	Universidad de Costa Rica Escuela de ingeniería Eléctrica Laboratorio 2	EIE Escuela de Ingeniería Eléctrica
IE0424: Laboratorio de Circuitos Digitales II-2019		

Objetivo General

Utilizar la FPGA para el desarrollo de circuitos combinatorios.

Objetivos específicos

- Investigar el funcionamiento de la tarjeta de desarrollo FPGA Nexys 4.
- Utilizar las herramientas del Xilinx Vivado.
- Conocer y aplicar el flujo de diseño para sistemas basados en FPGA.

Propuesta del problema.

En este laboratorio se trabajará con multiplicaciones. En particular, se calcularán factoriales. Recuerde, que el factorial de un número se puede definir como:

$$n! = 1 \cdot 2 \cdot 3 \ldots (n-2) \cdot (n-1) \cdot n$$

La implementación de RISC-V que se usó en el Laboratorio 1 es capaz de hacer multiplicaciones si se le habilita el parámetro *ENABLE_MUL*. Para ello, proceda a agregar este parámetro en la instancia de picorv32.

Este parámetro habilita instrucciones de multiplicación que son parte de las extensión RV32M. Para efectos del **anteproyecto**, investigue qué instrucciones forman parte de esta extensión y describa cada una de estas instrucciones brevemente.

Ejercicio 1 (Obligatorio) (5%):

Proceda a cambiar el código del *firmware.c* para que se pueda desplegar en los LEDs el resultado del factorial de diferentes números. Por defecto el compilador (GCC) no va a agregar instrucciones que son parte de la extensión RV32M.

Por tanto, modifique el *Makefile* de *firmware.c* para que se le pase el siguiente argumento a gcc:

```
-march=rv32im
```

Verifique que su código ahora utiliza instrucciones de esta extensión. Para ello, puede utilizar la herramienta *objdump*. Esta herramienta lee los archivos ELF (ar-

chivos ejecutables y linkeables) y puede desensamblar el binario. Use el *objdump* que pertenece al toolchain de RISCV, ejecutando dentro del ambiente de Vagrant:

```
$ riscv32-unknown-elf-objdump -D firmware.elf
```

Escoja unos 3 números y muestre los resultados mediante alguna simulación que demuestre el correcto cálculo del factorial de los números escogidos.

Asimismo, calcule de la simulación cuántos ciclos de reloj en promedio le toma al CPU calcular cada uno de los factoriales. Anote los resultados del reporte de síntesis (número de slices, sumadores, LUTs, flip flops, etc).

Ejercicio 2 (Obligatorio) (20%)

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
```

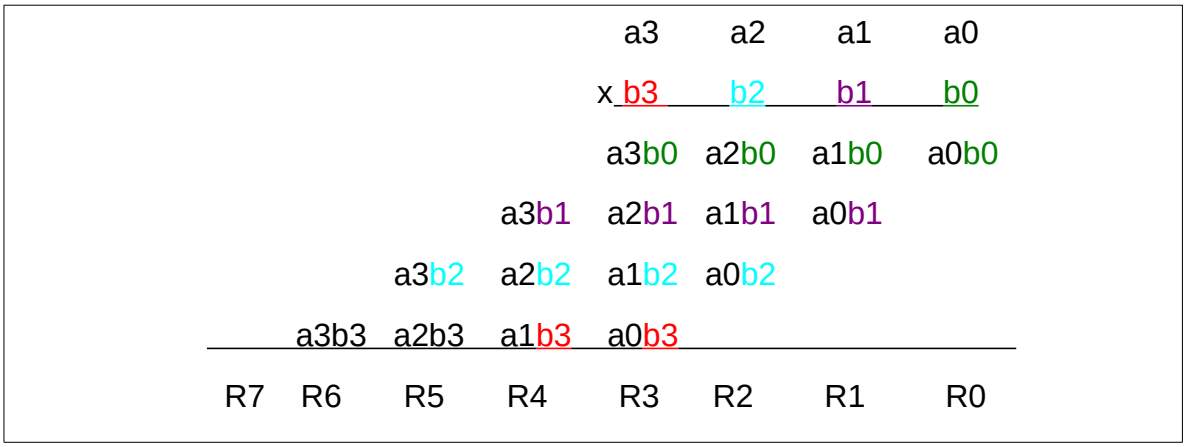
Como puede notar de la figura anterior, la mecánica de la multiplicación es la misma que en base 10. Note que a la hora de sumar se debe tomar en cuenta el acarreo.

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;
```

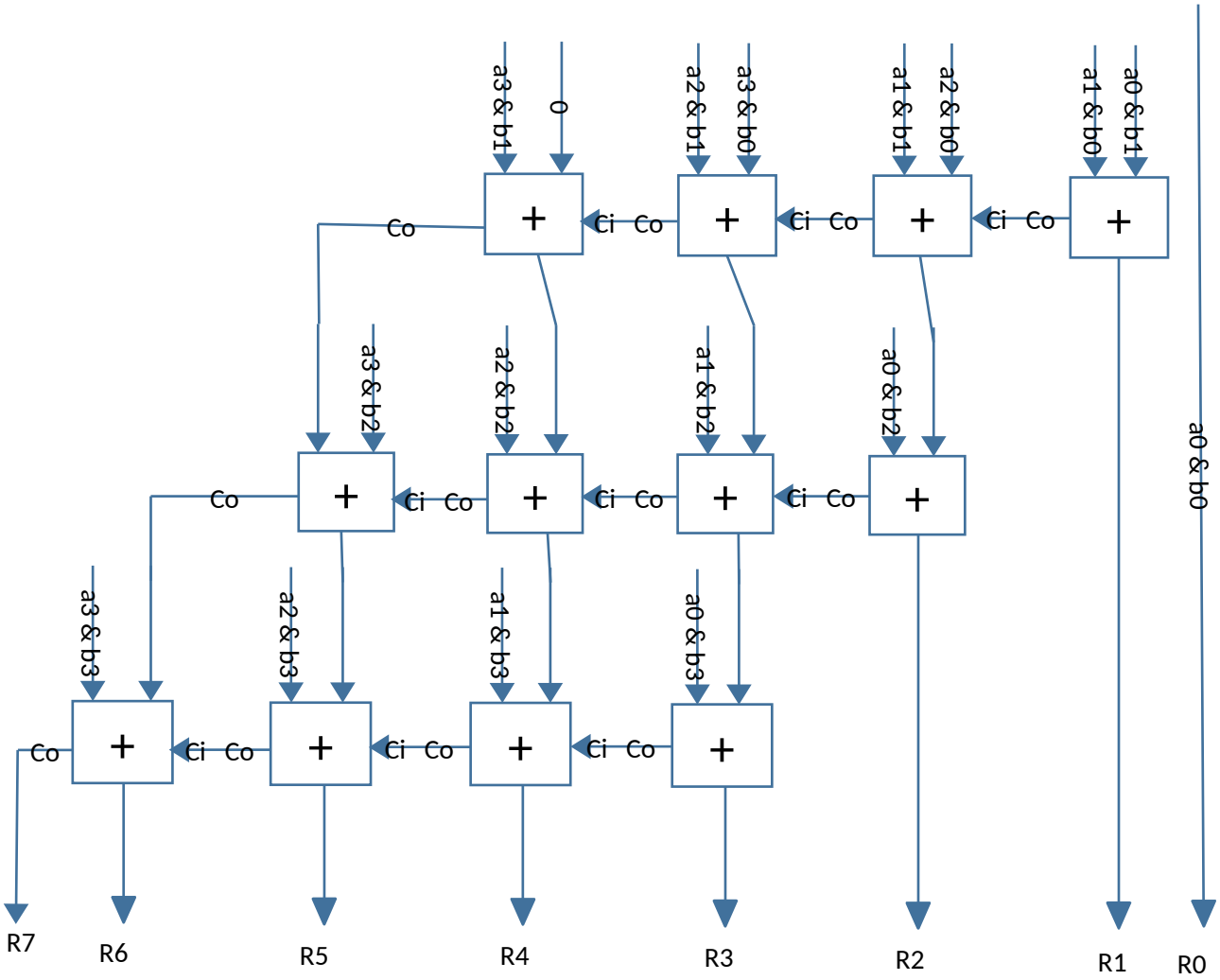
Sean dos números de 4 bits A = {a3,a2,a1,a0} y B = {b3,b2,b1,0} entonces note que la multiplicación se lleva a cabo de la siguiente manera:



De la figura anterior se puede observar lo siguiente:

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

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



Siguiendo esta lógica implemente un módulo de Verilog que multiplique dos números de 4 bits sin signo. Instancie el módulo dentro de su diseño de forma tal que se multiplique el contenido de las direcciones 0x0FFFFFF0 y 0x0FFFFFF4. El resultado se colocaría en la dirección 0x0FFFFFF8.

Pruebe la implementación escribiendo un nuevo firmware que use las direcciones antes descritas para obtener la multiplicación de varios números que escoja y lo muestre en los LEDs de la tarjeta. Como en este diseño no se utilizará el CPU para hacer la multiplicación, deshabilite el parámetro ENABLE_MUL de la instancia de picorv32.

Asimismo, utilice simulaciones para verificar su diseño.

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

Ejercicio 3 (Obligatorio) (20%)

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

Para implementar esto estudie la construcción de Verilog llamada ***generate***.

Los bloques de Verilog ***generate*** le permitirán ahorrar mucho tiempo y le enseñarán cómo 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 puede declarar arreglos de cables. Observe 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;
```

Vuelva a verificar la funcionalidad de su diseño usando las direcciones mencionadas en el ejercicio anterior para los factores y el producto de la multiplicación.

Realice algunas multiplicaciones, muestre el resultado en los LEDs y realice simulaciones. Finalmente, anote la frecuencia que la herramienta de síntesis estima para esta parte, además del número de LUTs, Slices y Flip-Flops. ¿Cómo se compara con el ejercicio 1 y 2?

¿Son los bloques **generate** sintetizables?

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

¿Qué ocurre con la frecuencia del circuito si se añaden latches entre cada etapa de sumadores?

Ejercicio 4 (Obligatorio) (20%)

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.

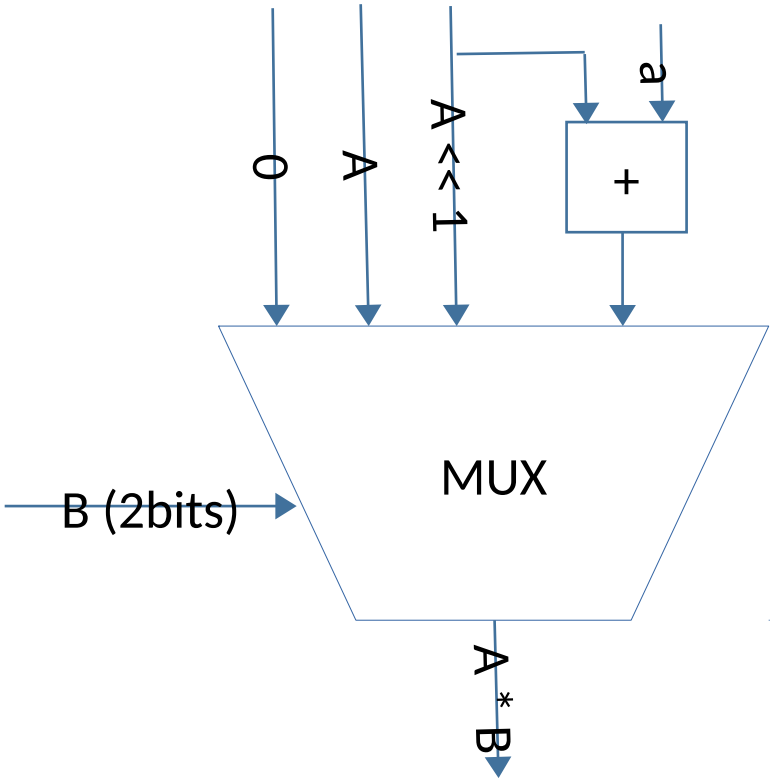
¿Cuantas filas y columnas tendría una LUT que permita multiplicar dos números de 32 bits?

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 se deberá preguntar, este MUX permite multiplicar dos 2 números de 2 bits, pero ¿qué 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 número apropiado de posiciones, se pueden multiplicar números de cualquier tamaño.

Para esta parte no se incluye un diagrama de cómo conectar los multiplexores así 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 cómo 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 números de 4 bits para comenzar.

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

Vuelva a verificar la funcionalidad de su diseño usando las direcciones mencionadas en el ejercicio 2 para los factores y el producto de la multiplicación.

Realice algunas multiplicaciones, muestre el resultado en los LEDs y realice simulaciones. Finalmente, anote la frecuencia que la herramienta de síntesis estima para esta parte, además del número de LUTs, Slices y Flip-Flops. ¿Cómo se compara con el ejercicio 1, 2 y 3?

Ejercicio 5 (Obligatorio) (35%)

Utilice alguna de las implementaciones de multiplicación anteriores de 16 bits (array multiplier o la implementación que utiliza look up tables) para calcular factoriales. Para ello, utilice la dirección 0x0FFFFFF0 para colocar el número al cual se le quiere calcular el factorial. El resultado del factorial se colocaría en la dirección 0x0FFFFFF8.

Para esta parte el cálculo del factorial puede tomar varios ciclos de reloj. Para ello, haga que su diseño use un registro de control para empezar a calcular el factorial. Para ello, puede usar la dirección 0x0FFFFFF4, de tal forma que si se le escribe un 1, su diseño deberá empezar a calcular el factorial.

Asimismo, cuando se tenga calculado el resultado, su diseño escribirá un 1 a la dirección 0x0FFFFFFC (registro de status). Esto le permitirá al programador saber cuándo el resultado que se colocará en la dirección 0x0FFFFFF8 es válido.

Escriba un firmware que use las direcciones anteriores para calcular el factorial de los mismos números que escogió para el Ejericicio 1. Note que esta vez su código, después de mandar a calcular el factorial, deberá de monitorear el registro de status (dirección 0x0FFFFFFC) para saber cuándo el resultado está listo y luego leer el resultado correcto para desplegarlo en los LEDs.

Realice simulaciones para los números que escogió y calcule de la simulación cuántos ciclos de reloj en promedio le toma al CPU calcular cada uno de los factoriales. Asimismo anote los resultados del reporte de síntesis (número de slices, sumadores, LUTs, flip flops, etc).

Compare los resultados con los obtenidos en el Ejercicio 1.