PRÁCTICA 3. Trabajo previo

Implementación de un multiplicador secuencial de números naturales.

Objetivos que se deben haber alcanzado antes de realizar la práctica

Antes de preparar esta práctica se deben haber alcanzado los objetivos específicos que se indican en la siguiente tabla. Para ello, es recomendable haber estudiado las secciones de la documentación que se indican en la primera columna de la tabla.

Secciones de la documentación a estudiar	**Objetivos específicos a alcanzar**
Capítulo 7: Secciones de la 7.1 a la 7.3.	Del 7.1 al 7.8.

Estos objetivos serán evaluados en parte del informe previo que debéis entregar al inicio de la sesión de laboratorio y en la prueba previa individual que se hará al inicio de la sesión.

3.1 Introducción

En esta práctica vamos a implementar un multiplicador **secuencial** de dos números naturales codificados en binario con 16 bits. Pero vamos a hacer el diseño poco a poco, como cuando aprendimos a multiplicar en la escuela.
¿Cómo aprendimos a multiplicar? Si recordáis, comenzamos a multiplicar números de *n* dígitos por números de un solo dígito y solamente cuando dominábamos esto, pasamos al caso general de multiplicar dos números de *n* dígitos cada uno. ¡Aquí vamos a proceder de la misma forma!

3.2 Multiplicación de un número por un dígito

Vamos a repasar lo que hacemos en decimal cuando multiplicamos un número natural por un dígito. Por ejemplo, para $n = 4$, $X = 9381$ e $Y = 8$:

	9	3	8	1
x				8
	7	5	0	4	8

¿Sabemos de memoria el resultado de la multiplicación de cualquier número de 4 dígitos por cualquier número de 1 dígito? La respuesta es ¡NO! Sólo aprendimos de memoria en la escuela la multiplicación de dos números de un dígito cada uno, o lo que es lo mismo, aprendimos la tabla de multiplicar dos dígitos decimales:

X	0	1	2	3	…	8	9
0	0	0	0	0	…	0	0
1	0	1	2	3	…	8	9
2	0	2	4	6	…	16	18
3	0	3	6	9	…	…	…
…	…	…	…	…	…	…	…
8	0	8	16	24	…	64	72
9	0	9	18	27	…	72	81

La multiplicación la realizamos aplicando un algoritmo, que avanza dígito a dígito, comenzando por el de menor peso. En cada paso efectuamos la multiplicación de un dígito del multiplicando por el único dígito del multiplicador. Dado que al multiplicar dos dígitos decimales el resultado puede requerir dos dígitos para ser representado, para obtener un dígito decimal del resultado en cada paso del algoritmo debemos pasar (o llevar) el acarreo al dígito siguiente, si es distinto de cero. En la multiplicación por un dígito en decimal, a diferencia de lo que ocurre en la suma, el acarreo puede tomar valores en el rango de 0 a 8.

Podríamos obtener el algoritmo de la multiplicación de un número por un dígito, en general para el sistema de numeración convencional en base *b*, manipulando las expresiones de la representación de un número en función de sus dígitos, como hicimos en el caso de la suma. Pero no lo vamos a hacer así porque este algoritmo es trivial para el caso particular de la representación en binario, base 2. De hecho lo vais a encontrar vosotros respondiendo a las siguientes preguntas.

- **Informe previo**

  Pregunta 1

  a) Al igual que al principio de esta sección hemos dibujado la tabla de multiplicación de dos dígitos en decimal, rellenad, ahora, la tabla de multiplicación de dos dígitos binarios:

X	0	1
0		
1		

  b) Dibujad la tabla de verdad de un circuito combinacional que multiplica dos dígitos binarios. ¿Cuántos bits de entrada y de salida tiene el circuito? Dibujad el esquema lógico del circuito en suma de minterms. A este circuito lo denominaremos "BitBit".

  c) Si entendéis por qué aparece el acarreo al multiplicar un número por un dígito decimal, a la vista de la tabla de la multiplicación de dos dígitos binarios y comparándola con la equivalente para decimal, podréis responder con éxito a la siguiente cuestión: ¿Existe acarreo en el algoritmo de multiplicación de un número por un dígito en el caso binario? Razonad la respuesta.

  d) Dado el vector de 16 bits, $X = x_{15}x_{14}\cdots x_1x_0$ y el vector de 1 bit $Y = y_0$, con $x_i, y_0 \in \{0,1\} \forall i$, que representan en binario a los números naturales X_u e Y_u ¿cuántos bits son necesarios para representar en binario el resultado de la multiplicación $X_u \times Y_u$ (o lo que es lo mismo, $X_u \times y_0$, ya que *Y* es un vector de 1 bit y el valor que representa en binario es igual al valor de su único dígito, y_0)?

  e) Dibujad el esquema lógico de un circuito combinacional para multiplicar un número natural representado en binario con 16 bits por un dígito binario. Este circuito, que denominamos MULBIT, tiene 16 bits de entrada para el vector *X*, que es la representación binaria de X_u, una entrada de un bit para el dígito y_0 y 16 bits de salida para *W*, que representa el resultado $W_u = X_u \times y_0$.

  f) ¿Cuál es la pareja entrada-salida del circuito MULBIT que has diseñado en la pregunta anterior, con un tiempo de propagación mayor? ¿Cuál es ese tiempo de propagación en función del tiempo de propagación de las puertas?

3.3 Multiplicación de dos números de n dígitos

Rango del resultado

Dados dos vectores de 16 bits, $X = x_{15}x_{14}\cdots x_1x_0$ e $Y = y_{15}y_{14}\cdots y_1y_0$ con $x_i, y_i \in \{0,1\} \forall i$, que representan en binario a los números naturales X_u e Y_u ¿cuántos bits son necesarios para representar en binario el resultado de la multiplicación de los dos números, $X_u \times Y_u$?

Como X_u e Y_u se encuentran en el rango de valores $0 \le X_u, Y_u \le 2^{16} - 1$, su multiplicación puede valer $0 \le X_u \times Y_u \le 2^{32} - 2^{17} + 1$. Esto nos indica que necesitamos 32 bits para representar los posibles valores que resulten de la multiplicación, aunque algunos valores, de los más grandes que puede representar un vector de 32 bits en binario, no se darán nunca. Generalizando esta afirmación para el

caso de operandos representados con n bits, podemos afirmar que el resultado correcto de la multiplicación binaria requiere 2n bits.

Planteamiento del problema

Dados dos vectores de n bits, $X = x_{n-1}x_{n-2}\cdots x_1x_0$ e $Y = y_{n-1}y_{n-2}\cdots y_1y_0$ con $x_i, y_i \in \{0,1\}\,\forall i$, que representan en binario a los números naturales X_u e Y_u, debemos obtener un algoritmo que produzca los *2n* bits del vector $W = w_{2n-1}w_{2n-2}\cdots w_1w_0$ con $w_i \in \{0,1\}\ \forall i$, que representa en binario al número natural $W_u = X_u \times Y_u$.

3.3.1 Un algoritmo secuencial

Vamos a obtener un algoritmo secuencial en el que en cada paso se multiplica el multiplicando X_u por un dígito, y_i, del multiplicador. Va a resultar un algoritmo parecido al que realizamos cuando multiplicamos en decimal. Por ejemplo:

```
          4 3 7 5
        x 4 6 3 9
  ---------------
        3 9 3 7 5
      1 3 1 2 5
    2 6 2 5 0
  1 7 5 0 0
  ---------------
  2 0 2 9 5 6 2 5
```

Lo mismo que hacemos en decimal lo podemos hacer en binario, con la diferencia de que los dígitos valen {0, 1} en vez de {0,1,…9} y esto implica diferencias en los acarreos al multiplicar el multiplicando por un dígito del multiplicador y al sumar los resultados de las multiplicaciones por cada dígito. Un ejemplo en binario es:

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

- **Informe previo**
  Pregunta 2

  a) Expresad en decimal el valor que representan en binario los siguientes vectores de bits: *X* = 1101, *Y* = 1011, *W* = 10001111, esto es ¿cuánto vale X_u, Y_u y W_u? ¿Es correcto el resultado de la multiplicación de números naturales en binario que hemos hecho en el párrafo anterior?
  b) Realizad, en binario, la multiplicación de los números naturales X_u = 23 e Y_u = 17 y comprobad, pasando el resultado a decimal, que habéis realizado la multiplicación correctamente.

Aunque ya sabemos multiplicar en binario, ahora vamos a obtener el algoritmo de la multiplicación a partir de las fórmulas que nos dan el valor representado por un vector de bits y demostraremos que el algoritmo que acabamos de usar es correcto. Además, a partir de las expresiones matemáticas, o del algoritmo expresado formalmente, nos resulta relativamente fácil diseñar el circuito que lo implementa. Para ello partimos de la expresión del resultado W_u y sustituimos el multiplicador por su valor en función de los dígitos que lo representan:

$$W_u = X_u \times Y_u = X_u \times \left(\sum_{j=0}^{n-1} y_j \times 2^j \right)$$

Manipulando esta expresión obtenemos:

$$W_u = X_u \times \left(\sum_{j=0}^{n-1} y_j \times 2^j \right) = \sum_{j=0}^{n-1} ((X_u \times 2^j) \times y_j)$$

De la última igualdad de esta ecuación obtenemos el siguiente algoritmo secuencial, donde $W_u(j)$ va acumulando en cada iteración *j* del algoritmo el resultado de multiplicar el multiplicando por un dígito del multiplicador. Hemos utilizado las variables intermedias D_u y M_u para separar las tres operaciones que requiere cada iteración del bucle:

$W_u(0) = 0$;
for (*j* = 0; j < *n*; j = j + 1) **{**
    $D_u = X_u \times 2^j$;
    $M_u = D_u \times y_j$;
    $W_u(j+1) = W_u(j) + M_u$;
}
$W_u = W_u(n)$;

Por último, transformamos el algoritmo para ahorrarnos multiplicar en cada iteración X_u por 2^j (ahorrarnos multiplicar por una potencia distinta de 2, dependiendo de la iteración en la que nos encontremos). En vez de esto calcularemos $D_u = X_u \times 2^j$ mediante la recurrencia:

$D_u(0) = X_u$;
for (*k* = 0; k < j; k = k + 1) {
    $D_u(k+1) = D_u(k) \times 2$;
}
$D_u = D_u(j)$;

Insertando el cálculo de esta recurrencia en el mismo bucle del algoritmo de multiplicación, tenemos que ahora en todas las iteraciones multiplicamos por el valor 2 independientemente de la iteración en la que estemos:

Algoritmo MUL

$D_u(0) = X_u$;
$W_u(0) = 0$;
for (*j* = 0; j < *n*; j = j + 1) **{**
    $M_u = D_u(j) \times y_j$;
    $W_u(j+1) = W_u(j) + M_u$;
    $D_u(j+1) = D_u(j) \times 2$;
}
$W_u = W_u(n)$;

Este algoritmo es parecido al que usamos en decimal con *lápiz y pape*l (y al que hemos usado en binario al principio de esta sección). La diferencia consiste en que con lápiz y papel efectuamos los cálculos en dos fases. La primera fase tiene *n* (n es el número de dígitos) iteraciones o pasos y en cada iteración calculamos la multiplicación del multiplicando por un dígito del multiplicador y desplazamos el resultado adecuadamente para prepararlo para la suma de la segunda fase (desplazar una posición a la izquierda un vector de dígitos que representa un número natural en el sistema convencional en base *b* es multiplicar el número por *b, como ya sabemos*). En la segunda fase efectuamos la suma de los *n* resultados parciales de la primera fase.

Sin embargo, en el algoritmo que acabamos de encontrar las dos fases se efectúan entremezcladas en una única fase. El algoritmo completo consiste en *n* iteraciones o pasos. En cada iteración se calcula un resultado parcial sumando al resultado parcial de la iteración anterior el resultado de multiplicar el multiplicando por un dígito del multiplicador y desplazarlo adecuadamente.

Vamos a hacer un seguimiento del algoritmo en una sola fase para dos números concretos $X_u = 18$ e $Y_u = 13$. Representamos el multiplicador con, por ejemplo, *n* = 8 bits, por lo que el algoritmo tendrá 8 iteraciones o pasos. El vector de bits que representa Y en binario es Y = 00001101. El resultado de la multiplicación debe ser $W_u = 234$. La siguiente tabla muestra los valores numéricos y el bit de Y involucrados en cada iteración del algoritmo. Vemos que el resultado final es correcto, $W_u = 234$.

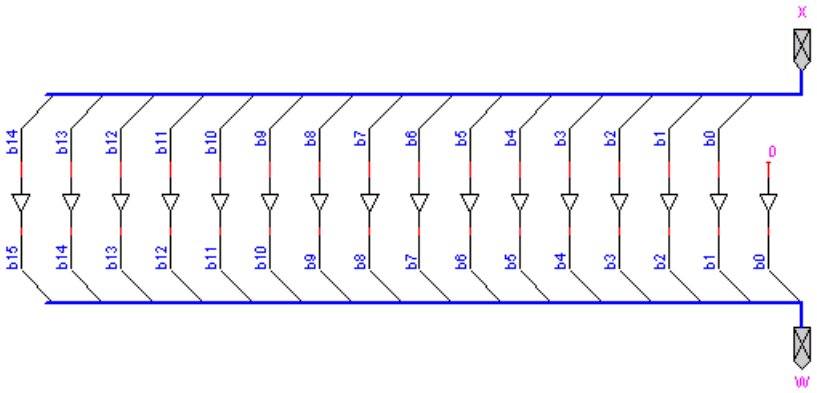
Estado inicial		$W_u(0) = 0$	$D_u(0) = 18$	
Iteración j	$M_u = D_u(j) \times y_j$	$W_u(j+1) = W_u(j) + M_u$	$D_u(j+1) = D_u(j) \times 2$	y_j
0	18	18	36	1
1	0	18	72	0
2	72	90	144	1
3	144	234	288	1
4	0	234	576	0
5	0	234	1152	0
6	0	234	2304	0
7	0	234	4608	0
Resul. Final W_u		234		

3.3.2 Implementación secuencial del multiplicador binario

Vamos a implementar el algoritmo anterior para el caso de $n = 16$. Como vimos al principio, si X_u e Y_u están representados en binario con *n* bits, necesitamos 2*n* bits para que el resultado de la multiplicación sea siempre representable. Dado que en un computador las instrucciones operan con datos de *n* bits y los registros donde se guardan los resultados son de *n* bits, tiene sentido estar interesados únicamente en los *n* bits de menor peso del resultado de la multiplicación. Sabemos, claro está, que con esos *n* bits no siempre tendremos el resultado correcto de la multiplicación. Así, cuando X_u y/o Y_u sean números grandes, se puede producir un resultado incorrecto. Vamos a implementar un multiplicador que **sólo calcula los *n* bits de menor peso del resultado**. Además, para simplificar, **no vamos a implementar la detección de resultado no representable con n bits,** aunque os haremos una pregunta al respecto.

Como sólo queremos los *n* bits de menor peso de la representación de $W_u = X_u \times Y_u$, podemos usar únicamente *n* bits para codificar en binario cada uno de los resultados de las operaciones del algoritmo en cada iteración: M_u, $W_u(j+1)$ y $D_u(j+1)$.

Conocemos algoritmos y sabemos implementar circuitos lógicos combinacionales para las tres operaciones que hay que realizar en cada iteración del algoritmo. La operación $M_u = D_u(j) \times y_j$ la realizaremos con el circuito MULBIT que habéis diseñado en la pregunta 1 del informe previo y la operación $W_u(j+1) = W_u(j) + M_u$ la podemos realizar con el sumador en propagación del acarreo que estudiamos en el tema 3 y que diseñasteis en la práctica 2 y ahora se encuentra encapsulado en el dispositivo ADD de la librería de esta práctica, LibPrac3. La operación $D_u(j+1) = D_u(j) \times 2$ también la estudiamos en el tema 2. Denominamos SL-1, del inglés *Shift Left*, al circuito combinacional para multiplicar por 2 que se muestra en la siguiente figura. Como estamos interesados en los *n* bits de menor peso de la multiplicación, aun sabiendo que el resultado puede no ser correcto, nuestro circuito sólo calculará los n bits de menor peso.



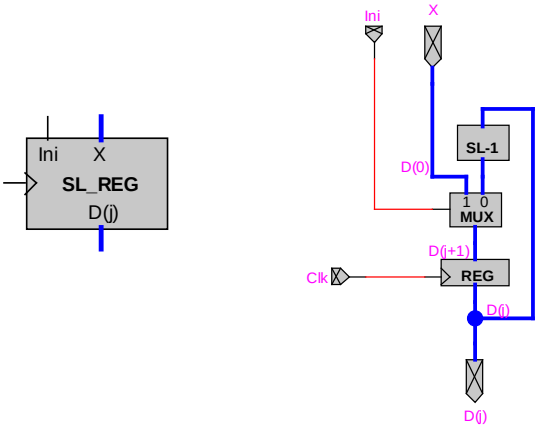
Los triángulos son dispositivos denominados Buffers, que a nivel lógico es como si no estuvieran. La salida es igual que la entrada y el tiempo de propagación es cero. Se usa porque LogicWorks no deja conectar directamente un cable de salida de un Breakout con un cable de entrada de otro Breakout que tengan distinto nombre.

La implementación del algoritmo de multiplicación es sencilla, ya que tenemos dispositivos combinacionales para cada una de las operaciones que se realizan en cada iteración del algoritmo. Los cálculos de cada iteración del algoritmo se realizarán durante un ciclo de reloj del circuito: iteración es a algoritmo como ciclo a circuito. Para almacenar el valor $W_u(j)$ y $D_u(j)$ durante el ciclo *j* utilizaremos dos registros de *n* bits. La entrada de datos de estos dos registros durante el ciclo *j* será respectivamente $W_u(j+1) = W_u(j) + M_u$ y $D_u(j+1) = D_u(j) \times 2$. Cuando llegue el flanco ascendente del reloj marcando el final del ciclo *j* y el inicio del *j*+1, estos valores serán cargados en los registros. La llegada del flanco de reloj es equivalente a comenzar una nueva iteración del algoritmo.

Unidad de Proceso

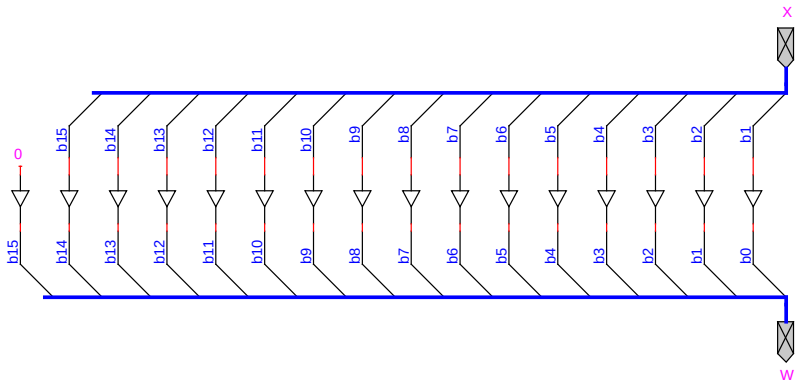
Vamos a diseñar el multiplicador en dos partes, primero diseñamos la unidad de proceso y luego diseñaremos la unidad de control. La unidad de proceso calcula los n bits de menor peso del resultado (sin detección de resultado no representable en *n* bits) siguiendo el algoritmo secuencial que hemos obtenido. La construimos con 3 bloques secuenciales y uno combinacional que definimos a continuación:

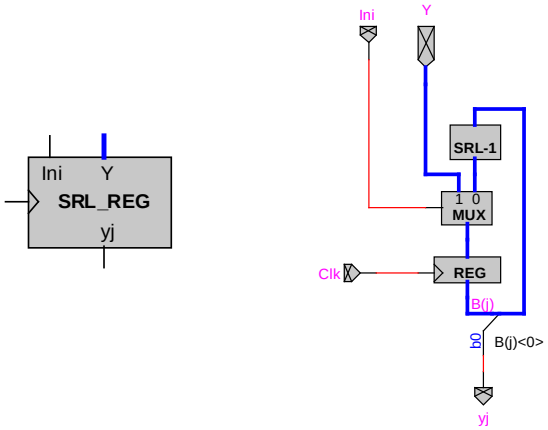
- El dispositivo secuencial SL_REG (Shift Left Register) es un registro con carga paralela (carga X al final del ciclo durante el que Ini vale 1) y desplazamiento de un bit a la izquierda (cuando Ini vale 0). Lo utilizamos para calcular $D_u(j+1) = D_u(j) \times 2$ para cada una de las iteraciones del algoritmo. El bloque que lo implementa y su circuito interno son los siguientes:



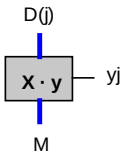
- El dispositivo secuencial SRL_REG (Shift Right Logic Register) es un registro con carga paralela (cuando Ini vale 1) y desplazamiento lógico a la derecha (cuando Ini vale 0). La salida es el bit de menos peso del dato almacenado en el registro. Como a cada ciclo se desplaza una posición a la derecha el dato almacenado, k ciclos después de que Ini valga 1, el bit de menor peso del dato almacenado es el bit k del dato original que se cargo cuando Ini valía 1. Esta parte del circuito se usa para obtener el bit y_j en cada ciclo, desde *j = 0* hasta *n-1*. El circuito combinacional SRL-1

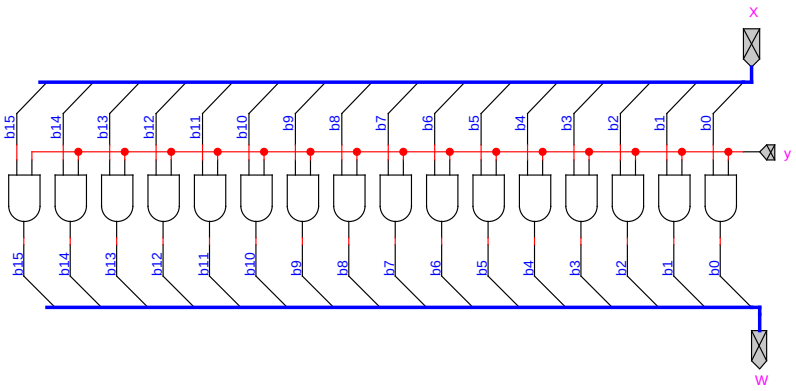
(que forma parte del circuito SRL_REG) y el circuito a bloques que implementa el SRL_REG son los siguientes:



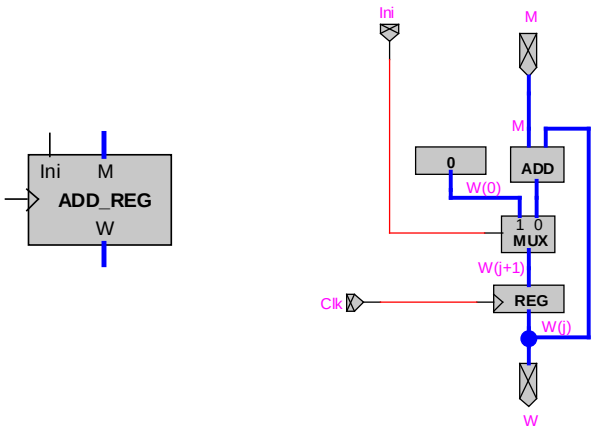


- El dispositivo combinacional MULBIT, cuyo símbolo esta etiquetado con el mnemotécnico X·y, calcula $M_u = D_u(j) \times y_j$. El bloque que lo implementa y su circuito interno son los siguientes (podéis comprobar ahora que respondisteis bien a la pregunta 1 del informe previo):

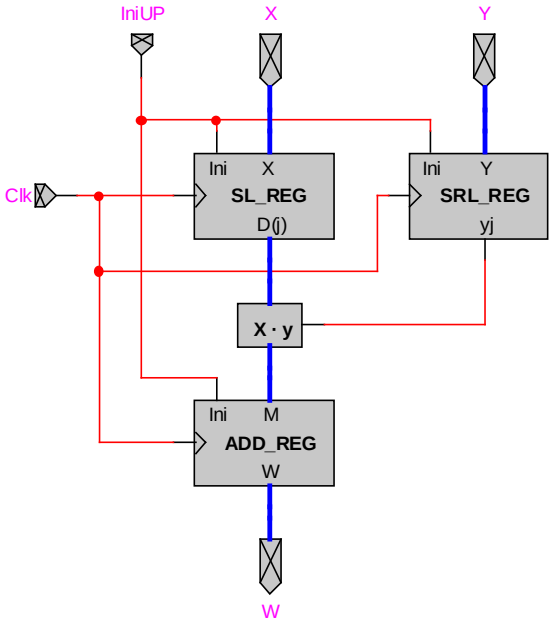




- El dispositivo ADD_REG es un registro acumulador que se inicializa a cero (al final del ciclo en el que Ini vale 1) y que suma el contenido del registro con el valor de la entrada M y deja el resultado en el registro (cuando Ini vale 0). Este dispositivo calcula $W_u(j+1) = W_u(j) + M_u$ para cada iteración. El bloque que lo implementa y su circuito interno se muestran a continuación. Los 16 bits de salida del símbolo etiquetado con el mnemotécnico 0 valen siempre todos 0; está construido con un breakout de 16 bits, con cada uno de los 16 bits fijados al valor 0 y es necesario para inicializar el registro acumulador al valor 0.



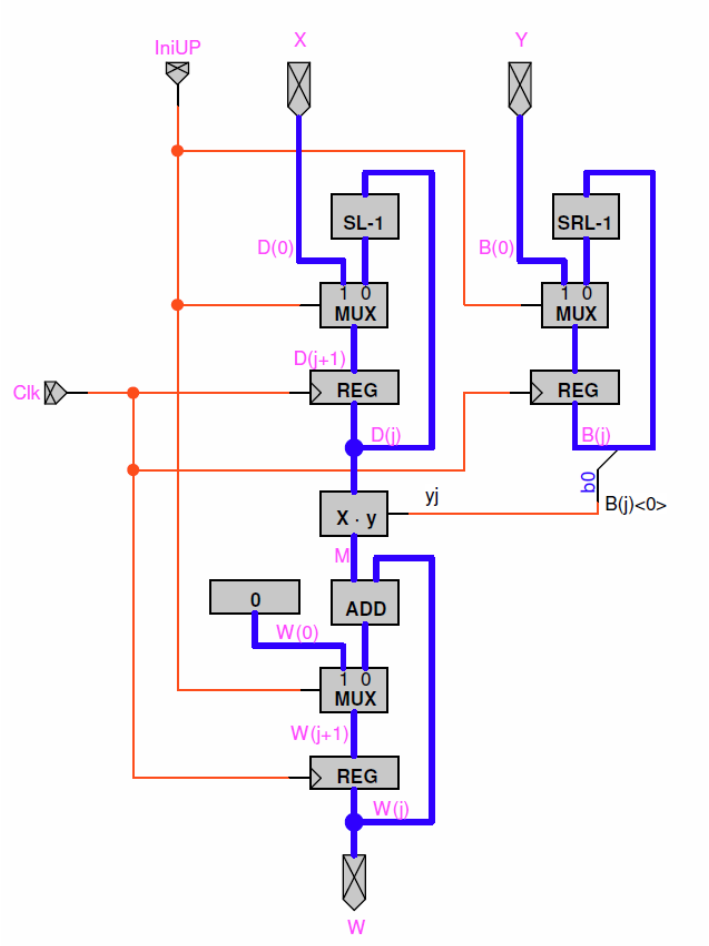
La siguiente figura muestra la interconexión de estos cuatro bloques para formar la unidad de proceso del multiplicador.



La unidad de control, que definimos más adelante, es la encargada de activar la señal IniUP, que sirve para cargar los registros de los bloques SL_REG, SRL_REG y ADD_REG con los valores iniciales, antes de comenzar las *n* iteraciones del algoritmo.

Una vez entendido lo que hace cada bloque secuencial y cómo lo hace internamente, es más útil para lo que sigue de la práctica ver la unidad de proceso en un solo nivel, viendo a la vez el contenido de los tres bloques secuenciales que acabamos de diseñar. Esto se muestra en la siguiente figura.

Los nombres que etiquetan los buses del circuito hacen referencia a los vectores de 16 bits que representan en binario a las variables del algoritmo (sólo las representan correctamente cuando con 16 bits se puede representar el valor correcto de la variable).



Algoritmo aritmético

Para reforzar la comprensión del algoritmo de multiplicación y para comprobar su correcto funcionamiento a nivel de operaciones con bits vamos a realizar un seguimiento del algoritmo aritmético de la multiplicación en binario para dos vectores de bits concretos. Un algoritmo aritmético obtiene los bits del vector W, que representa en binario el valor del resultado, mediante operaciones con los bits de los operandos, vectores X e Y. El algoritmo que expresamos a continuación se obtiene del algoritmo MUL, que usa valores numéricos en vez de vectores de bits, y de la implementación del algoritmo que estamos construyendo. Usamos la notación $B(j)<0>$ para referirnos al bit 0, bit de menos peso, del vector de bits $B(j)$, que es el vector B en la iteración o ciclo j. El algoritmo sólo obtiene los *n* bits de menor peso del resultado, o sea, que los *n* bits de W representan a Wu solamente cuando Wu se puede representar con *n* bits.

Algoritmo aritmético MUL

$D(0) = X$;
$B(0) = Y$;
$W(0) = 0,0,...,0$;
for (j = 0; j < n; j = j + 1) **{**
  $M = MULBIT(D(j), B(j)<0>)$;
  $W(j+1) = ADD(W(j), M)$;
  $D(j+1) = SL-1(D(j))$;
  $B(j+1) = SRL-1(B(j))$;
}
$W = W(n)$;

Aunque en esta práctica *n* vale 16, para simplificar, vamos a hacer un seguimiento del algoritmo aritmético para un par de vectores concretos con *n* = 8 bits: X = 00010010 e Y = 00001101. Estos vectores representan en binario a los números naturales $X_u = 18$ e $Y_u = 13$, por lo que el resultado de su multiplicación debe ser $W_u = 234$. Como 234 se puede representar en 8 bits, el resultado que obtendremos será correcto. La siguiente tabla muestra los vectores de bits involucrados en el algoritmo para cada una de las iteraciones (ciclos de reloj en la implementación que estamos realizando).

Estado inicial		W(0) = 0 0 0 0 0 0 0 0	D(0) = 0 0 0 1 0 0 1 0	B(0) = 0 0 0 0 1 1 0 1
Iteración / ciclo j	M = MULBit (D(j), B(j)<0>)	W(j+1) = ADD(W(j), M)	D(j+1) = SL-1(D(j))	B(j+1) = SRL-1(B(j))
0	0 0 0 1 0 0 1 0	0 0 0 1 0 0 1 0	0 0 1 0 0 1 0 0	0 0 0 0 0 1 1 0
1	0 0 0 0 0 0 0 0	0 0 0 1 0 0 1 0	0 1 0 0 1 0 0 0	0 0 0 0 0 0 1 1
2	0 1 0 0 1 0 0 0	0 1 0 1 1 0 1 0	1 0 0 1 0 0 0 0	0 0 0 0 0 0 0 1
3	1 0 0 1 0 0 0 0	1 1 1 0 1 0 1 0	0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0
4	0 0 0 0 0 0 0 0	1 1 1 0 1 0 1 0	0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0
5	0 0 0 0 0 0 0 0	1 1 1 0 1 0 1 0	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
6	0 0 0 0 0 0 0 0	1 1 1 0 1 0 1 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
7	0 0 0 0 0 0 0 0	1 1 1 0 1 0 1 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
Resul. Final W		1 1 1 0 1 0 1 0		

El resultado final W codifica en binario el resultado correcto de la multiplicación, $W_u = 234$.

- **Informe previo**

  Pregunta 3

  Efectuad el seguimiento del algoritmo aritmético de multiplicación que obtiene los 8 bits de menor peso del resultado para los valores numéricos $X_u = 22$ e $Y_u = 77$. Para ello, rellenad la siguiente tabla, tal como se ha hecho en el ejemplo anterior.

Estado inicial		W(0) =	D(0) =	B(0) =
Iteración / ciclo j	M = MULBit (D(j), B(j)<0>)	W(j+1) = ADD(W(j), M)	D(j+1) = SL-1(D(j))	B(j+1) = SRL-1(B(j))
0				
1				
2				
3				
4				
5				
6				
7				
Resul. Final W				

¿Cuál es el resultado correcto de la multiplicación, $W_u = X_u \times Y_u$?

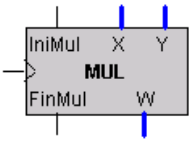
¿Los 8 bits que se obtienen como resultado del algoritmo anterior, representan el resultado correcto de la multiplicación? ¿Por qué?

Ahora continuamos con el diseño del multiplicador. Para que el multiplicador funcione correctamente hay que generar la entrada de control *IniUP* correctamente. De esto se encarga la Unidad de Control.

Este multiplicador es un Procesador de Propósito Específico, que estará inmerso en un sistema más complejo que le dará los datos a multiplicar de manera síncrona (sincronizados con su mismo reloj) y recogerá el resultado para usarlo en otra parte del sistema, también de manera síncrona.

Especificación del multiplicador como caja negra

Vamos a especificar el funcionamiento del multiplicador (unidad de proceso más unidad de control) como caja negra (desde el punto de vista de sus entradas y salidas y sin hacer mención de la implementación interna). Al multiplicador entran dos buses de entrada de datos de 16 bits *X* e *Y* que codifican en binario X_u e Y_u y sale un bus de 16 bits W que codifica en binario el resultado de la multiplicación $W_u = X_u \times Y_u$ cuando este se puede representar con *n* bits (es decir, cuando $X_u \times Y_u \leq 2^{16} - 1$). Hay dos señales de control de un bit, una de entrada, *IniMul*, y otra de salida, *FinMul*. Además, como todo circuito secuencial, tiene una entrada de reloj. El dispositivo que implementa este multiplicador se muestra a continuación.

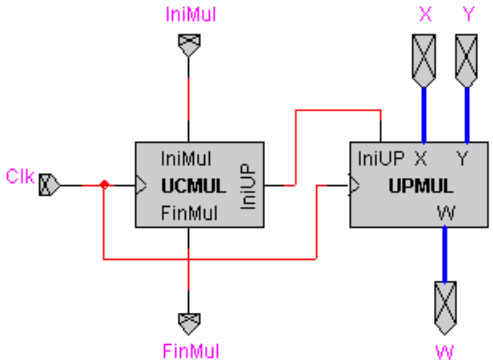


El funcionamiento del multiplicador como caja negra debe ser el siguiente:

- En el ciclo en el que la señal *IniMul* vale 1, los datos de los buses de entrada *X* e *Y* son válidos y se debe comenzar su multiplicación.
- Los datos sólo están un ciclo en la entrada, el mismo ciclo en el que *IniMul* vale 1.
- Cuando el multiplicador muestra el resultado correcto en el bus de salida, lo indicará poniendo a 1 durante un ciclo la señal de control FinMul.
- El resultado estará presente en el bus de salida del multiplicador durante ese ciclo, así que el sistema exterior debe estar esperando este ciclo, para no perder el resultado.
- Si mientras se está multiplicando un par de números (después del ciclo en el que *IniMul* vale 1 y antes de que *FinMul* valga 1), se recibe por la entrada *IniMul* un 1, el multiplicador hará caso omiso de esta petición y continuará sin inmutarse hasta que ponga a 1 *FinMul*.
- En el ciclo en el que *FinMul* vale 1, ya puede comenzar otra multiplicación, si *IniMul* vale 1 en ese mismo ciclo.

Unidad de Proceso y Unidad de Control

En la siguiente figura se muestra el circuito interno del dispositivo MUL formado por dos dispositivos: la Unidad de Control (UCMUL) y la Unidad de Proceso (UPMUL).



El circuito interno de la unidad de proceso es el que ya hemos diseñado anteriormente. Ahora vamos a diseñar la unidad de control del multiplicador. Observad que de la UPMUL que acabamos de diseñar no se genera ninguna señal que indique que la multiplicación ha terminado. Esto no lo sabe la UPMUL. Así pues, tenemos que diseñar la UCMUL para que mientras está esperando que IniMul valga 1, active la señal IniUP para inicializar el sistema y cargar los operandos de la multiplicación, después cuente los 16 ciclos (iteraciones) del algoritmo y como pasados esos 16 ciclos sabe que el resultado ya está disponible en el bus W, genere la señal FinMul y se disponga a esperar los operandos de una nueva multiplicación.

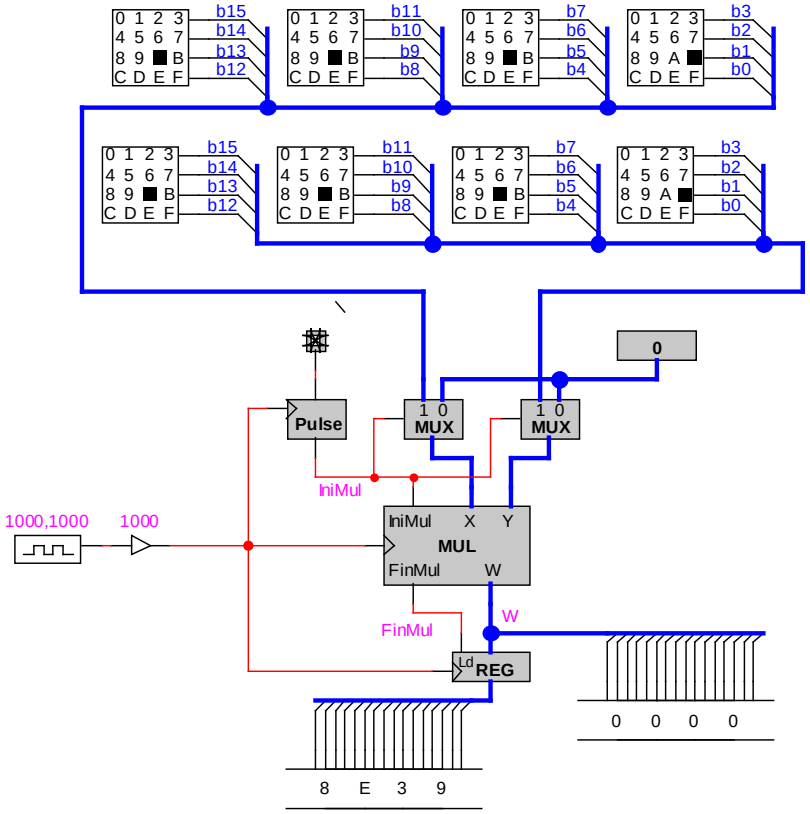
Unidad de Control

- **Informe previo**

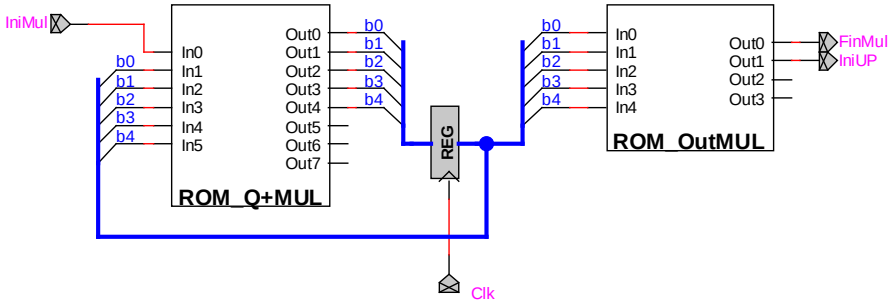
  Pregunta 4

  Dibujad el grafo de estados de Moore de la Unidad de Control del multiplicador de dos números para que funcione con la Unidad de Proceso que hemos diseñado tal como se ha especificado. (Recordad el pequeño grafo de estados que implementasteis en la práctica 2, ya que el grafo que tenéis que hacer ahora es bastante parecido, también cuenta, aunque este tiene 18 estados y distintas señales de entrada y salida).

En el laboratorio tenéis el circuito ProbeMUL que hemos construido para vosotros, es una implementación completa del multiplicador que hemos visto aquí junto con los dispositivos para entrar datos y visualizar el resultado. El esquema a bloques es el que se muestra a continuación.



A continuación se muestra el esquema interno de la unidad de control con el número mínimo de biestables y dos memorias ROM.



Observad que de los 4 bits (un dígito hexadecimal) de menor peso de la salida de la ROM_OutMUL, solamente usamos los dos bits de menor peso, para codificar las señales IniUP y FinMul. Hemos usado un dígito hexadecimal en vez de dos bits, para que sea más fácil escribir el contenido de la ROM en hexadecimal, aunque se necesite una ROM con 2 bits más por palabra. Lo mismo hemos hecho en la ROM que obtiene el estado siguiente, ROM_Q+MUL, que sólo necesita 5 bits de salida y hemos usado palabras de 8 bits (dos dígitos hexadecimales), de los que los 3 de más peso quedan sin utilizar.

- **Informe previo**

  Pregunta 5

  Indicad el contenido de cada una de las dos memorias ROM, para que el multiplicador funcione como se ha especificado (y de acuerdo con el grafo de 18 estados que habéis diseñado en la pregunta 4 del informe previo). Codificad el estado inicial con el 0, cinco bits a cero, ya que así no hace falta hacer nada especial para inicializar el registro de estado, que por defecto se inicializa a 0). Codificad el resto de estados del 1 al 17 ya que así sólo tenéis que especificar las 36 primeras palabras de la ROM del estado siguiente y las 18 primeras de la ROM de las salidas. A las otras posiciones de las ROMs no hace falta especificarlas ya que el circuito nunca leerá esas posiciones (su contenido puede ser cualquiera). Los bits que no se usan de cada palabra de las ROM haced que sean siempre 0 (bits 5, 6 y 7 de la ROM del estado siguiente y bits 2 y 3 de la ROM de las salidas). Expresad el contenido de cada una de las dos ROM en hexadecimal, separando cada palabra por un espacio o un cambio de línea acordando que la primera palabra corresponde a la palabra con dirección 0, la siguiente a la de dirección 1, etc. Usad el espacio siguiente.

ROM_Q+MUL

ROM_OutMUL

Tiempo de ciclo

Dada la importancia del tiempo de ciclo mínimo adecuado para que un circuito secuencial funcione correctamente, es importante que repaséis el tema 5.

Para responder a la pregunta siguiente del informe final es necesario saber el tiempo de propagación del sumador ADD que está inmerso en el circuito. Aunque ya calculamos el tiempo de propagación del ADD en la práctica anterior, se hizo suponiendo el caso peor de cambio en los datos y que el acarreo de entrada podía valer 1. Sin embargo, en este sumador el acarreo de entrada es siempre 0 y además

- el operando de la entrada de la derecha del sumador vale 0 la primera vez que se suma, cuando en el operando de la izquierda está Y;
- en la segunda suma el bit de menor del operando de la izquierda del sumador vale 0, pues el vector X se ha desplazado una posición a la izquierda,
- en la tercera suma los dos bits de menos peso del operando de la izquierda valen 0 y así sucesivamente.

Por ello, el camino más largo de los acarreos por el sumador se produce la segunda suma para los vectores de bits X = 0xAAAB e Y = 0xAAAB, que el acarreo se genera en el Fa 1, se propaga por los Fa del 2 al 14 y finalmente sale por la Xor de la salida s del Fa 15 (el camino crítico en este caso no pasa por el Fa 0). Podemos asegurar que para los valores de la puerta Not-1, And-2 y Or-2 de 10, 20 y 20 u.t. respectivamente, en este algoritmo de multiplicación y para los valores de X e Y que requieren más tiempo de propagación, **el tiempo de propagación del ADD es de 610 u.t.** Este tiempo se obtiene de la suma de:

- 40 u.t. de las puertas And-2 y Or-2 de la generación del acarreo en el Fa 1,
- 40 x 13 = 520 u.t. de las puertas And-2 y Or-2 de la propagación del acarreo por los Fa del 2 al 14 y
- 50 u.t. de la puerta Xor-2 que genera la salida s del Fa 15.

- **Informe previo**

  Pregunta 6

  Considerando,

  - que la puerta Not y las puertas And-2 y Or-2 tienen unos tiempos de propagación de 10, 20 y 20 u.t. (unidades de tiempo) respectivamente y que el Flip-Flop (biestable D activado por flanco), con el que están hechos los registros, tiene un tiempo de propagación de 100 u.t. (desde que llega el flanco ascendente de reloj hasta que la salida del biestable tiene el valor estable, pasan 100 u.t.),
  - que el tiempo de propagación del bloque ADD es de 610 u.t.,

- que cada ROM de la Unidad de Control tiene un tiempo de acceso de 200 u.t. (tiempo desde que las entradas de dirección están estables hasta que lo están los datos en la salida),
- que la señal IniMul y los buses de entrada al multiplicador X e Y vienen directamente de la salida de un Flip-flop y de dos registros de otro subsistema, y
- que la señal FinMul y el bus de salida W del multiplicador van directamente a la entrada de un Flip-Flop y un registro de otro subsistema,

responded a las siguientes preguntas.

a) ¿A través de qué dispositivos pasa el camino crítico del multiplicador?

b) ¿Cuál es el tiempo de ciclo mínimo para que el multiplicador funcione correctamente? Justifica la respuesta indicando los tiempos de propagación de cada uno de los dispositivos del camino crítico.

Informe previo Práctica-3

Apellidos y nombre: ..... Grupo: .....

Apellidos y nombre: ..... Grupo: .....

(por orden alfabético)

Pregunta 1

a)

X	0	1
0		
1		

b)

c)

d)

e)

f)

Pregunta 2

a)

b)

Pregunta 3

Estado inicial		W(0) =	D(0) =	B(0) =
Iteración / ciclo j	M = MULBit (D(j), B(j)<0>)	W(j+1) = ADD(W(j), M)	D(j+1) = SL-1(D(j))	B(j+1) = SRL-1(B(j))
0				
1				
2				
3				
4				
5				
6				
7				
Resul. Final W				

¿Cuál es el resultado correcto de la multiplicación, $W_u = X_u \times Y_u$?

¿Los 8 bits que se obtienen como resultado del algoritmo anterior, representan el resultado correcto de la multiplicación?

¿Por qué?

Pregunta 4

Pregunta 5

ROM_Q+MUL

ROM_OutMUL

Pregunta 6

a)

b)

