Objetivo General.
Utilizar el FPGA para el desarrollo de circuitos combinatorios.

Objetivos específicos.

Investigar el funcionamiento de la tarjeta de desarrollo FPGA Spartan 3E

Utilizar las herramientas del Xilinx ISE

Conocer y aplicar el flujo de diseño para sistemas basados en FPGA

Propuesta del problema.
Agregue una operación *MUL al experimento 1, esta operación deberá multiplicar los valores de src1 y src2 y guardar el resultado en dst como se muestra a continuación.

Operacion	Destino	Fuente1	Fuente2	Descripcción
*MUL	dst	Src1	Src2	Dst = src1 * src2

Ejercicio 1 (Obligatorio):

Agregue una operación SMUL al experimento 1, esta operación deberá multiplicar los valores de src1 y src2 y guardar el resultado en dst .

Antes de comenzar, anote la frecuencia que la herramienta de síntesis estima para el experimento 1, además el número de LUTs, Slices y Flip-Flops.
Utilice el operador '*' de verilog para implementar la multiplicación.

El Spartan 3E cuenta con bloques dedicados de multiplicación. Por defecto, la herramienta de síntesis intentará usar estos bloques dedicados de multiplicación en lugar de sintetizarlos usando los bloques lógicos de las celdas.

Observe el reporte de la herramienta de síntesis.

```
=========================================================================
===
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                  : 2
 9x16-bit dual-port distributed RAM                : 2
# ROMs                                  : 1
 16x28-bit ROM                         : 1
# Multipliers                           : 1
 16x16-bit registered multiplier                : 1
# Adders/Subtractors                    : 2
 16-bit adder                          : 2
```

```
# Counters                                 : 1
 16-bit up counter                        : 1
# Registers                                : 72
 Flip-Flops                               : 72
# Comparators                              : 1
 16-bit comparator lessequal              : 1

=========================================================================
```

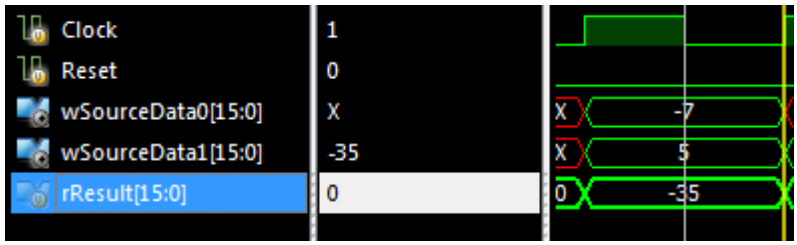
Los bloques de multiplicación del FPGA son además capaces de llevar acabo multiplicaciónes con signo (con números representados en complemento a 2). Para esto es necesario que la herramienta de síntesis entienda que las lineas de entrada a los puertos del multiplicador tienen signo.

Esto se hace de la siguiente manera:

```
wire [9:0] wA, wB;
wire [31:0] R = wA * wB;        //multiplicación sin signo


wire signed [15:0] wA, wB;
wire signed [31:0] wR = wA * wB; // multiplicación con  signo
```

Note en la figura siguiente como el simulador reconoce la multiplicación con signo:



Implemente la multiplicación con signo usando el operador “*” de verilog.

Anote la frecuencia que la herramienta de síntesis estima para esta parte, además el número de LUTs, Slices y Flip-Flops.

Modifique el código en la ROM para calcular la multiplicación de varios números tanto con signo como sin signo y despliegue lo el resultado en los LEDs.

Ejercicio 2 (Obligatorio):
Agregue una operación IMUL al MiniAlu, esta operación deberá multiplicar los valores de src1 y src2 y guardar el resultado en dst.

En esta parte se va a implementar un multiplicador del tipo “array multiplier”.

Para ilustrar el funcionamiento de este multiplicador comience por repasar el algoritmo de multiplicación fundamental que aprendió en la escuela:

```
   1 2
x  1 3
   3 6
  1 2
  1 5 6
```

Esto mismo se puede escribir en binario como sigue:

```
       1 1 0 0
   x   1 1 0 1
       1 1 0 0
     0 0 0 0
   1 1 0 0
 1 1 0 0
 1 0 0 1 1 1 0 0
```

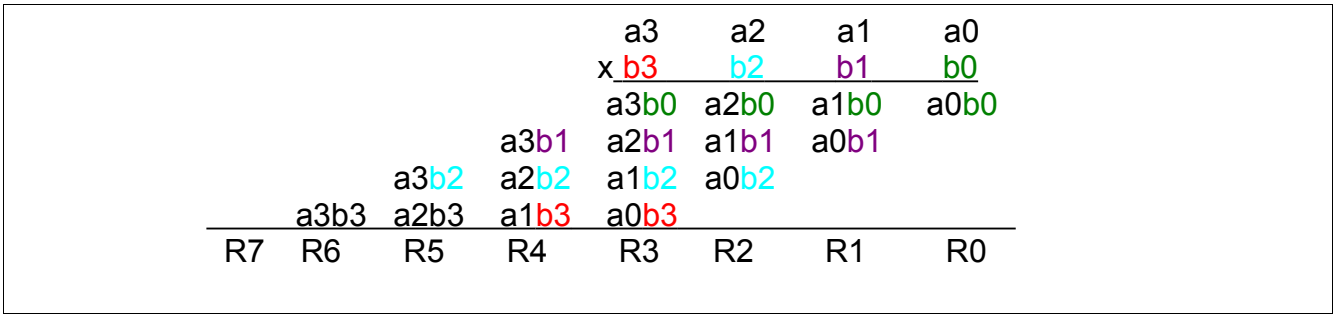
Como puede notar de la figura anterior, la mecánica de la multiplicación es la misma que en base 10. Note que a la hora de sumar se debe tomar en cuenta el acarreo.

Para utilizar el operador de suma de verilog tomando el cuenta el acarreo puede hacer lo siguiente:

```
wire [n-1:0] wResult;
wire         wCarry;

assign {wCarry, wResult } = wA + wB;
```

Sean dos números de 4 bits A = {a3,a2,a1,a0} y B = {b3,b2,b1,0} entonces note que la multiplicación se lleva a cabo de la siguiente manera:



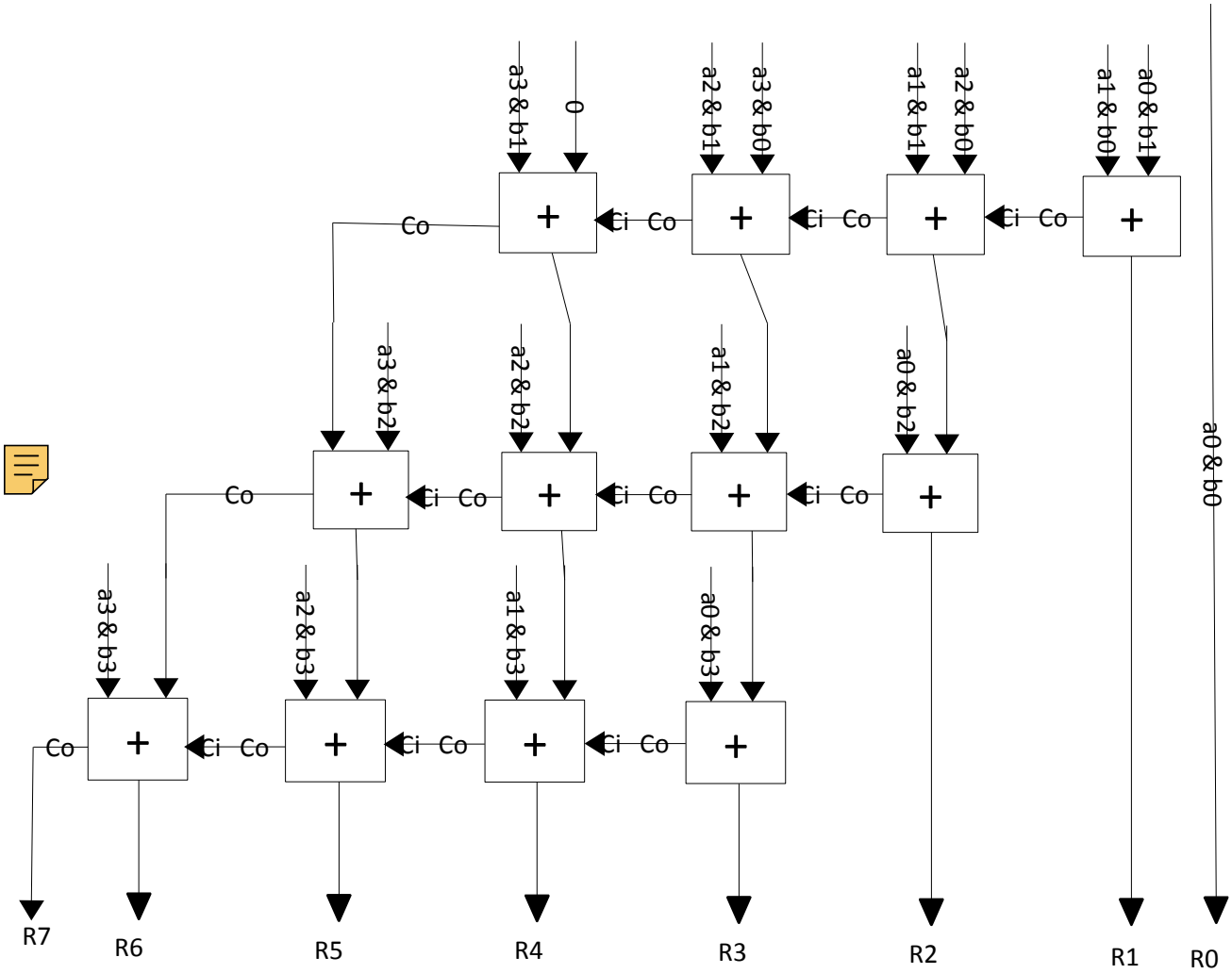
De la figura anterior se ve como

- R0 = a0 & b0
- R1 = a1 & b0 + a0 & b1
- R2 = a2 & b0 + a1 & b1 + a0 & b2 + el acarreo de R1
- etc...

Siguiendo esta lógica implemente un módulo de verilog que multiplique dos números de 4 bits sin signo y úselo para la operación MUL del miniALU.

Recuerde que la los registros de la mini ALU son se 16 bits asi que deberá poner ceros en los bits restantes.

A continuación se muestra una figura que ilustra la estructura del circuito que deberá implementar.



Anote la frecuencia que la herramienta de síntesis estima para esta parte, además el número de LUTs, Slices y Flip-Flops.

¿Como se comparan estos resultados con los del ejercicio anterior y porqué?

Modifique el código en la ROM para calcular la multiplicación de varios números sin signo y despliegue el resultado en los LEDs.

Ejercicio 3 (Obligatorio):

Extienda el circuito del ejercicio anterior para multiplicar dos números de 16 bits.

Para esto implementar esto estudie la construcción de verilog llamada "**generate**".

Los bloques de verilog **generate** le permitirán ahorrar mucho tiempo y le enseñarán como escribir código genérico para una tarea que de otra forma puede resultar muy tediosa.

Para escribir los bloques generate puede usar el constructor "**for**" y pensar en una estructura con filas y columnas como la de la figura anterior.

Recuerde que que puede declarar arreglos de cables, mire el siguiente código (los puntos suspensivos son partes dejadas intencionalmente en blanco):

```
wire[2:0] wCarry[2:0];
genvar  CurrentRow,  CurrentCol;
generate
for ( CurrentCol = 0; CurrentCol < `MAX_COLS; CurrentCol = CurrentCol + 1)
begin : MUL_ROW
…
MODULE_ADDER  # (4) MyAdder
(
  .A( … ),
  .B( … ),
  .Ci(  wCarry[ CurrentRow ][ CurrentCol ] ),
  .Co( wCarry[ CurrentRow ][ CurrentCol + 1]),
);
…
end
endgenerate
```

Además recuerde que Ci de los sumadores de las columnas de la izquierda son cero.

```
assign wCarry[ CurrentRow ][ 0 ] = 0;
```

Finalmente anote la frecuencia que la herramienta de síntesis estima para esta parte, además el número de LUTs, Slices y Flip-Flops. ¿Como se compara con el ejercicio 1?

¿Son los bloques **generate** sintetizables?

Anote la frecuencia que la herramienta de síntesis estima para esta parte, además el número de LUTs, Slices y Flip-Flops.

¿Cuantas etapas tiene este nuevo circuito de multiplicación? ¿Que ocurre con el periodo del reloj si se añaden más y más etapas de lógica combinatoria?

¿Que ocurre con la frecuencia del circuito si añade Latches entre cada etapa de sumadores?

Modifique el código en la ROM para calcular la multiplicación de varios números sin signo y despliegue el resultado en los LEDs. Note que solo cuenta con 8 LEDs así que deberá ser creativo para desplegar estos resultados.

Ejercicio 4 (Obligatorio):

Agregue una operación IMUL2 al MiniAlu, esta operación deberá multiplicar los valores de src1 y src2 y guardar el resultado en dst.

Se va implementar un multiplicador con un algoritmo distinto.

Para entender este algoritmo debe recordar una tabla de multiplicar como la siguiente

	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	4	6
3	0	3	6	9

Esta tabla puede estar almacenada en una memoria como por ejemplo una ROM y es lo que se conoce como una LUT (Look up Table).

Buscar el resultado de una multiplicación en una LUT es muy rápido, sin embargo se vuelve poco práctico cuando hay que multiplicar números muy grandes.

¿Cuantas filas y columnas tendría una LUT que permita multiplicar dos números de 32 bits?

Por otro lado se pueden combinar pequeñas tablas LUT como la anterior para obtener el resultado de números con muchos bits.

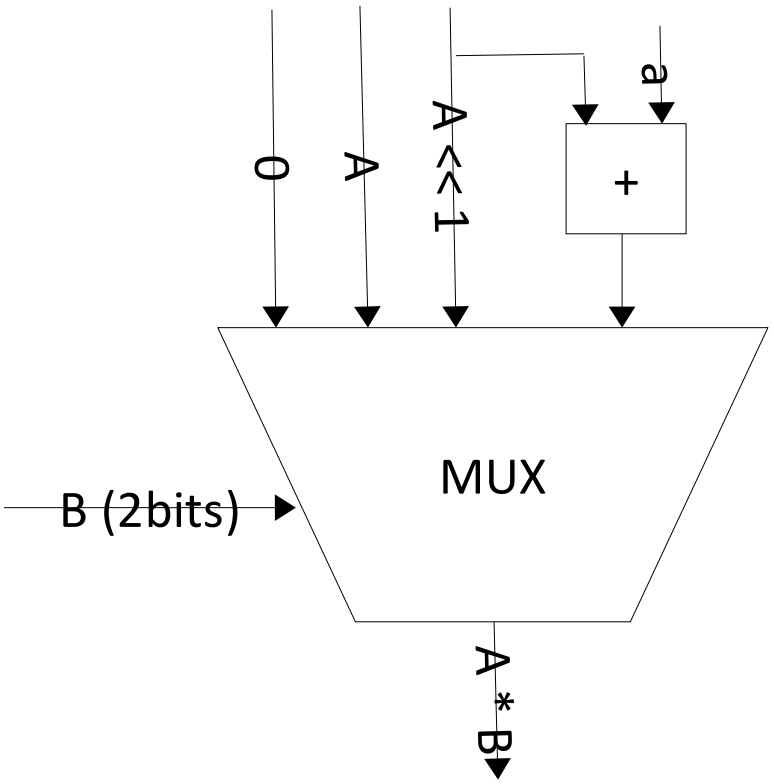
Para dos números de 2 bits A y B la tabla anterior se puede representar de la siguiente manera

B	A*B	Descripción
0 (00)	0	Sin importar el valor de A, si B es cero A*B es cero.
1 (01)	A	Sin importar el valor de A, si B es uno A*B es A.

2 (10)	2*A	Multiplicar por 2 es un corrimiento a la izquierda, A << 1
3 (11)	3*A	Multiplicar por 3 es un corrimiento a las izquierda mas A, (A << 1) + a

.

Esto ultimo se acostumbra implementar como un multiplexor como se muestra a continuación:



En este momento usted se deberá preguntar, este MUX permite multiplicar dos 2 números de 2 bits, ¿pero que ocurre con números de más de 2 bits?

Note que la figura anterior aplica para B de 2 bits, pero no importa el número de bits que tenga A.

Colocando varios bloques de multiplexores como el que se muestra y sumando los resultados parciales corridos a la izquierda el numero apropiado de posiciones, se pueden multiplicar números de cualquier tamaño.

No se incluye un diagrama de como conectar los muxes asi que usted mismo deberá pensar en la forma, como una pista recuerde que:

$A * B = A * (b7* 2^7 + b6* 2^6 + b5* 2^5 + b4* 2^4 + b3 * 2^3 + b2 *2^2 + b1*2^1 + b0* 2^0)$

$= (A* b7* 2^7 + A*b6* 2^6 + A*b5* 2^5 + A*b4* 2^4 + A*b3 * 2^3 + A*b2 *2^2 + A*b1*2^1 + A*b0* 2^0)$

Recuerde que 2^n es un corrimiento a la izquierda n posiciones, piense como agrupar lo anterior en grupos de 2 bits y acomodarlo en sumadores en cascada como en el ejercicio 2.

Implemente un circuito de multiplicación para multiplicar dos número de 4 bits para comenzar.

Luego, implemente el un circuito de multiplicación para multiplicar dos número de 16 bits.

Anote la frecuencia que la herramienta de síntesis estima para esta parte, además el número de LUTs, Slices y Flip-Flops.

¿Como se comparan estos resultados a los del experimento anterior?

Ejercicio 5 (Opcional):

El multiplexor del ejercicio anterior permite codificar una LUT para 2 bits.
¿Puede usted implementar una MUX para 4 bits? Que ventajas o desventajas tiene esta nueva implementación.