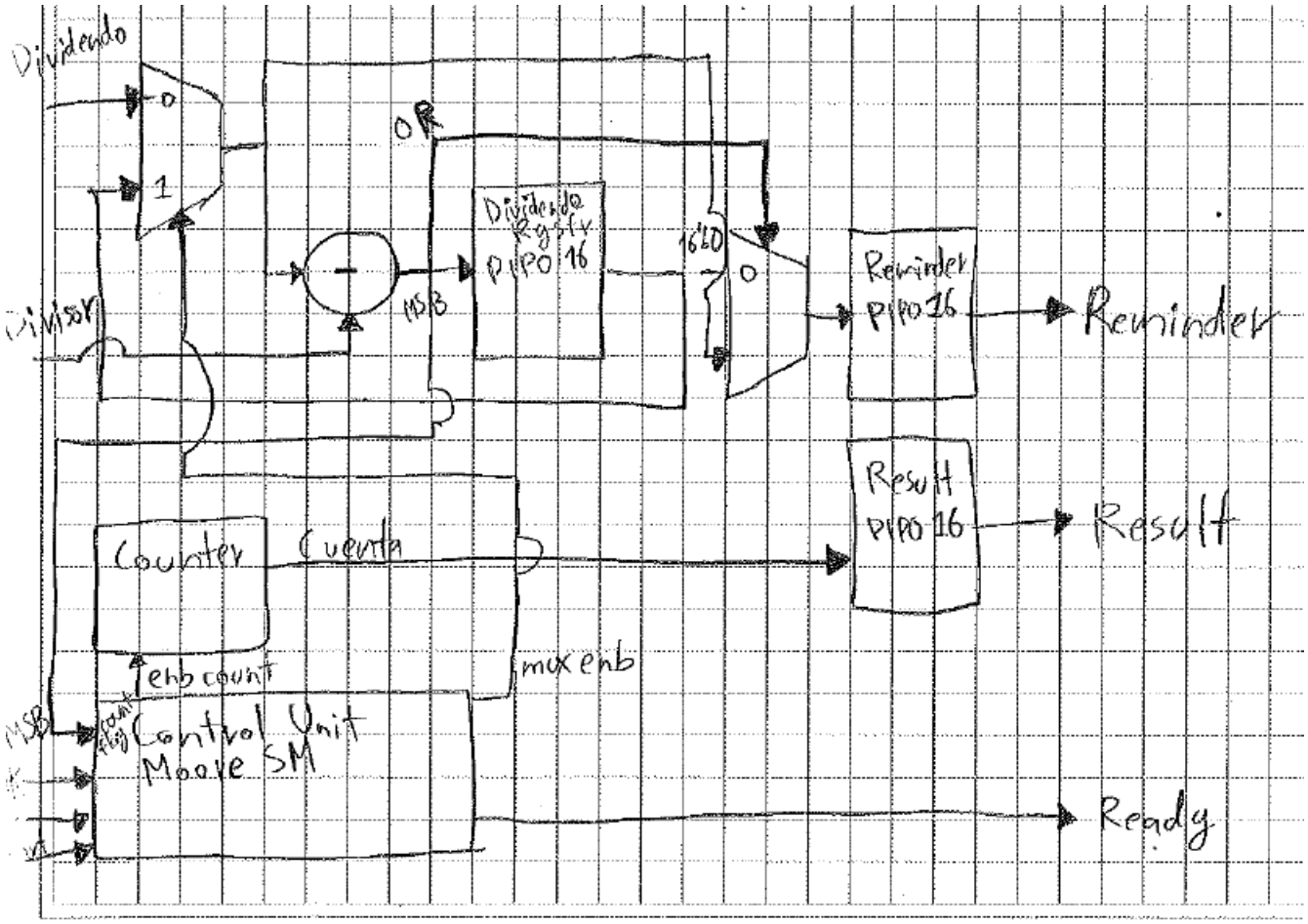


Ilustración 3. Microarquitectura del módulo de división.

Diseño sencillo con 8 módulos en total, que se basa en ir restando el divisor al dividendo para ver cuántas veces cabe dentro de él. A continuación, se describe cada uno de ellos y su utilización:

3 PIPOs/Registros de 16 bits c/u: Para almacenar el dividendo, residuo y resultado.

1 Contador: para contar los ciclos de la operación, este está sujeto al número de restas hasta que la operación de un resultado negativo.

2 Multiplexores: Para escoger lo que se va guardando en el registro del Residuo.

1 Sumador/Restador (general): Realizar la operación de resta, restando el divisor del dividendo las veces que sea necesario.

1 Unidad de control (Máquina de estados): Para controlar el flujo de la operación.

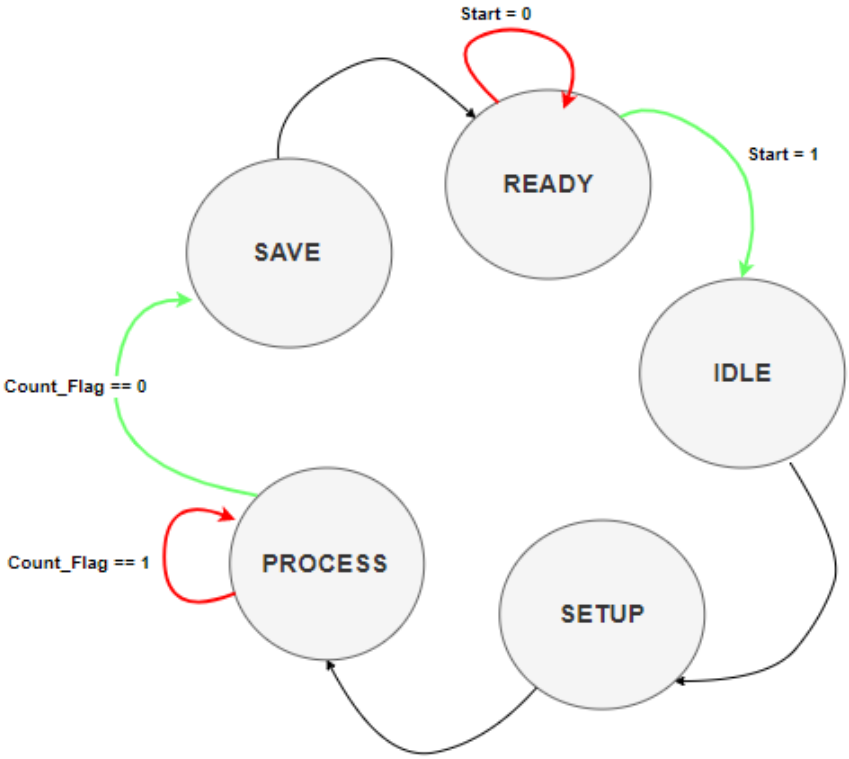
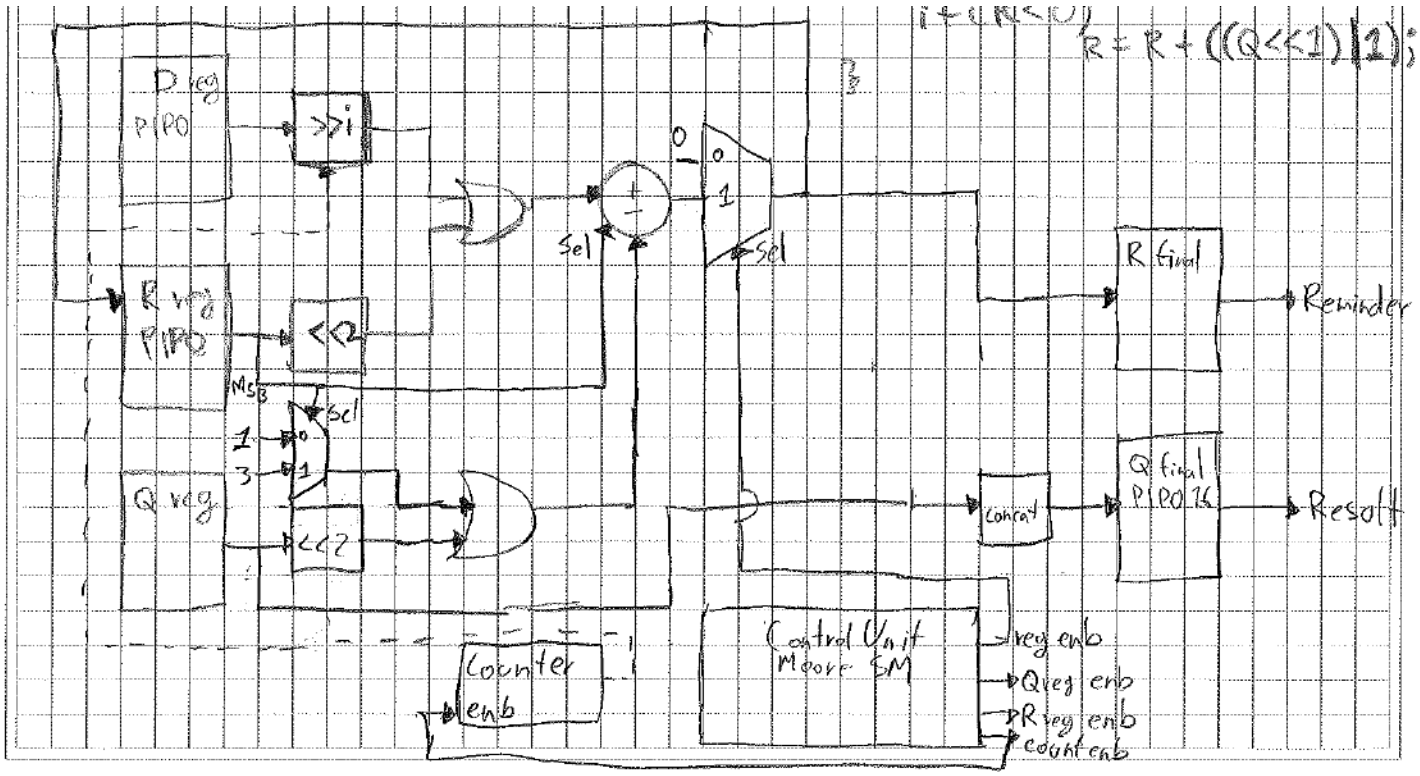


Ilustración 4. Máquina de estados del módulo de división.

Raíz Cuadrada



Para la raíz cuadrada se realizó la abstracción del código en C proporcionado por el profesor, abstrayendo los siguientes 19 módulos para la arquitectura:

5 PIPOs/Registros de 16 bits c/u: Para almacenar D, Q, R y el residuo y resultado finales.

1 Contador: para contar los ciclos de la operación.

5 Multiplexores: Para escoger lo que se va guardando en el registro del Residuo.

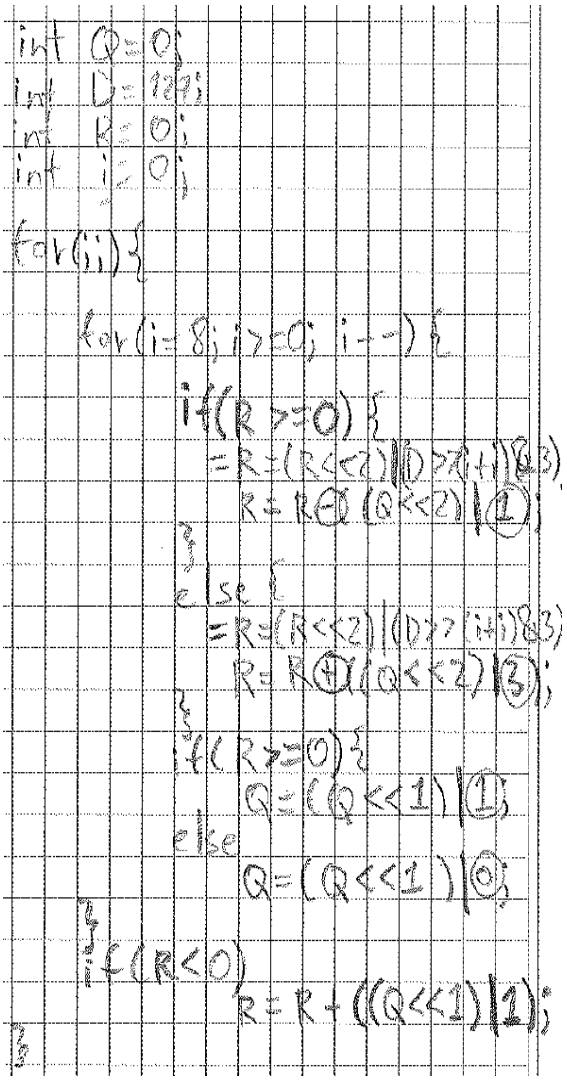
3 Shift module: Para escoger lo que se va guardando en el registro del Residuo.

2 OR: Para escoger lo que se va guardando en el registro del Residuo.

1 Módulo para concatenar: Básicamente para hacer un shift a la izquierda y agregar en el LSB un 0 o 1.

1 Sumador/Restador (general): Realizar la operación de suma o resta dependiendo de si R es positivo o negativo (su MSB).

1 Unidad de control (Máquina de estados): Para controlar el flujo de la operación.



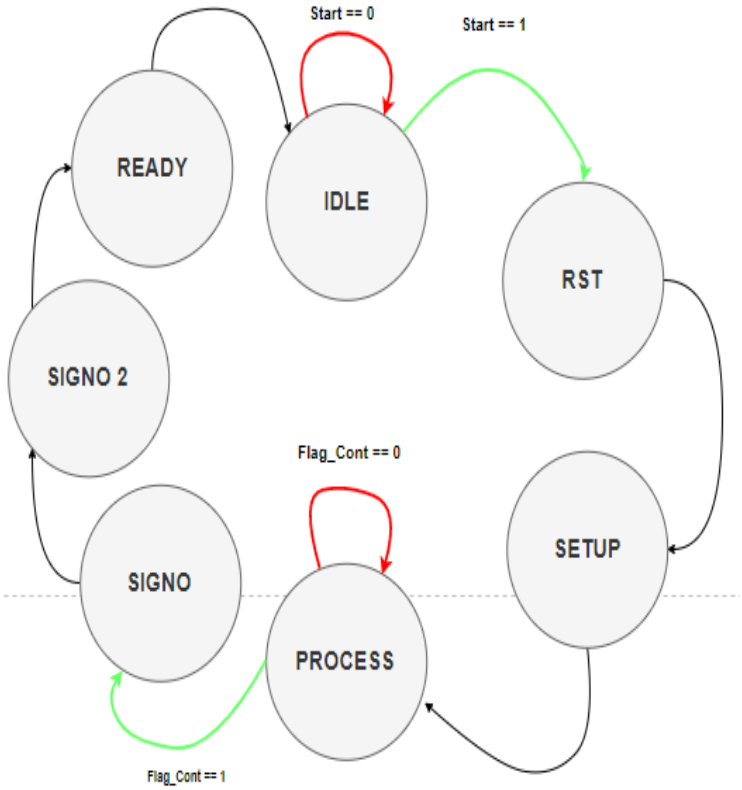
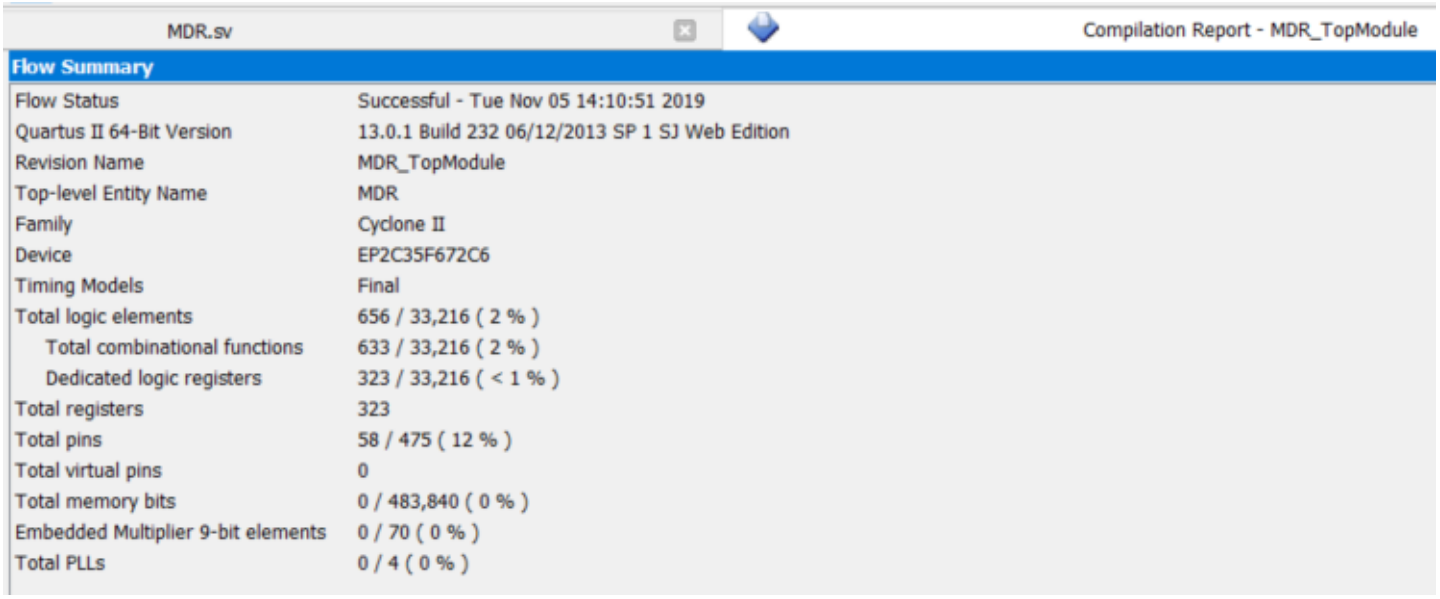


Ilustración 5. Diagrama de estados de la máquina del módulo de raíz cuadrada.

RESULTADOS DE SÍNTESIS

La compilación del diseño del módulo MDR arroja un total de 656 elementos lógicos:

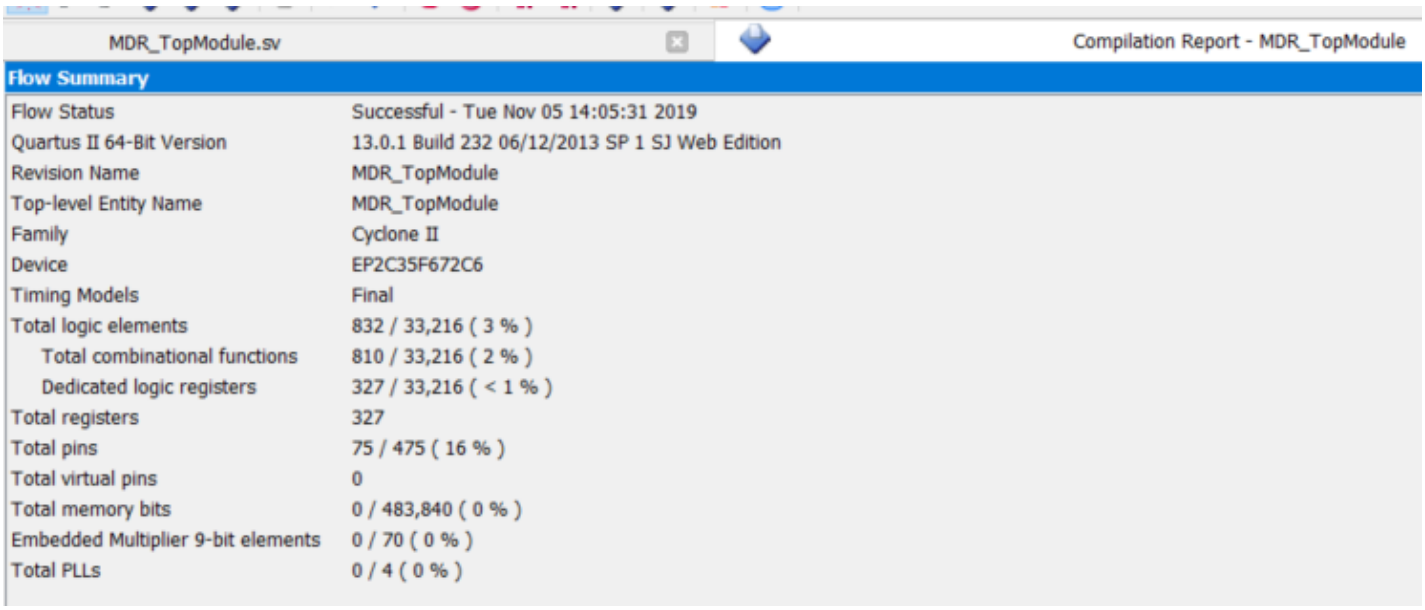


MDR.sv — Compilation Report - MDR_TopModule

Flow Summary

Flow Status	Successful - Tue Nov 05 14:10:51 2019
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	MDR_TopModule
Top-level Entity Name	MDR
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	656 / 33,216 (2 %)
Total combinational functions	633 / 33,216 (2 %)
Dedicated logic registers	323 / 33,216 (< 1 %)
Total registers	323
Total pins	58 / 475 (12 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Considerando a los módulos para los displays de 7 segmentos, y un par de módulos one-shot para señales de Start y Load, el diseño en su totalidad al compilar arroja un total de 832 **logic elements**:



MDR_TopModule.sv — Compilation Report - MDR_TopModule

Flow Summary

Flow Status	Successful - Tue Nov 05 14:05:31 2019
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	MDR_TopModule
Top-level Entity Name	MDR_TopModule
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	832 / 33,216 (3 %)
Total combinational functions	810 / 33,216 (2 %)
Dedicated logic registers	327 / 33,216 (< 1 %)
Total registers	327
Total pins	75 / 475 (16 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

SIMULACIÓN EN MODELSIM

Se realizaron 4 simulaciones, tres para los módulos individuales de multiplicación, división y raíz cuadrada y una última del módulo MDR en conjunto que muestra el funcionamiento de las tres operaciones cambiando el selector de operación, así como los resultados negativos/excepciones.

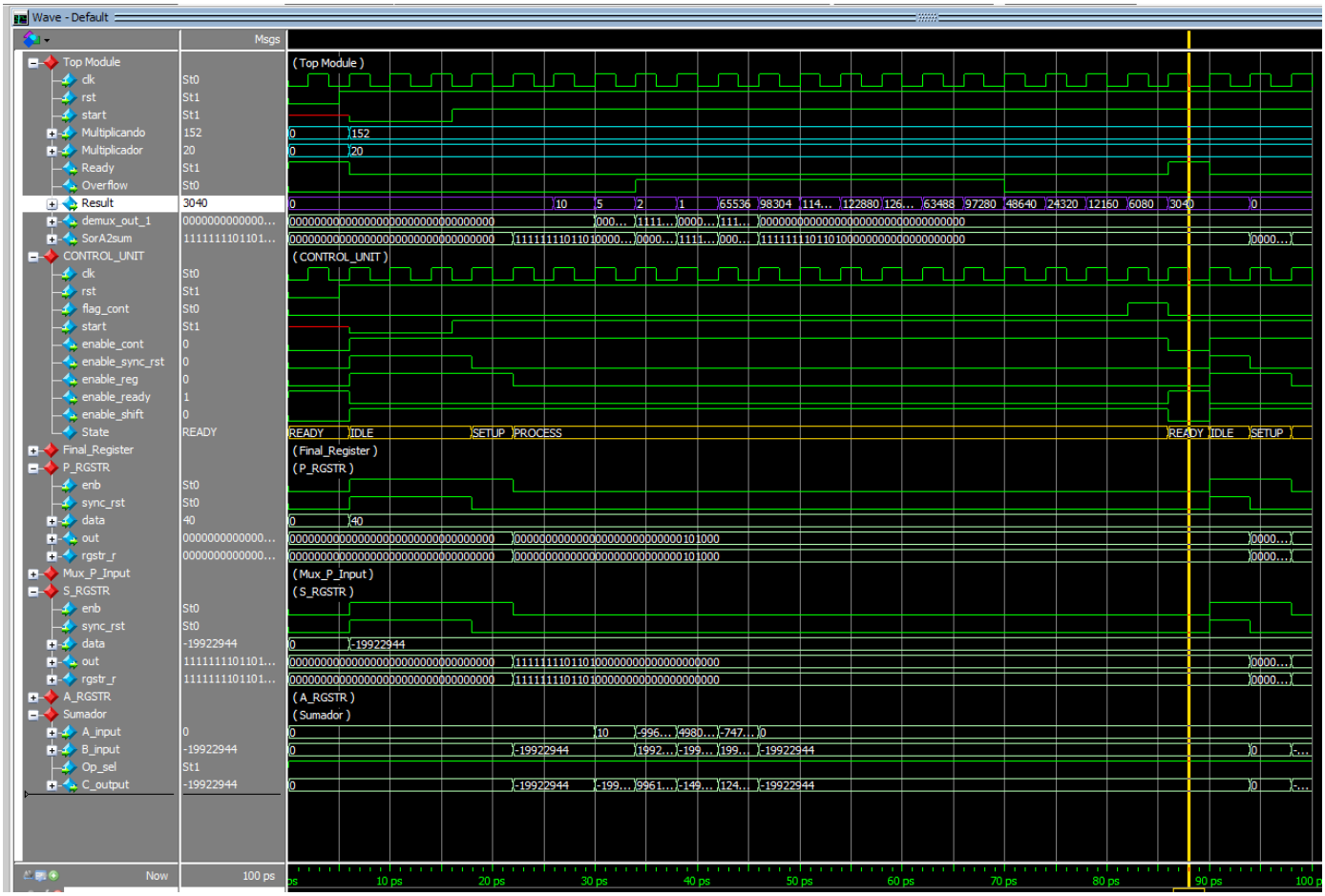
Al abrir en ModelSim uno de los cuatro proyectos de simulación: Load, después buscar la macro file correspondiente a la simulación que se está corriendo: DIV_review_wave.do, MDR_review_wave.do, MULT_review_wave.do, SQRT_review_wave.do.

NOTA: Para reproducir la simulación de cada módulo se debe des comentar en el código el sumador interno de cada módulo, ya que por requerimientos de la práctica se dejó todo conectado al sumador externo general para los 3 módulos. De igual manera comentar las entradas de cada módulo del sumador (adder_out) para que el módulo tomé adder_out_w como la salida del sumador interno y asegurarse que se utilice específicamente la salida del sumador interno (_w) en el multiplexor en el caso de la multiplicación.

Multiplicación

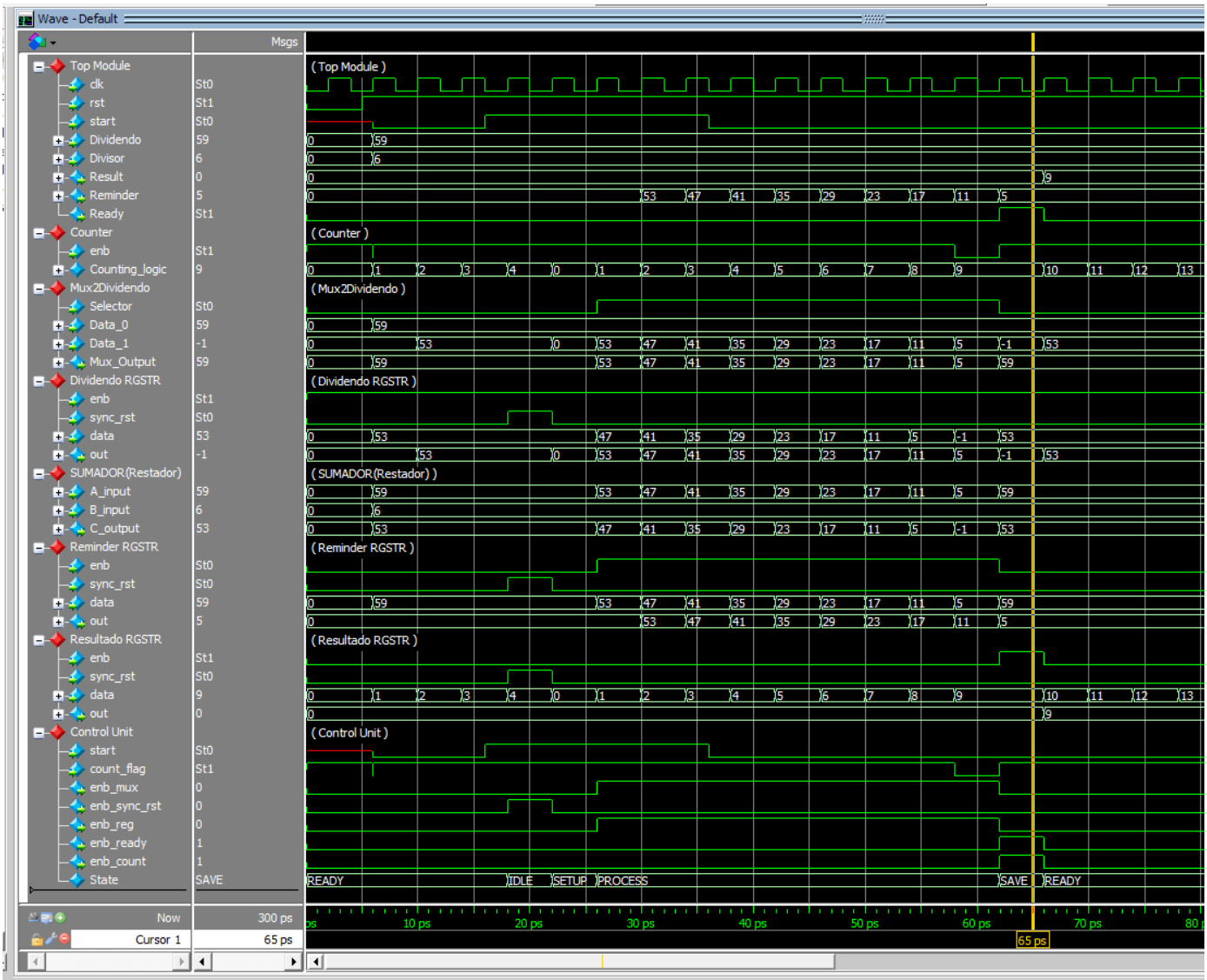
En la siguiente captura se aprecia la operación del módulo multiplicador, donde el multiplicando y multiplicador son

152 x 20, y el resultado 3040 en el momento en que la máquina de estados está en READY:



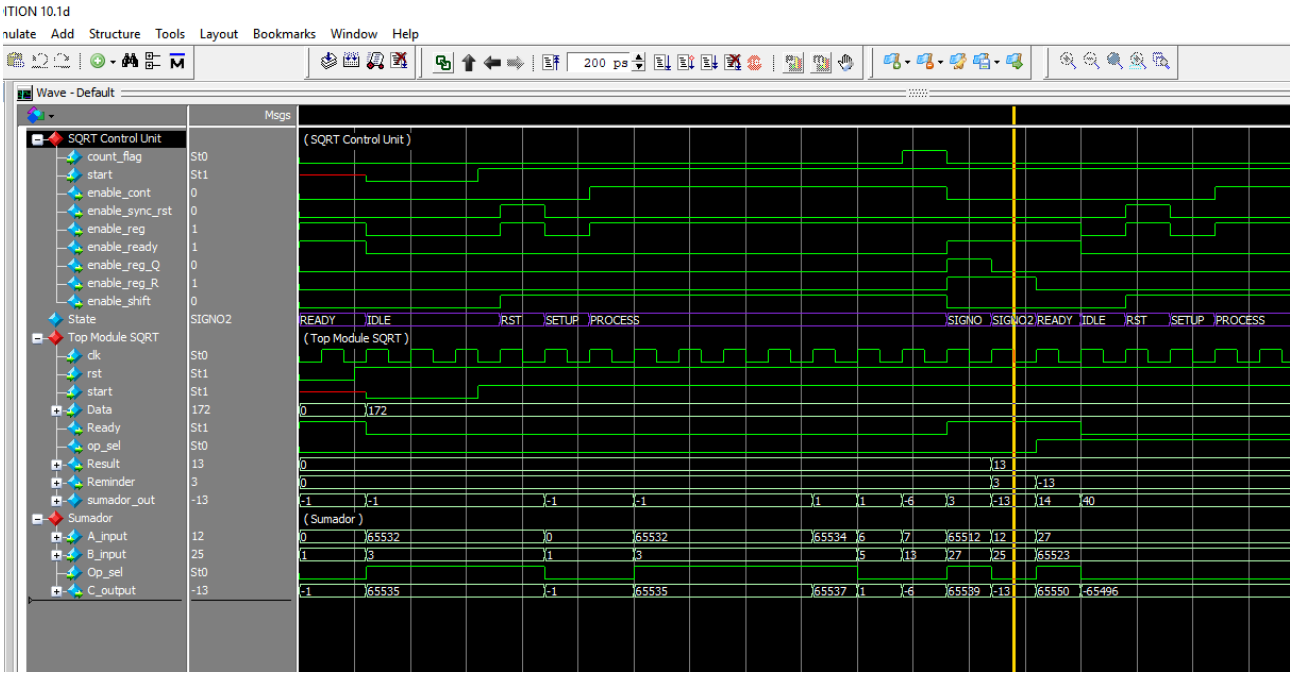
División

A continuación, se muestra la simulación del módulo de la división, donde se divide 59 por 6 (señales Dividendo y Divisor) teniendo como resultado 9 y residuo 5, esto en el tiempo en que la máquina de estados indica el estado READY:



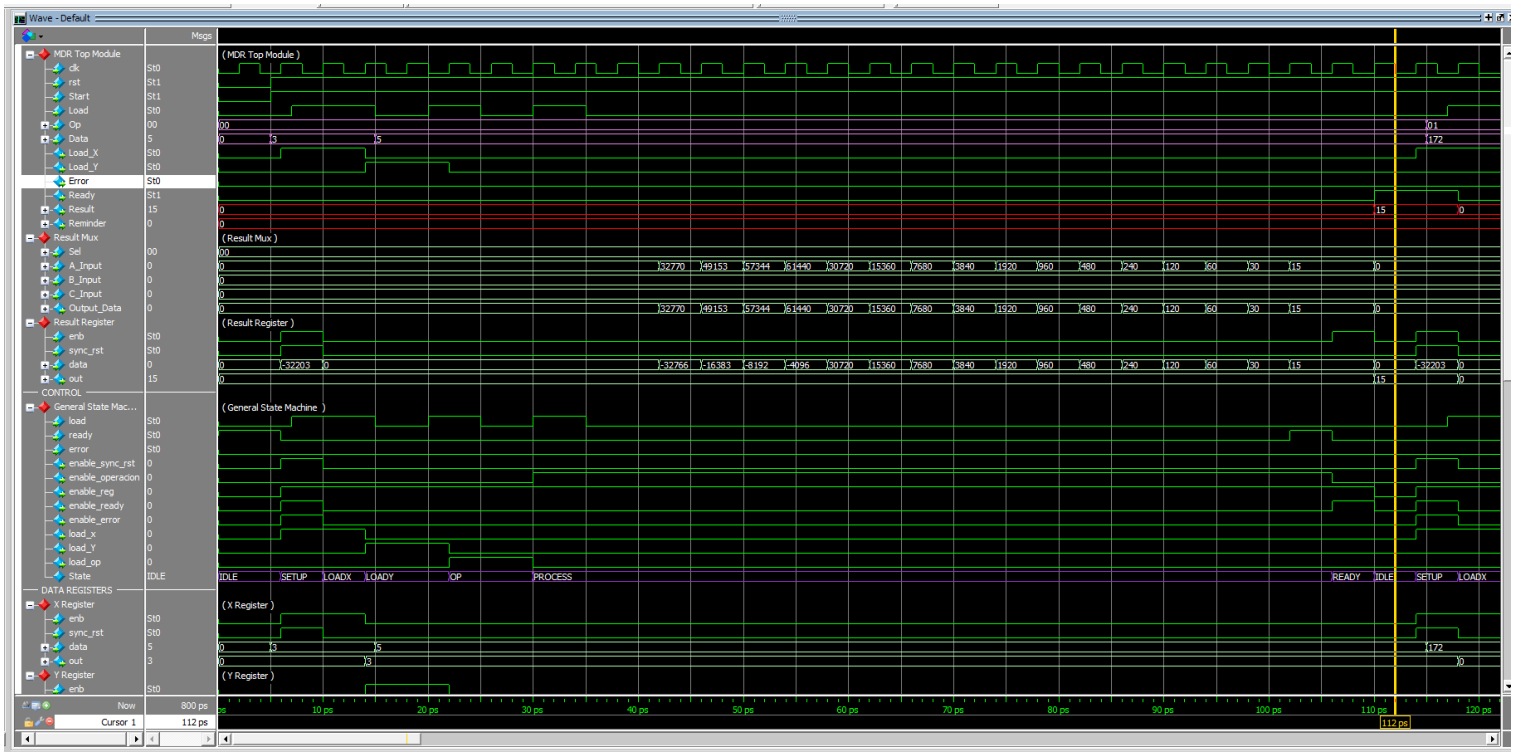
Raíz Cuadrada

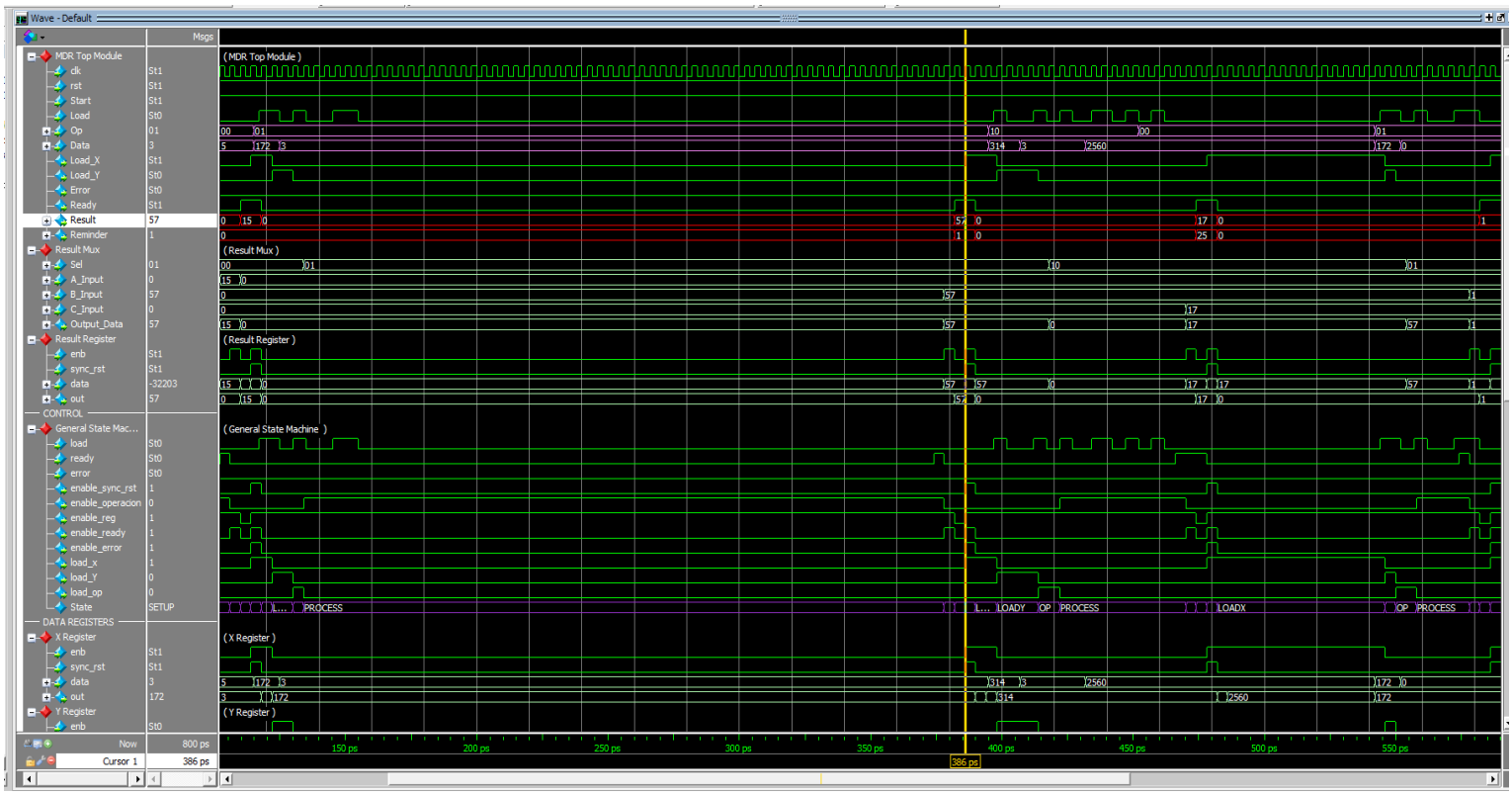
Para la raíz cuadrada se aprecia en la siguiente simulación una operación de raíz cuadrada de 172 y se aprecian en las señales de Result y Reminder como resultados 13 y 3 respectivamente, así mismo se aprecia el cambio en la máquina de estados del módulo de la raíz (señal morada):



Simulación General

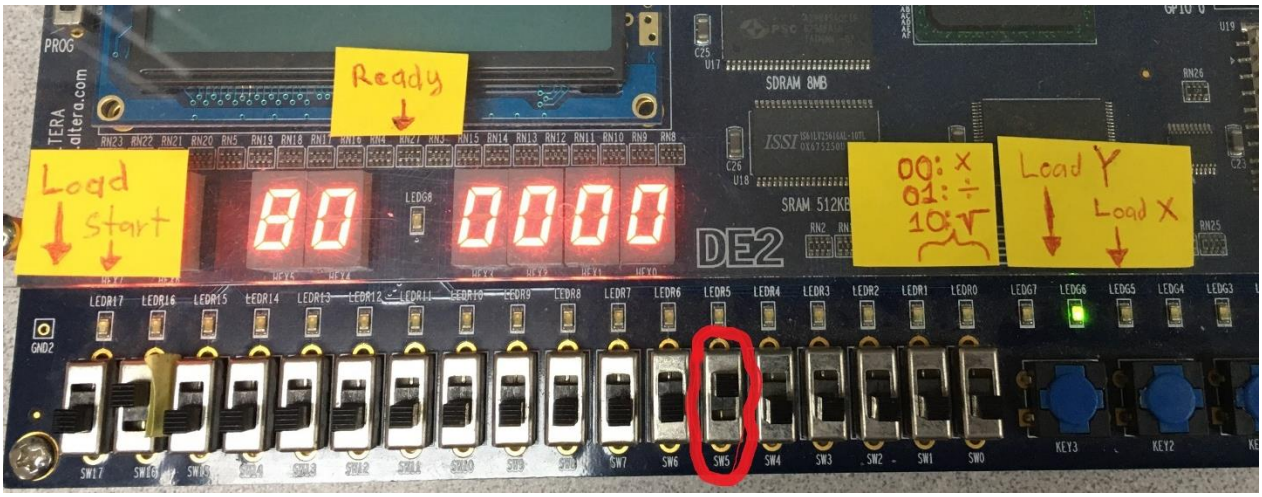
En las siguientes dos capturas se aprecia primero una multiplicación (op. 00) de 3 x 5 = 15, y en la segunda captura una división (op. 01) de 172/3 = 57 residuo 1 y una raíz cuadrada (op. 10) de 314 = 17 con residuo 25.




Wave - Default
Msgs
MDR Top Module
clk
rst
Start
Load
Op
Data
Load_X
Load_Y
Error
Ready
Result
Reminder
Result Mux
Sel
A_Input
B_Input
C_Input
Output_Data
Result Register
enb
sync_rst
data
out
CONTROL
General State Mac...
load
ready
error
enable_sync_rst
enable_operacion
enable_reg
enable_ready
enable_error
load_x
load_Y
load_op
State
DATA REGISTERS
X Register
enb
sync_rst
data
out
Y Register
enb
(MDR Top Module)
(Result Mux)
(Result Register)
(General State Machine)
(X Register)
(Y Register)
PROCESS
LOADY
OP
LOADX
Now
800 ps
Cursor 1
386 ps
150 ps
200 ps
250 ps
300 ps
350 ps
400 ps
450 ps
500 ps
550 ps

IMPLEMENTACIÓN DEL DISEÑO EN FPGA

El funcionamiento del diseño se muestra en las siguientes imágenes. Primero se debe colocar el switch 16 (Start) en 1 lógico, una vez hecho esto prenderá el LED verde LEDG6 que indicará que se está esperando el dato X. En la siguiente imagen se ha dado Start y está esperando X, está en el puerto de datos (switch 0 al 15) el dato 32 (0000000000100000):



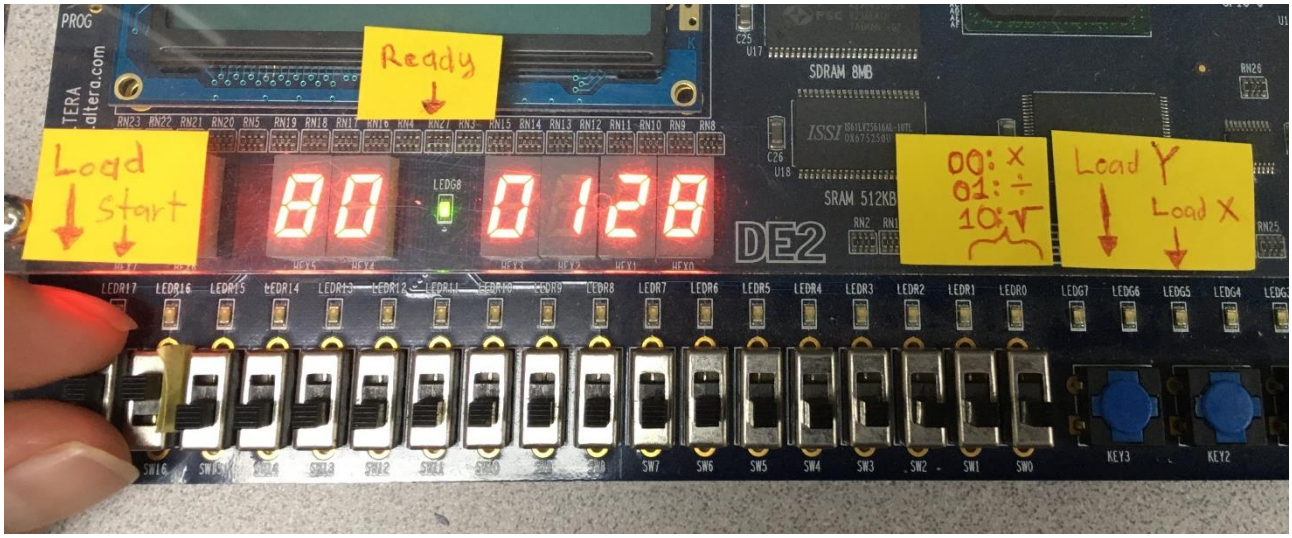
Una vez se pone en alto el switch 17 (Load) se pasa al siguiente estado esperando el dato Y, encenderá el LED verde LEDG7 y esperará el dato que está en el puerto de datos que será guardado una vez se de Load nuevamente. En la siguiente imagen introducimos el dato 4 (0000000000000100) y después damos Load:

Después de esto se debe introducir la operación como un tercer dato, en los dos bits menos significativos del puerto de datos (sin importar los demás, únicamente se tomarán estos dos):

- 00 - > multiplicación
- 01 - > división
- 10 - > raíz cuadrada

En la siguiente imagen introducimos un 00 para indicar que queremos multiplicar los números anteriores (32 x 4) y damos Load nuevamente:

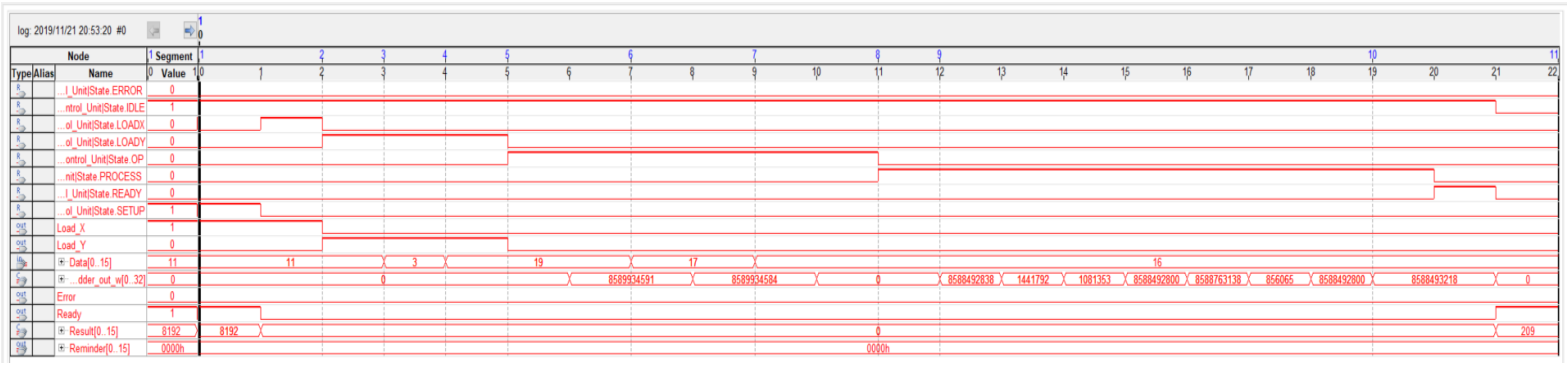
En la siguiente imagen apreciamos el resultado (128) y así mismo el LED verde de READY que indica que la operación está lista y el resultado es desplegado en los displays de 7 segmentos:



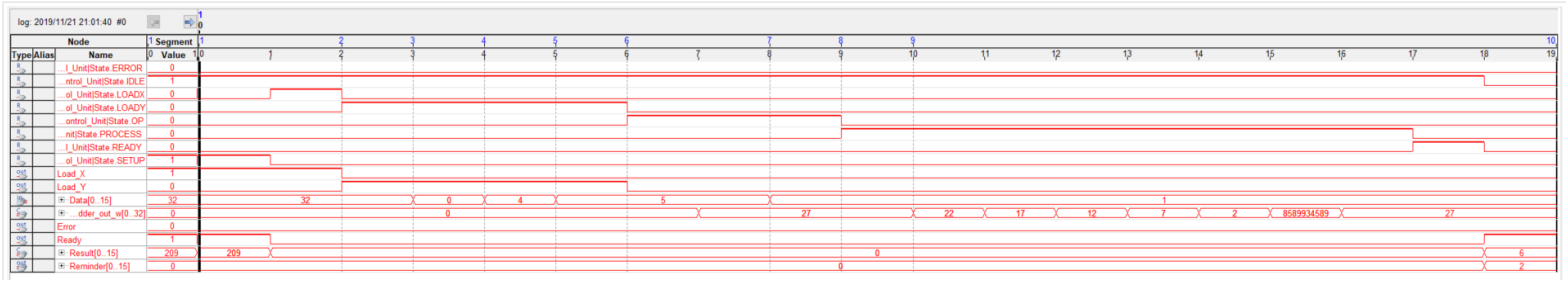
SIGNAL TAP LOGIC ANALYZER

Para la operación en tiempo real del diseño implementado en el FPGA se apoyó de la herramienta Signal Tap Logic Analyzer. Se adjuntan las imágenes por fuera del reporte para poder apreciarse con más detalle los cambios de estados y la salida del acumulador:

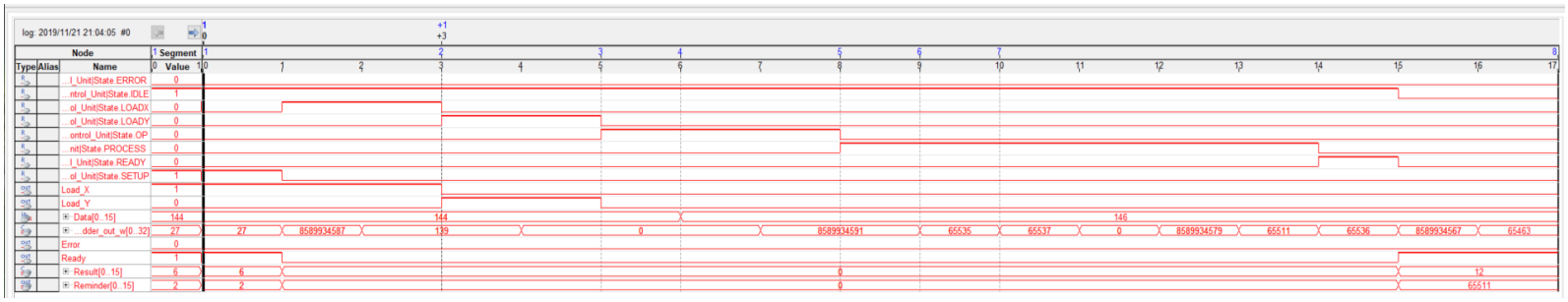
Multiplicación:



División:



Raíz cuadrada:



CONCLUSIONES

.

- En el caso de la raíz cuadrada el diseño de la arquitectura fue diferente porque se contaba con un código en C del cual se pudo abstraer módulos básicos como compuertas OR, multiplexores y shifters en conjunto con registros, por lo que ahí se tuvo qué comprender la lógica y flujo para poder dar una propuesta. Considero que esto facilitó las cosas comparado a la multiplicación y división donde se tenía que comprender la operación primero, en la raíz realmente no se comprendió el porqué del algoritmo si no que se siguió únicamente su "traducción" a un hardware aterrizado.

- En cuanto al Datapath y el Controlpath, cada módulo individual cuenta con su máquina de estados de Moore controladora, así como el módulo de operación de MDR completo, y es bastante lógico que tiene que haber comunicación entre las máquinas de estado menores para que se les indique cuando deben activar su módulo correspondiente y para que estas retroalimenten cuando ya está listo el dato que arroja la operación de su módulo correspondiente.

- La simulación ayudó bastante a verificar el correcto funcionamiento de cada módulo. Aunado a esto, es evidente que se vuelve más complejo cada vez a medida que el módulo crece en complejidad y submódulos, y esto se notó en la simulación del módulo MDR completo al querer simular las tres operaciones, pues se tiene que tener en cuenta en qué momento (picosegundos) se le introduce qué datos y en qué momento se activan qué señales, después de esto el uso del SignalTap es meramente para verificar el funcionamiento en tiempo real, pues al simular no te asegura que el diseño correrá de igual manera ya implementado en el hardware.

REPOSITORIO

. El código completo del MDR se encuentra en el siguiente repositorio en GitHub bajo el folder P2:

https://github.com/Adramusic/DVVSD2019_OTONO