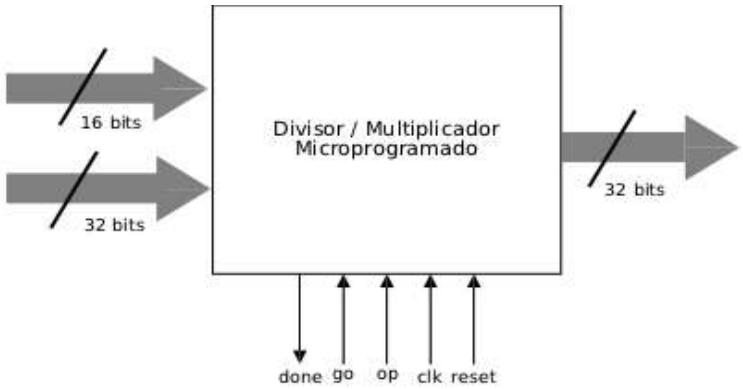
Robert Quirós	**A03347**
Roberto Herrera	**A02057**
Marco V. Chacón	**A01094**
Circuitos Digitales II	Prof. Randolph Steinvorth
Proyecto No. 3	Divisor / Multiplicador microprogramado 25 - noviembre - 2005

Especificación del diseño:

Describir un sistema divisor / multiplicador de números binarios microprogramado. El multiplicador utilizará como operandos dos números de 16 bits; el divisor utilizará un divisor de 16 bits y un dividendo de 32 bits. Por medio de las señales done y go, el sistema utilizará un protocolo tipo handshake para controlar el flujo de datos que entran y salen. La señal de control op selecciona el tipo de operación que se desea realizar. Los datos deben entrar por dos puertos, uno para cada operando, los puertos serán los mismos sin importar la operación que se vaya a realizar. En el caso de el puerto correspondiente el dividendo, cuando se esté leyendo de él uno de los multiplicandos, se tomarán los 16 bits menos significativos. Similarmente, el cociente de la división se colocará en los bits menos significativos del puerto de salida. Un diagrama de bloques de alto nivel se observa a continuación:



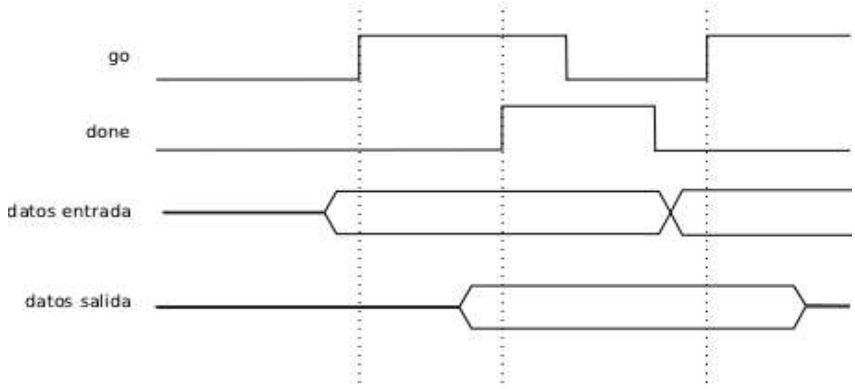


Diagrama de temporización para las señales del protocolo.

Descripción del diseño:

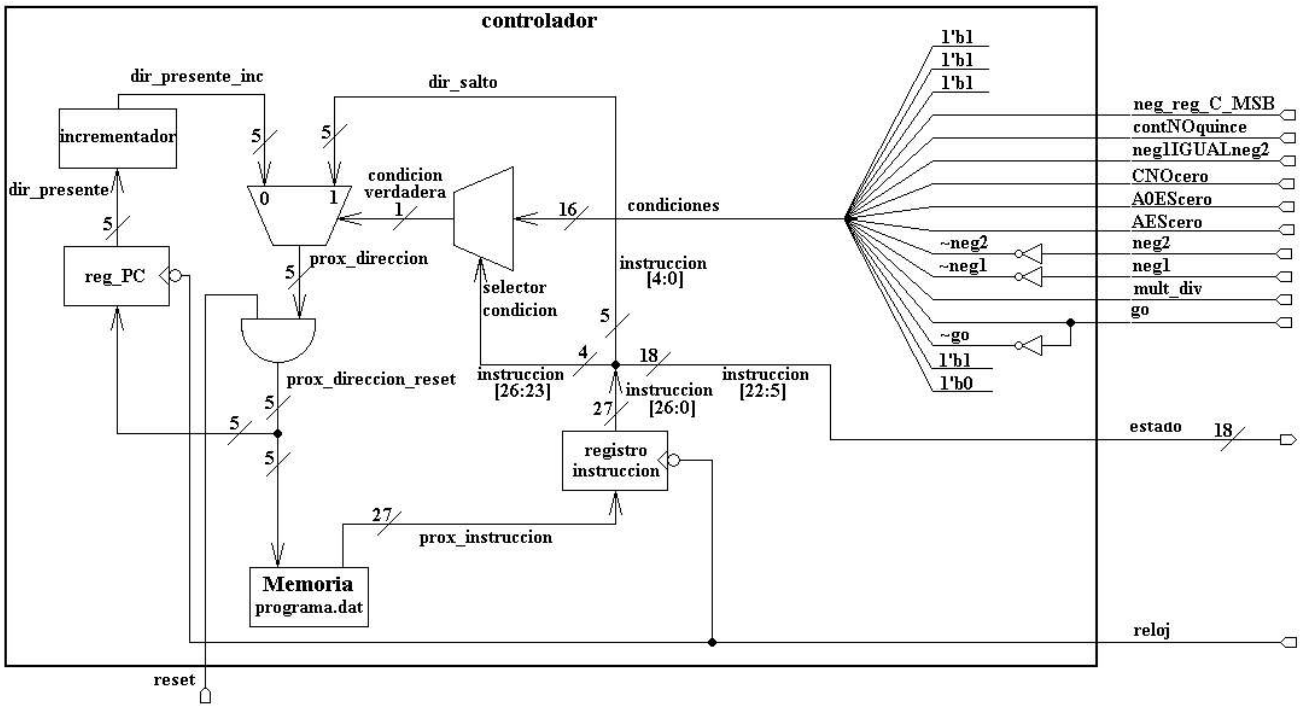
El diseño fue modularizado de la siguiente forma:

1. Controlador microprogramado:

Contiene la descripción del controlador, la memoria del microprograma, los registros de dirección

(PC) y de instrucción (IR), el incrementador de direcciones, el multiplexor o selector de condiciones y el selector de salto condicional. Esta descripción se halla en el archivo controlador.v. La memoria de programa se carga del archivo programa.dat.

La estructura del módulo controlador puede apreciarse en el siguiente diagrama:

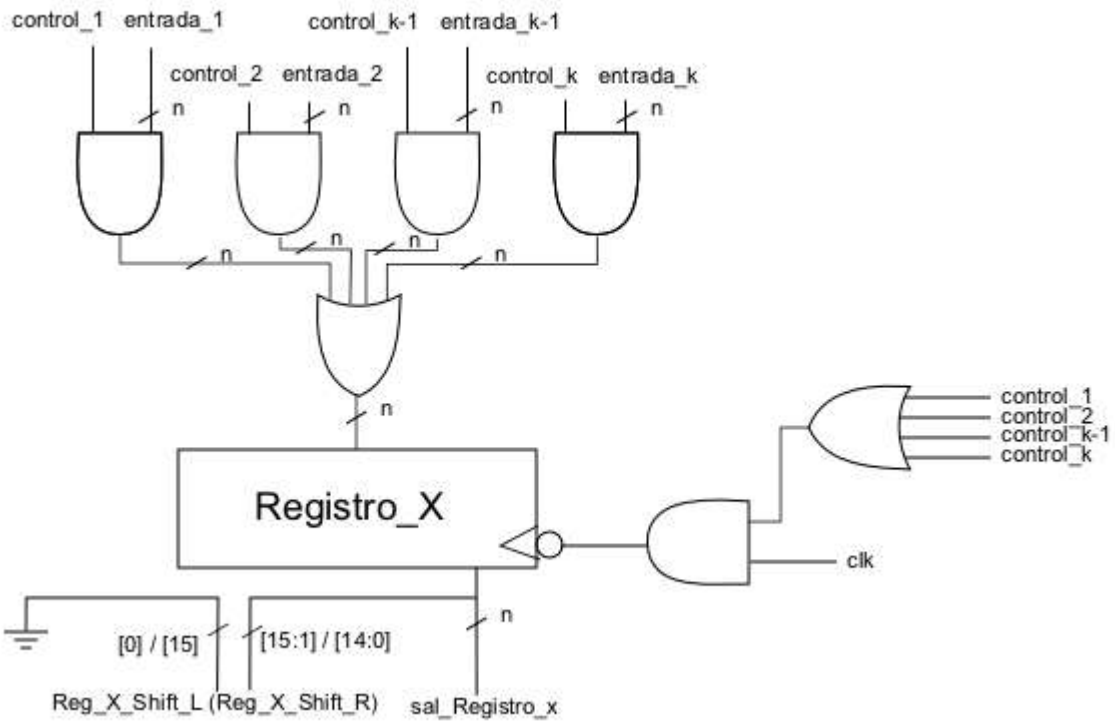


2. Módulos de descripción de los registros:

Los registros utilizados en el diseño del divisor/multiplicador microprogramado, así como su utilización según la operación requerida es la siguiente:

Nombre de registro	Tamaño (bits)	Función en Multiplicación	Función en División
A	16	Multiplicador	Divisor
B	16	Multiplicando	Cociente
C	32	Producto	Dividendo
D	4	contador	contador
neg1	1	Signo multiplicador	Signo Divisor
neg2	1	Signo multiplicando	Signo Dividendo
done	1	Señal protocolo	Señal protocolo
mult_div	1	Bandera multiplicación	Bandera división

La descripción de cada uno de estos registros junto con su lógica de entradas y control de selección se hizo de una manera prácticamente estándar, siguiendo prácticamente un “machote”, con algunas excepciones que se mencionarán más adelante. La estructura que siguen en general presenta la siguiente forma:



Como se observa, existe opcionalmente un alambrado para hacer desplazamientos hacia la izquierda, o hacia la derecha que pueden reingresar como entradas al registro.El código de Verilog correspondiente presenta también una estructura similar en casi todos los casos, de la siguiente forma:

```
//registro X con su respectivo control
module regX_con_ctrl(clk,ctrl_1,ctrl_2,ctrl_n,ent_1,ent_2,ent_n,sal_X);

        input clk, ctrl_1, ctrl_2,ctrl_n; //entradas de control
        input [n-1:0] ent_1,ent_2,ent_n; //entradas de datos
        output [n-1:0] sal_X;

        reg [n-1:0] X;

        wire selecc_cont; //control del registro
        wire [n-1:0] X_shift_r;

        //control del registro
        assign selecc_cont = (clk &(ctrl_1 | ctrl_2 | ctrl_3n));

        //alambrado para el desplazamiento
        assign X_shift_r [n-2:0] = X[n-1:1];
       assign  X_shift_r [n-1] = 1'b0;

        //alambrado de salida
        assign sal_X = X;

        //logica de seleccion
        always @ (negedge selecc_cont) //en flanco negativo asigna valor de ent
                                       //al registro, segun señal de control
                                begin
                                        if (ctrl_1)
                                                X <= ent_1;
                                        else if (ctrl_2)
                                                X <= ent_2;
                                        else if (ctrl_n)
                                                X <= ent_n;
                                end
endmodule
```

En este caso en particular se presenta el caso de un desplazamiento a la derecha, pero también se pueden realizar hacia la izquierda cambiando simplemente las siguientes líneas:

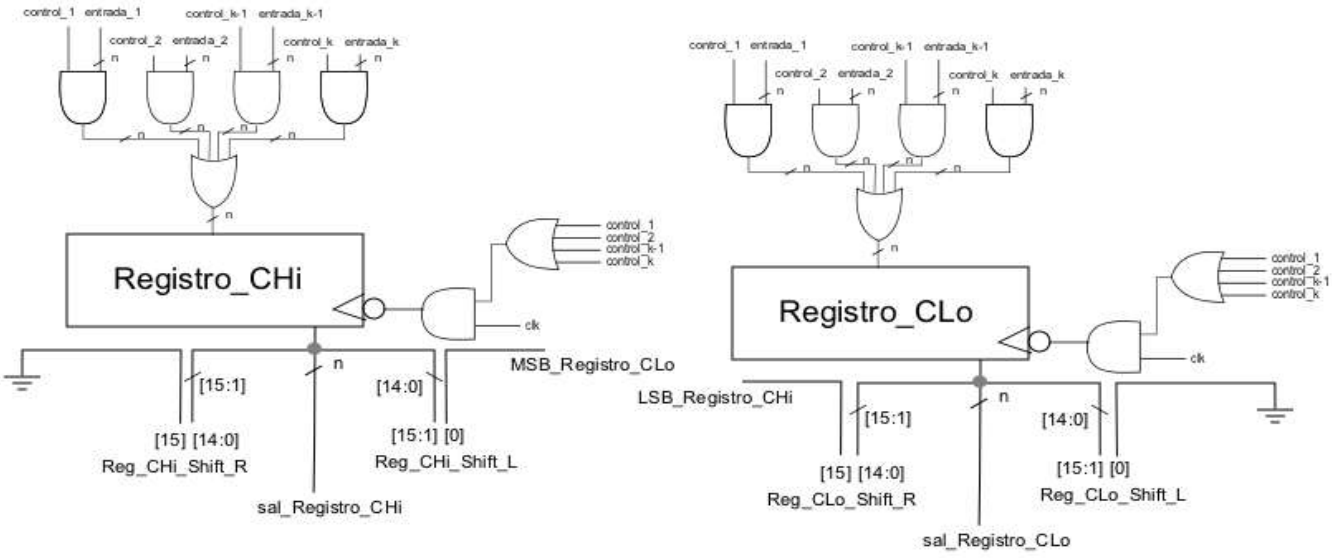
```
wire [n-1:0] X_shift_l;

//alambrado para el desplazamiento
assign X_shift_l [n-1:1] = X[n-2:0];
assign  X_shift_l [0] = 1'b0;
```

Ahora, para cada caso particular del registro, las señales de control de selección como de entrada son variadas, dependiendo de la función que realice cada registro.
A continuación se presenta una tabla que resume las señales de control específicas y las entradas seleccionadas para cada caso para cada uno de los registros (excepto las excepciones que se presentarán más adelante):

Registro	n	Señal Control	Entrada Seleccionada	Shift?
A	16	initDiv	Input_1	R
		initMult	Input_1	
		assignZ_to_A	Z[15:0] (ALU)	
		Shift_A	A_Shift_r	
B	16	initDiv	16'hOOOO	L
		initMult	Input_2[15:0]	
		assignZ_to_B	Z[15:0] (ALU)	
		Shift_B	B_Shift_l	
D	4	initDiv	4'hF	n/a
		initMult		
		assignZ_to_D	Z[3:0] (ALU)	
done	1	initDiv	1'bO	n/a
		initMult		
		done_1	1'b1	
neg1	1	initDiv	MSB_A	n/a
		initMult		
neg2	1	initDiv	MSB_C	n/a
		initMult	MSB_B	
mult_div	1	initDiv	mult_div (input)	n/a
		initMult		
C*	32	initDiv	Input_2[31:0]*	R/L
		initMult	16'hOOOOOOOO	
		assignZ_to_C	Z[31:0] (ALU)	
		CHi_assign_Z*	Z[15:0] (ALU)*	
		Shift_C_right	*	
		Shift_C_left	*	

El registro C, requiere una descripción es un tanto más compleja que los otros, siendo la excepción antes mencionada. Lo que se hizo fue dividirlo en parte alta (CHi) y parte baja (CLo) describiendo cada parte como un registro independiente, y por tanto el "registro C" era más que todo un módulo de más alto nivel, donde se instancian las partes baja y alta para formar el todo.

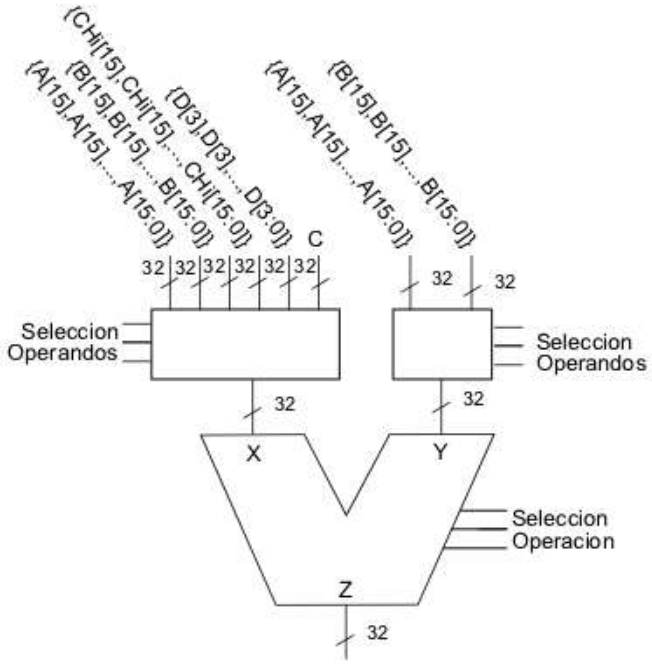


Como se observa, los registros CHi y CLo están interconectados entre sí para que puedan hacer desplazamientos tanto a la derecha como a la izquierda. Además, la parte alta de C puede recibir la parte baja de la salida de la ALU, para realizar operaciones especiales. Por tanto, se presenta también un resumen de las señales de control de estos registros y sus respectivas entradas:

Registro	n	Señal Control	Entrada Seleccionada	Shift?
CHi	16	initDiv	Input_2[31:16]	R/L
		initMult	16'hOOOO	
		assignZ_to_C	Z[31:16] (ALU)	
		CHi_assign_Z	Z[15:0] (ALU)	
		Shift_C_right	CHi_shift_r	
		Shift_C_left	CHi_shift_l	
CLo	16	initDiv	Input_2[15:0]	R/L
		initMult	16'hOOOO	
		assignZ_to_C	Z[15:0] (ALU)	
		Shift_C_right	CLo_shift_r	
		Shift_C_left	CLo_shift_l	

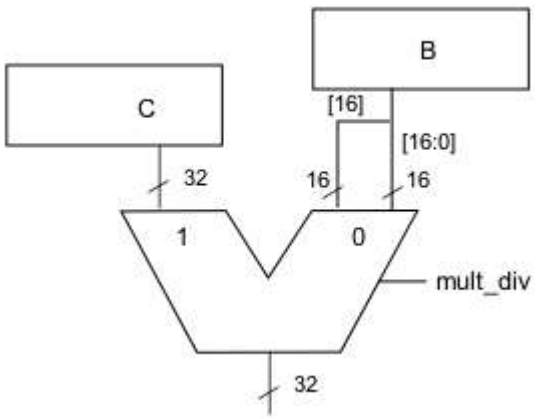
3. ALU Conductual

La ALU se trabajó completamente como un módulo con entradas y salidas de 32 bits. Por tanto resultaba necesario extender el signo en aquellos operandos que tenían una longitud menor. La estructura de la ALU es la siguiente:



4. El cascarón

El cascarón es básicamente el conjunto de módulos que conforman el divisor/multiplicador microprogramado, pues es ahí donde se instancian el conjunto de registros, el controlador y la ALU. Además, se hacen las conexiones correspondientes que reparten la microoperación proveniente del controlador en señales de control que van hacia los diferentes registros, a la ALU, etcétera. En este módulo también se incluye un selector de salida que dependiendo de la operación que se realice, tira el valor del resultado correspondiente hacia el exterior del dispositivo (C en el caso de la multiplicación y B en el caso de la división). La salida del cascarón es de 32 bits, por lo que debe extenderse el signo de B en caso de que este sea elegido. La elección de la salida entre el valor de un registro y el otro se hace por medio del valor que se haya guardado en el registro de mult_div. Este selector se puede apreciar en el siguiente ejemplo:



También en este cascarón se definen las condiciones que son generadas por los registros y que son sensadas por el controlador para saber si debe hacer saltos en el flujo del programa.

Microcódigo:

El microprograma ejecutado en el controlador se especifica en la siguiente tabla:

	m Instrucción																											
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	18	19	20	21	22	23	24	25	26	27	28	
	Seleccion de Condiciones				Asignacion de registros												Selección de operaciones			Selección de operandos			Dirección de Salto					
dir																												
X	X	X	X	X	InitDiv	InitMult	done <= 1	A<=z	A>>1	B<<1	B<=z	C<=z	C>>1	C<<1	CHi<=1	D<=z	X	X	X	X	X	X	X	X	X	X	X	
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 wait(go)
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	17 if(!mult_div)
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	0 init div
3	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	29 if(divisor)
4	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	6 if(negdivisor)
5	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	x	x	x	x	x	0 divisor = -divisor
6	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	8 if(negdividend)
7	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	1	x	x	x	x	x	0 dividend=-dividend;
8	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	x	x	x	x	x	0 quotient<<1;dividend<<1;
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	x	x	x	x	x	0 divisor
10	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	12 if(Dividend_noCero)
11	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	0	1	1	0	1	13 quotient = quotient +1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	x	x	x	x	x	0 +divisor
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	x	x	x	x	x	0 cont=cont+1
14	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	8 repeat_loop
15	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	29 if(negDivisor=negDividend)
16	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	1	1	1	0	1	29 quotient = -quotient
17	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	0 init mult
18	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	20 if(negMpy)
19	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	x	x	x	x	x	0 myMpy=-myMpy
20	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	22 if(negMend)
21	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	x	x	x	x	x	0 myMend=-myMend
22	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	24 if(myMpy[0])
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	x	x	x	x	x	0 prod=prod+{mend,16'h0000}
24	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	x	x	x	x	x	0 prod>>1 ; myMpy >> 1
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	x	x	x	x	x	0 cont=cont+1
26	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	22 repeat_loop
27	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	29 if(negMpy = negMend)
28	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	1	x	x	x	x	x	0 prod=-prod
29	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	0 done = 1
30	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	30 wait(~go)
31	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 end

La zona sombreada corresponde al algoritmo de multiplicación. Las primeras dos microinstrucciones junto con las últimas tres son comunes para ambos algoritmos. Ellas se encargan del manejar el protocolo «handshake» y de chequear la señal de control «mult_div», que define qué operación se desea realizar con los datos que se encuentran en los puertos de entrada. Observe que en las intrucciones init div e init mult se inicializan todos los registros, incluyendo done. En el flanco negativo correspondiente a esas microinstrucciones se capturan los datos que están en los puertos de entrada, al mismo tiempo que se pone en bajo la salida de control done. Cuando done es puesta en alto en la antepenúltima instrucción, el resultado de la operación está disponible en el puerto de salida.

Decodificación de las selección de condiciones, operaciones y operandos:

Seleccion de Condiciones	
0 0 0 0	Falso
0 0 0 1	Verdadero
0 0 1 0	~go
0 0 1 1	go
0 1 0 0	mult/div
0 1 0 1	~neg1
0 1 1 0	~neg2
0 1 1 1	AesCero
1 0 0 0	A[0]esCero
1 0 0 1	CesNeg
1 0 1 0	neg1=neg2
1 0 1 1	cont_no_$F

Selección de operaciones	
0 0 0	NOP
0 0 1	x+y
0 1 0	x-y
0 1 1	x+1
1 0 0	-x

Selección de operandos	
0 0 0	NOP
0 0 1	x=A
0 1 0	x=B
0 1 1	x=C
1 0 0	x=Chi, y = A
1 0 1	x=Chi, y = B
1 1 0	x=D

Simulación:

El sistema es compilado utilizando el archivo de proyecto mult_div_prog.prj. El comando para la compilación es:

```
$>iverilog -c mult_div_prog.prj -o mult_div_prog.vvp
```

Se genera el ejecutable mult_div_prog.vvp, que corre mediante el programa vvp:

```
$>vvp mult_div_prog.vvp
```

El resultado del paso anterior debería ser igual al siguiente:

```
VCD info: dumpfile mult_div_prog.vcd opened for output.
Prueba No. 1. Operacion: multiplicacion
multiplicando 1 = 0011010100100100
multiplicando 2 = 0101111010000001
producto esperado = 00010011100111011111111100100100
producto obtenido = 00010011100111011111111100100100

Prueba No. 2. Operacion: division
dividendo = 10000100100001001101011000001001
  divisor = 0101011001100011
cociente esperado = 11111111111111111110010000110100
cociente obtenido = 11111111111111111110010000110100

Prueba No. 3. Operacion: multiplicacion
multiplicando 1 = 0111101100001101
multiplicando 2 = 1001100110001101
producto esperado = 11001110110000011000101100101001
producto obtenido = 11001110110000011000101100101001

.
.
.

Prueba No. 99. Operacion: multiplicacion
multiplicando 1 = 1110010011111010
multiplicando 2 = 1110111001100001
producto esperado = 00000001110111000010111010111010
producto obtenido = 00000001110111000010111010111010

Prueba No. 100. Operacion: division
dividendo = 00001011100101000000100100010111
  divisor = 0111100010100001
cociente esperado = 00000000000000000001100010010010
```

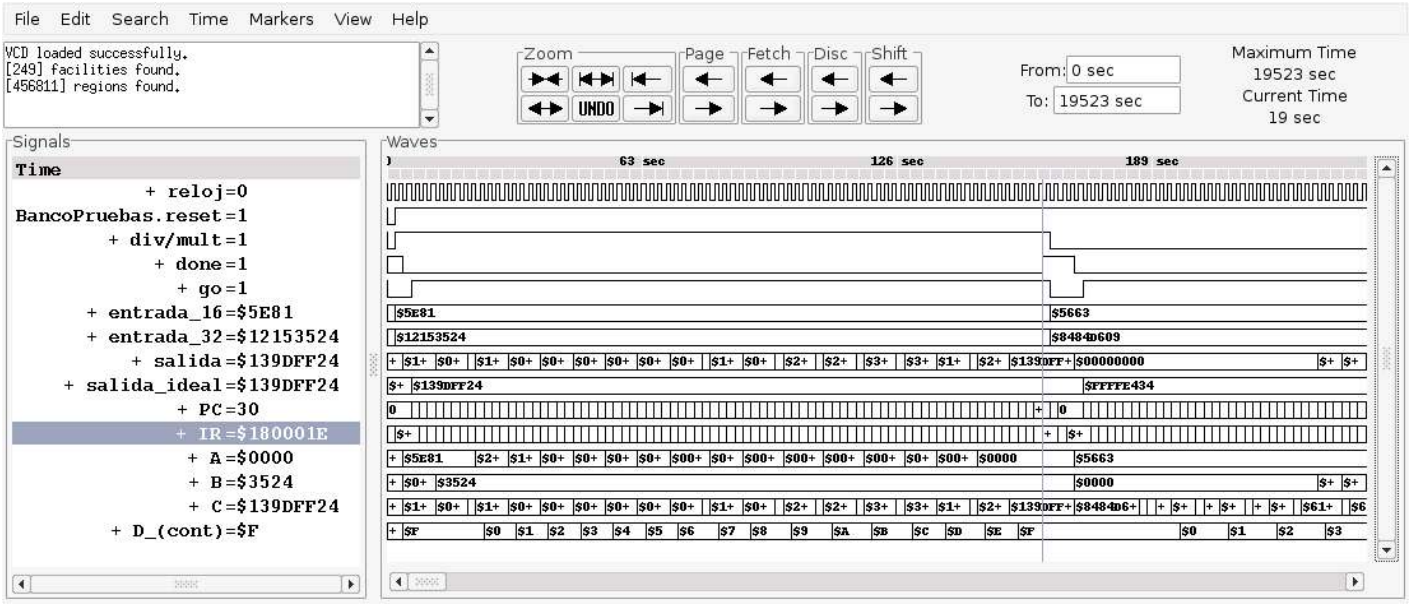
```
cociente obtenido = 00000000000000000001100010010010
```

Los operandos y resultados se despliegan en binario porque cuando se representan en octal, decimal o hexadecimal, el signo es ignorado y se toman todos como positivos.

El tercer paso es visualizar las señales más importantes en GTKwave. El comando es el siguiente:

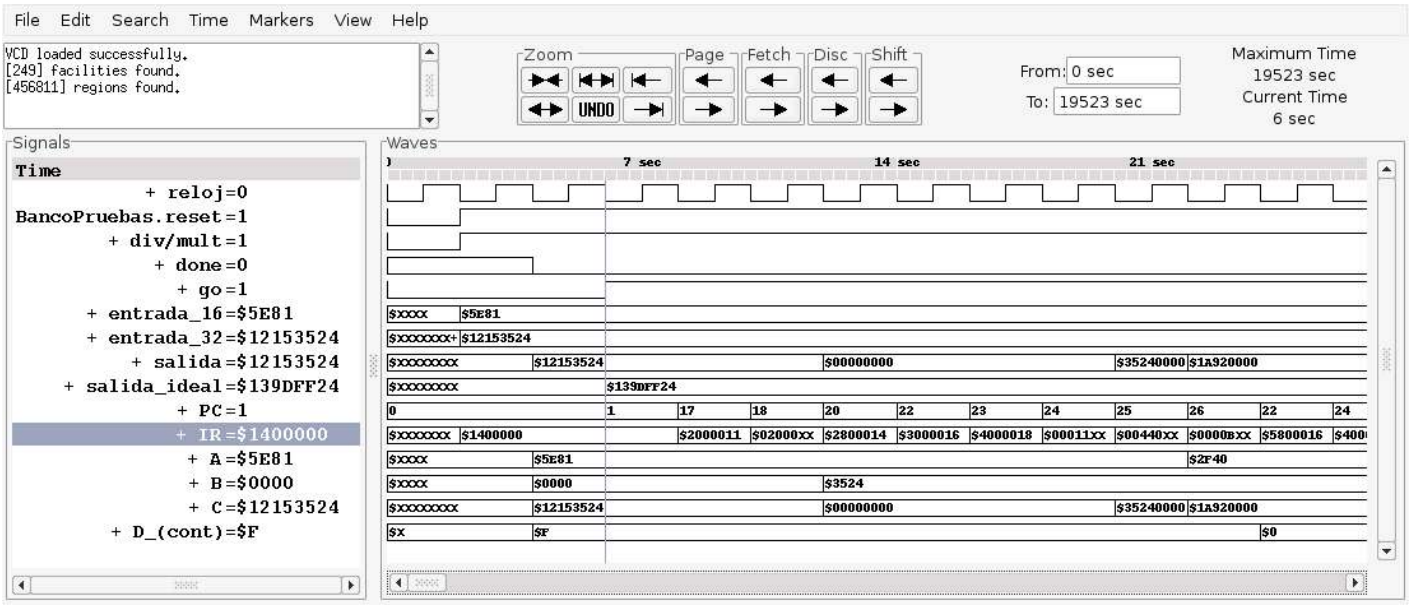
```
$>gtkwave mult_div_prog.vcd signals.gtkwave
```

El segundo argumento llama automáticamente un conjunto de señales escogidas arbitrariamente y que, a nuestro juicio, describen el funcionamiento del sistema. A continuación se presentan algunas capturas de secciones importantes de la simulación:



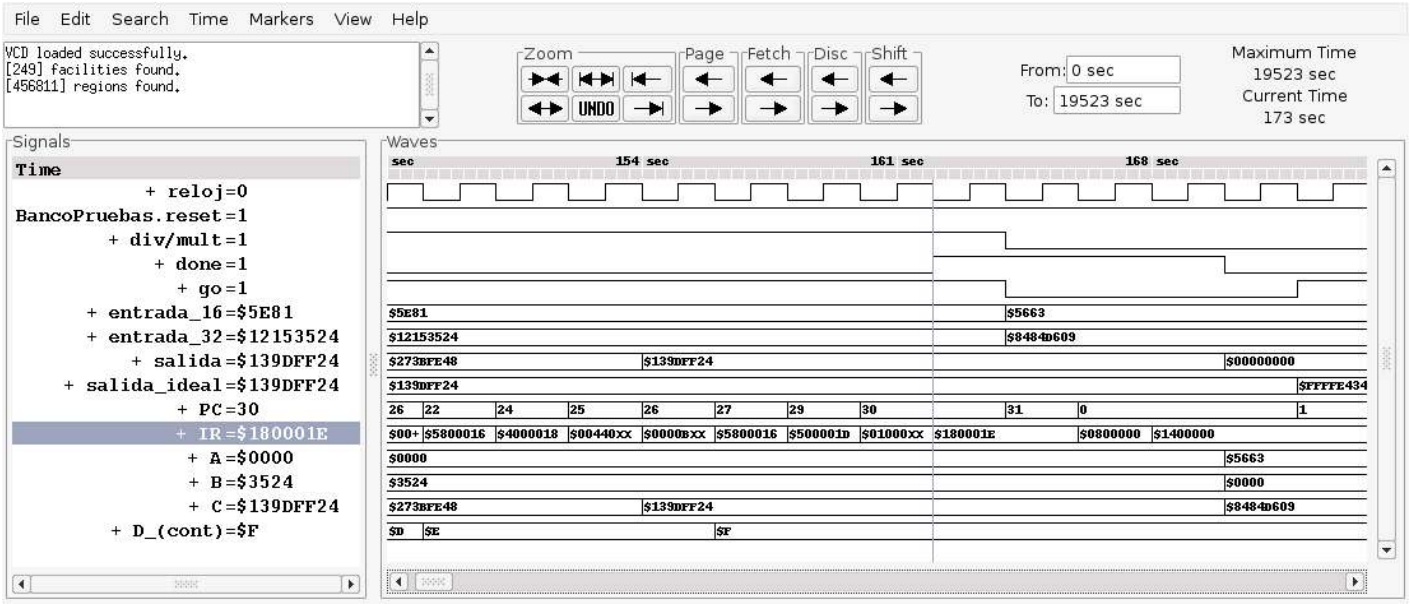
Captura No. 1: Operación completa.

En la captura anterior se observa la simulación desde el inicio hasta la concreción de la primera operación. Los valores de las variables en el cuadro «signals» ese el correspondiente al momento en que la señal «done» hace la transición hacia alto. También puede apreciarse el comportamiento del protocolo «handshake».



Captura No. 2: Secuencia de inicialización.

Arriba se tiene el secuencia de inicialización. El reset dura T+1 unidades de tiempo. Antes de reestablecerse, el controlador está en un estado desconocido. Una vez que el reset es puesto en alto, empieza la ejecución del microprograma. Lo primero es inicializar los registros según el tipo de operación que el usuario desee. Luego, se espera que go se levante para continuar con la ejecución. En este caso, se está realizando una multiplicación, por lo tanto el puntero de programa (PC) pasa de 1 a 17.



Captura No. 2: Final de operación

Aquí se observan los ciclos de reloj alrededor de los cuales se finaliza la primera operación. El cursor está colocado en el momento que «done» se levanta. En ese momento, el registro «salida» posee en resultado de la multiplicación, idéntico al obtenido en el registro «salida_ideal» contra el cual el probador determina si la operación fue realizada con éxito o no. La señal «go» es bajada por el probador, que se encarga de generar nuevos operandos e invertir la señal «div/mult» para alternar la operación. El sistema multiplicador/divisor, por su parte, baja la señal «done». El probador sensa esto, y en siguiente flanco negativo, «go» es levantada para que se inicie una operación.