**Arquitectura de Computadoras para Ingeniería.**

# Proyecto n°2

Implementación de un Multiplicador y Divisor de 8 bits en Verilog.

Primer Cuatrimestre de 2020.

Alumna: Parra, Nadina Guadalupe.
L.U. n°:106563.

## Alcance y limitaciones de la implementación:

En la realización tanto del multiplicador como del divisor no encontré ninguna limitación, los dos funcionan bien para cualquier valor ingresado en los mismos, teniendo siempre en cuenta que se va a cumplir el tamaño de los datos.

## Descripción de los bloques de código:

MULTIPLICADOR:

**module CLAA:**
-Datos de Entrada: clk(reloj), x(operando x de 4 bits), y(operando y de 4 bits), ci(carry inicial).
-Datos de Salida: s(suma de x e y, de 4 bits), cj(carry de salida), gj(valor de generación), pj(valor de propagación).
-Variables del bloque: c(arreglo que contiene los carrys internos), g(arreglo que contiene los valores de generación de cada suma), p(arreglo que contiene los valores de propagación de cada suma).
-Inicia los valores de cj, gj, pj, c, p y s.
-Cada vez que hay un flanco ascendente, realiza la suma de cada carácter de x e y. Calcula el generador y la propagación de cada suma y obtiene el carry. Así con cada carácter de x e y, y haciendo ripple de los carries internos. Luego calcula el carri de salida y los valores de generación y propagación finales.
-Los cálculos de la suma y los carries internos los hace con asignación no bloqueante y los cálculos de generación y propagación los realiza con asignación bloqueante.

**module SUMADOR:**
-Datos de Entrada: clk(reloj), x(operando x de 8 bits), y(operando y de 8 bits).
-Datos de Salida: s(operando suma de 8 bits)

Instancia dos CLAA de 4 bits en ripple.

**module MULTIPLICADOR:**
-Datos de Entrada: x(operando x de 8 bits), y(operando y de 8 bits), suma(operando suma de 8 bits), clk(reloj)
-Datos de Salida: A(registro A de 8 bits), B (registro B de 8 bits) y MQ(registro MQ de 8 bits)
- Datos internos: cont(contador) y cont_final(Que lo uso para cortar la ejecución).

Inicia los valores de las salidas, de cont y cont_final.
En cada flanco ascendente:
Lee el dígito menos significativo de MQ
si es 1 realiza la suma de A y B en A y realiza un shift en A y MQ.
si es 0 realiza un shift en A y MQ
Como la suma lleva 8 ciclos, con el contador lleva un registro para saber si la suma término.

Si los valores de x e y cambian se encarga de reestablecer todos los valores.

DIVISOR:

**module DIVISOR:**
-Datos de Entrada: x(Operando x de 8 bits), y(Operando y de 8 bits) y clk (señal de reloj).
-Datos de Salida: A(Registro de 8 bits A en donde se almacena el resto) y MQ(Registro de 8 bits  MQ en donde se almacena el resultado)
-Datos Internos: B(Registro de 8 bits que se utiliza para almacenar y o -y en cada caso), q(registro de 1 bits que guarda el bit menos significativo de A) y cont(lleva el registro de ciclos para dejar de ejecutar una vez que termina)

Inicializa las operadores y el contador.
Cuando el flanco es ascendente:

Si no estoy en el último ciclo, realiza un shif de A y MQ y si q=1 agrega un 0 en el bit menos significativo de MQ y almacena y en B, si q=0 ingresa un 1 en MQ y almacena -y en B.

Si estoy en el último ciclo, realiza un shift solo en MQ e ingresa el valor adecuado según el valor de q.

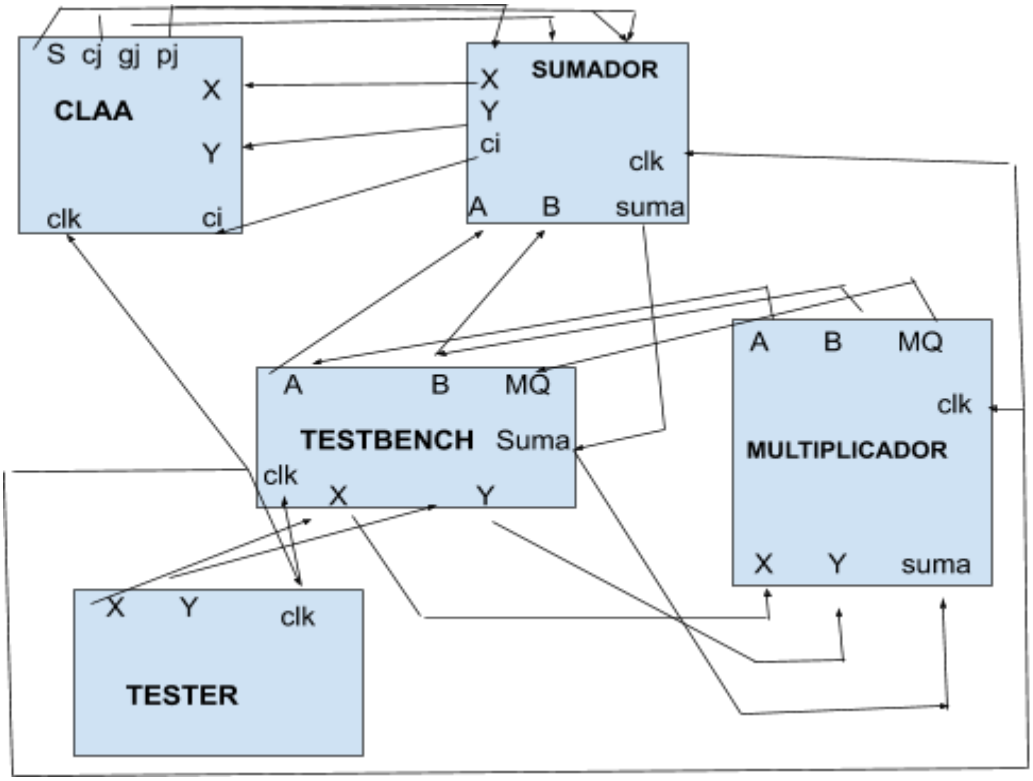
Cuando el flanco es descendente:

Si termine, y q es 1, entonces sumo y.

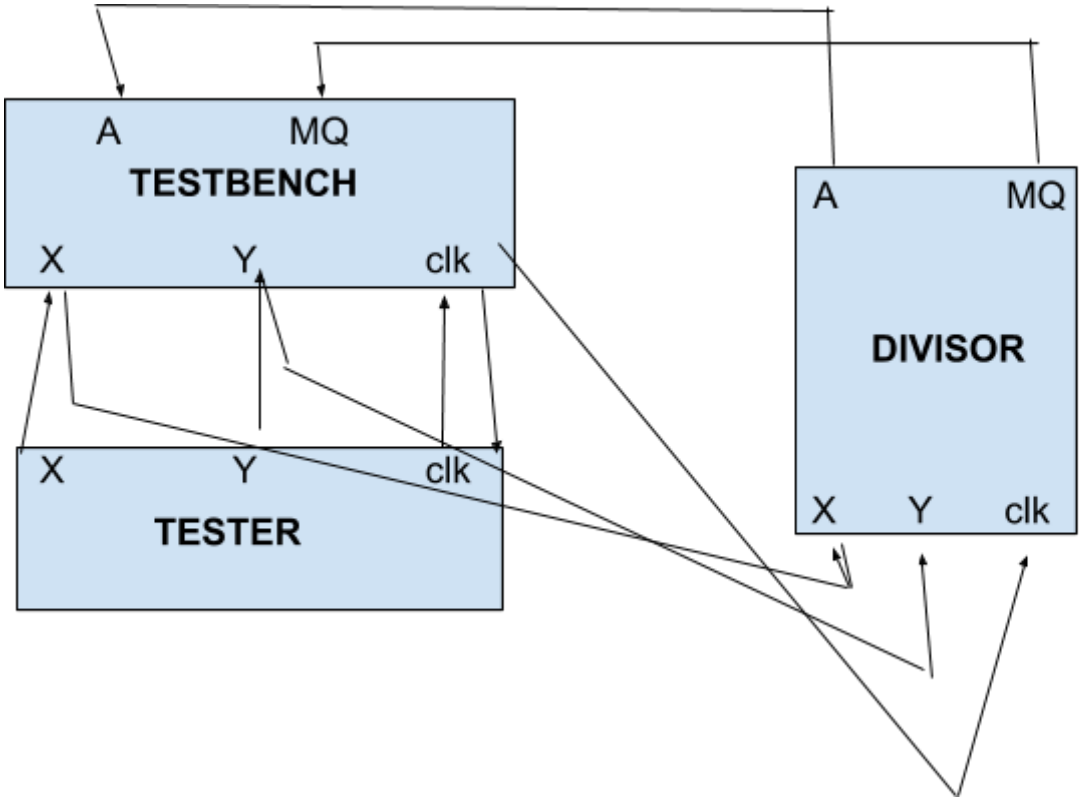
Si no termine realizó la suma o resta correspondiente en A.

Si los valores de x e y cambian se encarga de reestablecer todos los valores.

## Diagrama estructural del diseño: Multiplicador

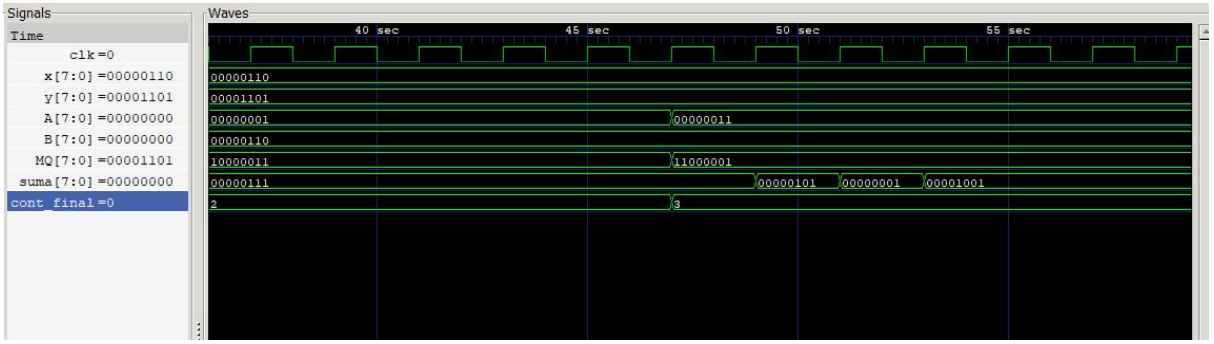


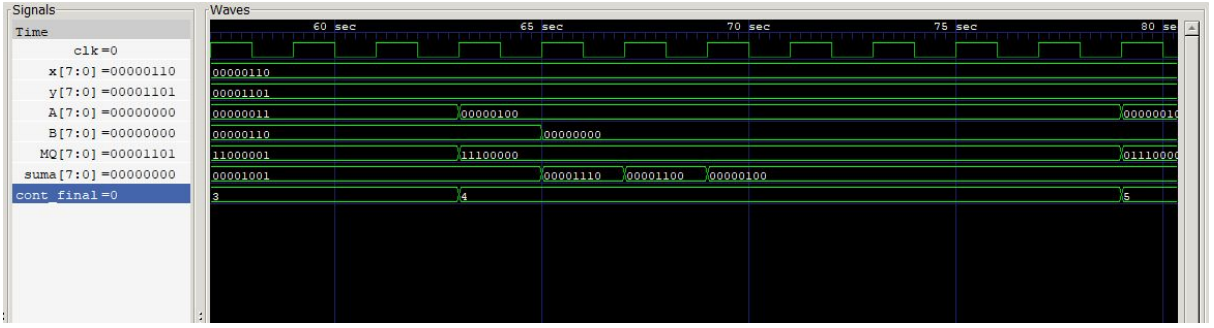
## Diagrama estructural del diseño: Divisor

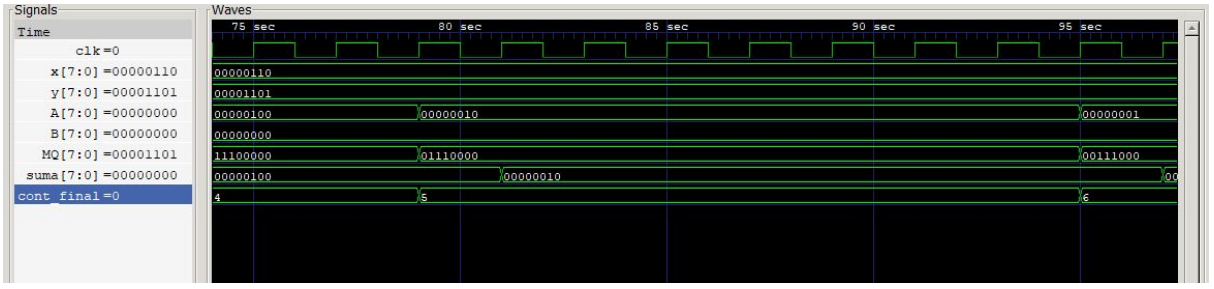


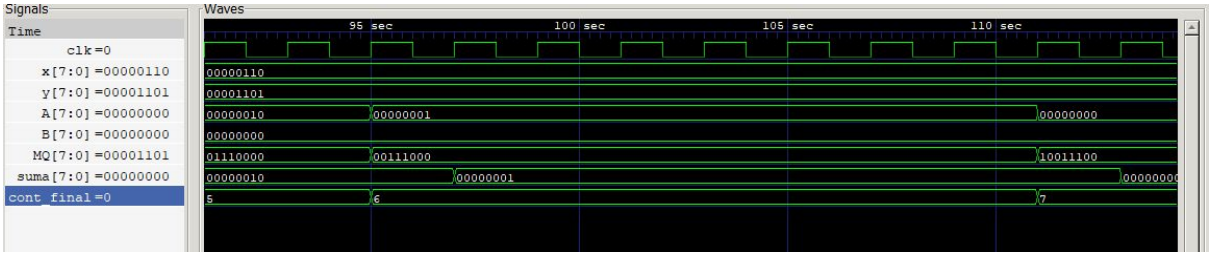

## GtkWave:

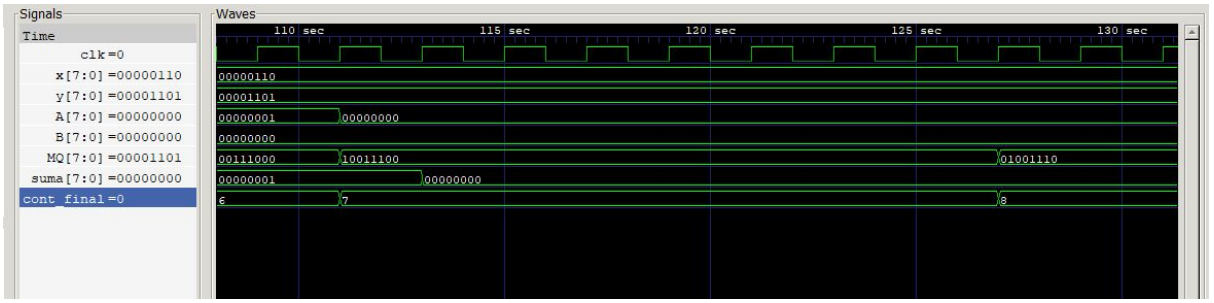
MULTIPLICADOR: X=6 Y=13

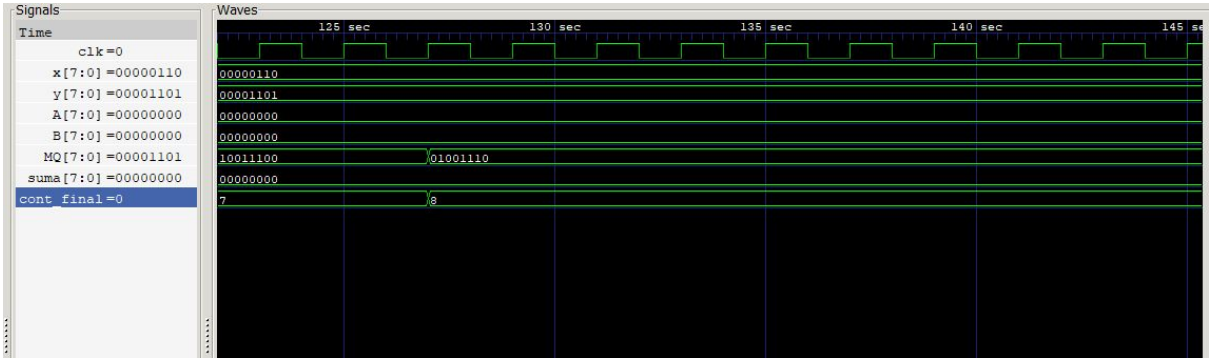

Signals
Time
clk=0
x[7:0]=00000110
y[7:0]=00001101
A[7:0]=00000000
B[7:0]=00000000
MQ[7:0]=00001101
suma[7:0]=00000000
cont_final=0
Waves
40 sec
45 sec
50 sec
55 sec
00000110
00001101
00000001
00000011
00000110
10000011
11000001
00000111
00000101
00000001
00001001
2
3


Signals
Time
clk=0
x[7:0]=00000110
y[7:0]=00001101
A[7:0]=00000000
B[7:0]=00000000
MQ[7:0]=00001101
suma[7:0]=00000000
cont_final=0
Waves
60 sec
65 sec
70 sec
75 sec
00000110
00001101
00000011
00000100
00000110
00000000
11000001
11100000
00001001
00001110
00001100
00000100
3
4
5

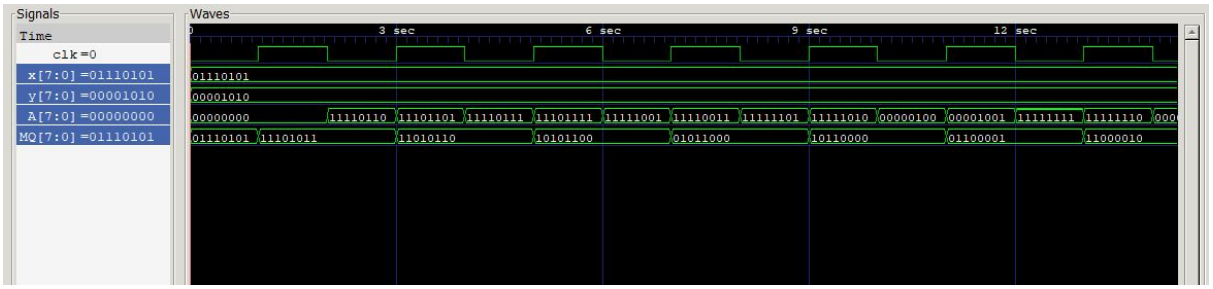

Signals
Time
clk=0
x[7:0]=00000110
y[7:0]=00001101
A[7:0]=00000000
B[7:0]=00000000
MQ[7:0]=00001101
suma[7:0]=00000000
cont_final=0
Waves
75 sec
80 sec
85 sec
90 sec
95 sec
00000110
00001101
00000100
00000010
00000001
00000000
11100000
01110000
00111000
00000100
00000010
4
5
6

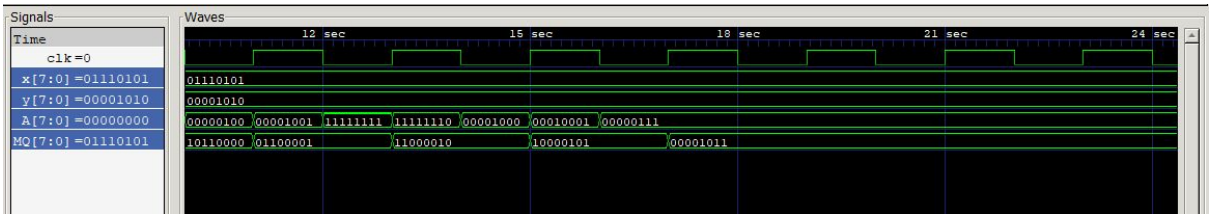

Signals
Time
clk=0
x[7:0]=00000110
y[7:0]=00001101
A[7:0]=00000000
B[7:0]=00000000
MQ[7:0]=00001101
suma[7:0]=00000000
cont_final=0
Waves
95 sec
100 sec
105 sec
110 sec
00000110
00001101
00000010
00000001
00000000
00000000
01110000
00111000
10011100
00000010
00000001
5
6
7


Signals
Time
clk=0
x[7:0]=00000110
y[7:0]=00001101
A[7:0]=00000000
B[7:0]=00000000
MQ[7:0]=00001101
suma[7:0]=00000000
cont_final=0
Waves
110 sec
115 sec
120 sec
125 sec
130 sec
00000110
00001101
00000001
00000000
00000000
00111000
10011100
01001110
00000001
00000000
6
7
8




DIVISOR: X=117 Y=10




## Observaciones y conclusiones del diseño:

En esta entrega pude entender mucho mejor el funcionamiento del multiplicador y el divisor y como lo visto en teoría poder implementarlo utilizando el lenguaje verilog, teniendo así un mejor manejo y una idea más clara de cómo llevar el algoritmo a la implementación y ejecución de los dos ejercicios.

Con respecto al diseño se puede observar que hay que prestar atención en los ciclos que llevan realizar las operaciones internas de los módulos, por lo que es muy útil ir llevando una cuenta de los mismos, esto nos da facilidad para realizar diferentes operaciones en cada ciclo según corresponda, esperar algunos resultados y finalizar la ejecución una vez que se termine de resolver la operación y no quede ciclando indefinidamente.