Laboratorio 2
Arquitectura de Computadores

Macarena Oyague

May 8, 2020

Resolución de ejercicios

En este informe se realiza la descripción de los códigos implementados para la solución de los ejercicios del Laboratorio 2. Se anexan los módulos, testbenches y archivos vcd correspondientes a estas explicaciones en la misma carpeta en la que está este archivo.

1 MUX 2:1 16 bits

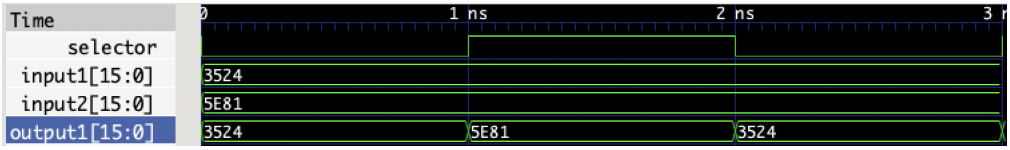
Para realizar el MUX 2:1 de 16 bits se debe tener

- Entrada: 2 inputs de 16 bits, 1 selector de 1 bit
- Salida: 1 output de 16 bits

Lo que hace el selector es determinar el valor de salida de un MUX según sea asignado un valor de verdad en la tabla. En el programa los inputs se generan de manera aleatoria, por lo que por temas de practicidad se tomarán variables:

Time	Selector	Input1	Input2	Output1
0	0	A	B	A
1	1	A	B	B

Para hacer esto posible, se debe incluir una variación en el selector entre sus valores binarios cada 1 unidad de tiempo. En esta ocación se trabajará con nanosegundos y se obtiene lo siguiente:



2 MUX

2.1 MUX 2:1 16 bits

Este ejercicio fue explicado en la sección 1 del informe.

2.2 MUX 8:1 16 bits

La solución de este problema surge de utilizar 2 MUX 4:1, por lo cual se explicará este inicialmente. El MUX 4:1 está compuesto por:

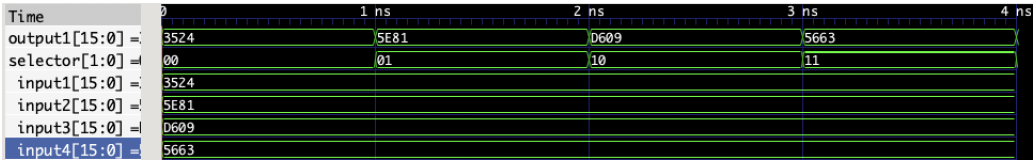
- Entrada: 4 inputs de 16 bits, 1 selector de 2 bits

- Salida: 1 output de 16 bits

Al igual que para esta subsección se necesitarán 2 MUX 4:1, para realizar el MUX 4:1 se necesitan 2 MUX 2:1. En este caso, se tienen dos bits en el selector: uno para determinar el cambio para seleccionar cuál será la salida en cada MUX, y otro para seleccionar cuál MUX se tomará en cuenta para ello. La siguiente tabla ilustra esto, nuevamente con una asignación de variables:

Time	Selector	Input1	Input2	Input3	Input4	Output1
0	00	A	B	C	D	A
1	01	A	B	C	D	B
2	10	A	B	C	D	C
3	11	A	B	C	D	D

El primer MUX debe estar conformado por los dos primeros inputs y el segundo por los últimos dos. Así, el bit0 del selector varía cada unidad de tiempo para determinar el cambio necesario para que estos submuxes arrojen el output a asignarse. Además, el bit1 del selector sirve para saber qué MUX se tomará en cuenta, por lo cual debe abarcar todos los valores y varía cada dos unidades de tiempo. En este caso si es que es 0, arrojará el output correspondiente al que arrojó el primer MUX 2:1, y si es 1 arrojará el del segundo MUX 2:1, que también están variando el output según lo explicado. La imagen puede ilustrar ello:



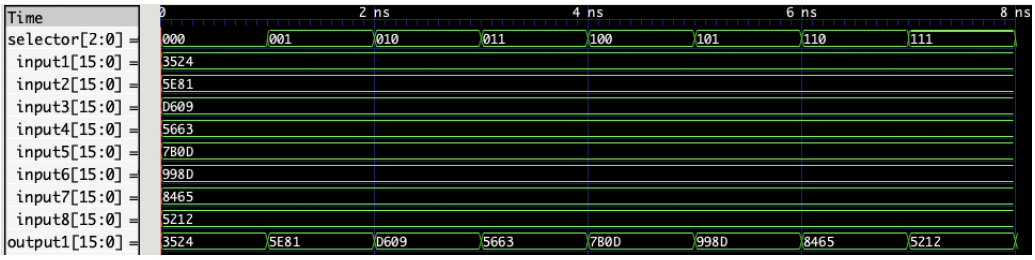
Luego de entender el MUX 4:1, podemos seguir con la explicación de los elementos del código del MUX 8:1. Este está compuesto de la siguiente manera:

- Entrada: 8 inputs de 16 bits, 1 selector de 3 bits
- Salida: 1 output de 16 bits

Para esta subsección se utilizarán 2 MUX 4:1, al primero de ellos se le asignarán los cuatro primeros inputs y al segundo los cuatro últimos. Además, se tienen tres bits en el selector: los dos primeros bits se mandan a cada MUX 4:1 para que arrojen el output correspondiente según la tabla que evaluamos anteriormente, y el tercero sirve para seleccionar cuál de los dos mux se tomará en cuenta para arrojar la salida de este MUX 8:1. Con asignación de variables, la tabla muestra esto:

Time	Selector	I1	I2	I3	I4	I5	I6	I7	I8	Output1
0	000	A	B	C	D	E	F	G	H	A
1	001	A	B	C	D	E	F	G	H	B
2	010	A	B	C	D	E	F	G	H	C
3	011	A	B	C	D	E	F	G	H	D
4	100	A	B	C	D	E	F	G	H	E
5	101	A	B	C	D	E	F	G	H	F
6	110	A	B	C	D	E	F	G	H	G
7	111	A	B	C	D	E	F	G	H	H

El bit0 del selector varía cada unidad de tiempo para determinar el cambio necesario de cada MUX 2:1 que contiene cada MUX 4:1, el bit1 varía cada dos unidades para que el MUX 4:1 pueda seleccionar cuál MUX 2:1 tomará en cuenta para el output, y el bit2 varía cada 4 unidades de tiempo, para determinar cuál MUX 4:1 tomará en cuenta para finalmente arrojar su salida como salida del MUX 8:1. Se puede ver de la siguiente manera:



2.3 MUX 16:1 16 bits

Siguiendo la misma línea, podemos afirmar que es posible construir un MUX 16:1 con dos MUX 8:1. La composición del MUX correspondiente a esta sección es la siguiente:

- Entrada: 16 inputs de 16 bits, 1 selector de 4 bits
- Salida: 1 output de 16 bits

Al primer MUX 8:1 se le asignarán los inputs 1-8 y al segundo los inputs 9-16. Los tres primeros inputs del selector se dirigen a cada MUX 8:1 para que cada uno de ellos arroje un output según cada asignación dada para estos bits del selector, la cual fue explicada en la sección anterior. El último bit del selector, en cambio sirve para determinar cuál de los submuxes será tomado en cuenta. Se puede apreciar ello con la siguiente tabla:

Time	Sel	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I5	I16	Out
0	0000	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	A
1	0001	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	B
2	0010	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	C
3	0011	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	D
4	0100	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	E
5	0101	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	F
6	0110	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	G
7	0111	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	H
8	1000	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	A
9	1001	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	B
10	1010	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	C
11	1011	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	D
12	1100	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	E
13	1101	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	F
14	1110	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	G
15	1111	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	H

El bit0, bit1 y bit2 del selector tendrán una variación de la misma manera que la subsección anterior, esto es: bit0 cada unidad de tiempo para determinar el cambio necesario de cada MUX 2:1 que contiene cada MUX 4:1; bit1 cada dos unidades de tiempo para que el MUX 4:1 pueda seleccionar cuál MUX 2:1 tomará en cuenta para el output; y el bit2 cada 4 unidades de tiempo, para determinar cuál MUX 4:1 tomará el MUX 8:1. El bit3 variará cada 8 unidades de tiempo y cumplirá la función de seleccionar cuál submux 8:1 debe ser tomado en cuenta para arrojar la salida final:

3 DEMUX

3.1 DEMUX 2:1 16 bits

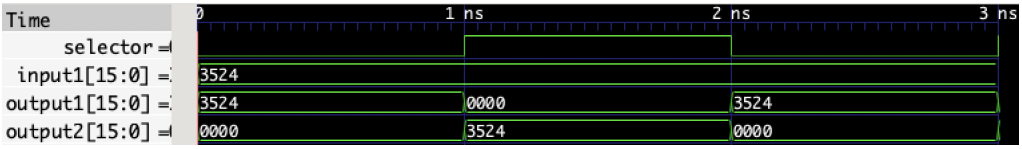
Para realizar el DEMUX 2:1 se debe tener:

- Entrada: 1 input de 16 bits, 1 selector de 1 bit
- Salida: 2 outputs de 16 bits

La función del selector es determinar los valores de salida de un DEMUX concorde a la asignación de un valor para ello en la tabla. En el programa los inputs se generan de manera aleatoria, por lo que se tomarán variables que reemplacen esta data:

Time	Selector	Input1	Output1	Output2
0	0	A	A	0
1	1	A	0	A

Los valores de salida pueden ser 0 en sistema 16-bits y A que es la variable que representa al input. Para hacer esto posible, se debe incluir una variación en el selector entre sus valores binarios cada 1 unidad de tiempo. En esta ocasión se trabajará con nanosegundos y se obtiene lo siguiente:



3.2 DEMUX 8:1 16 bits

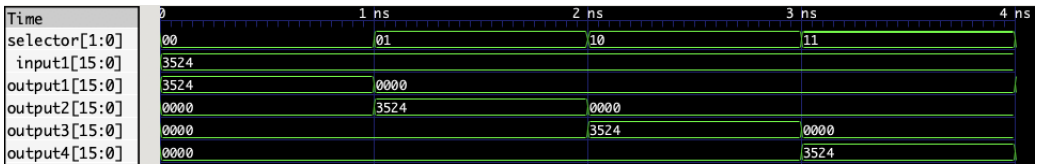
Al igual que en el caso del MUX 8:1, es posible realizar un DEMUX 8:1 al utilizar dos DEMUX 4:1. Por ello se explicará inicialmente esta solución. Debemos tomar en cuenta que el DEMUX 4:1 está conformado por:

- Entrada: 1 input de 16 bits, 1 selector de 2 bits
- Salida: 4 outputs de 16 bits

Para poder implementar el DEMUX 4:1, se necesitarán dos DEMUX 2:1: el primero tendrá como outputs los outputs para evaluar en la primera mitad de outputs del DEMUX 4:1 y el segundo tendrá los de la segunda. Como entrada, cada DEMUX 2:1 tendrá el input (A) y el primer bit del selector. Se debe tomar en cuenta que en el DEMUX, uno de los valores de salida será igual al input y el resto debe ser 0 en el sistema en el que se escoja:

Time	Selector	Input1	Output1	Output2	Output3	Output4
0	00	A	A	0	0	0
1	01	A	0	A	0	0
2	10	A	0	0	A	0
3	11	A	0	0	0	A

Dependiendo de la asignación, el segundo bit del selector se encargará de evaluar cuáles outputs de los subdemuxes debe ser igualado a 0 y cuál será el que retorne A. Si es que el bit1 del selector es 0, se deberá igualar a 0 la porción de outputs correspondientes al subdemux 2:1 asignado a este valor, y si es 1, se deberá arrojar los outputs correspondientes al otro subdemux, el cual estará evaluando cual de ellos será A y cuál 0. Al igual que en el caso de MUX 4:1, los tiempos de variación son de una unidad de tiempo para bit0 (que evalúa los subdemuxes) y dos unidades de tiempo para el bit1 (que evalúa cuál subdemux podrá arrojar el input como salida). En la experimentación, es posible verlo de la manera siguiente:



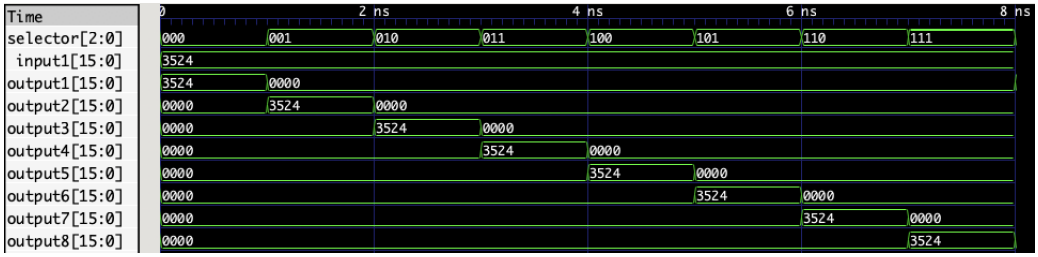
Luego de poder comprender el funcionamiento del DEMUX 4:1, es posible pasar a la explicación del DEMUX 8:1. Este está compuesto de la siguiente manera:

- Entrada: 1 input de 16 bits, 1 selector de 3 bits
- Salida: 8 outputs de 16 bits

Para esta subsección se utilizarán 2 DEMUX 4:1, al primero de ellos se le asignarán los cuatro primeros outputs y al segundo los cuatro últimos. Además, se tienen tres bits en el selector: los dos primeros bits se mandan a cada DEMUX 4:1 para que arrojen el output correspondiente según la tabla que evaluamos anteriormente, y el tercero sirve para seleccionar cuál de los dos demux se tomará en cuenta para arrojar las salidas de este DEMUX 8:1. Con el input A para cada subdemux y con asignación de variables, la tabla muestra:

Time	Selector	In	O1	O2	O3	O4	O5	O6	O7	O8
0	000	A	A	0	0	0	0	0	0	0
1	001	A	0	A	0	0	0	0	0	0
2	010	A	0	0	A	0	0	0	0	0
3	011	A	0	0	0	A	0	0	0	0
4	100	A	0	0	0	0	A	0	0	0
5	101	A	0	0	0	0	0	A	0	0
6	110	A	0	0	0	0	0	0	A	0
7	111	A	0	0	0	0	0	0	0	A

Con el mismo sentido de lógica explicado para el DEMUX 4:1, el DEMUX 8:1 funciona de manera similar. Los subdemuxes asociados estarán relacionados con el tercer bit del selector: arrojarán 0 en todos sus outputs si es que el valor del selector no concuerda con la asignación dada; y, si es que el valor del selector sí concuerda con la asignación dada al subdemux, se retornará A en sólo uno de los múltiples outputs y 0 en el resto siguiendo la tabla de los valores de DEMUX 4:1. El bit0 del selector en esta ocasión también variará cada unidad de tiempo, el bit1 cada dos unidades de tiempo y el bit2 cada cuatro unidades de tiempo, que es cuando se estará asignando la decisión de salida de A al otro demux asociado. Se puede observar lo siguiente:



3.3 DEMUX 16:1 16 bits

El DEMUX 16:1 está conformado por dos DEMUX 8:1. La composición del DEMUX correspondiente a esta sección es la siguiente:

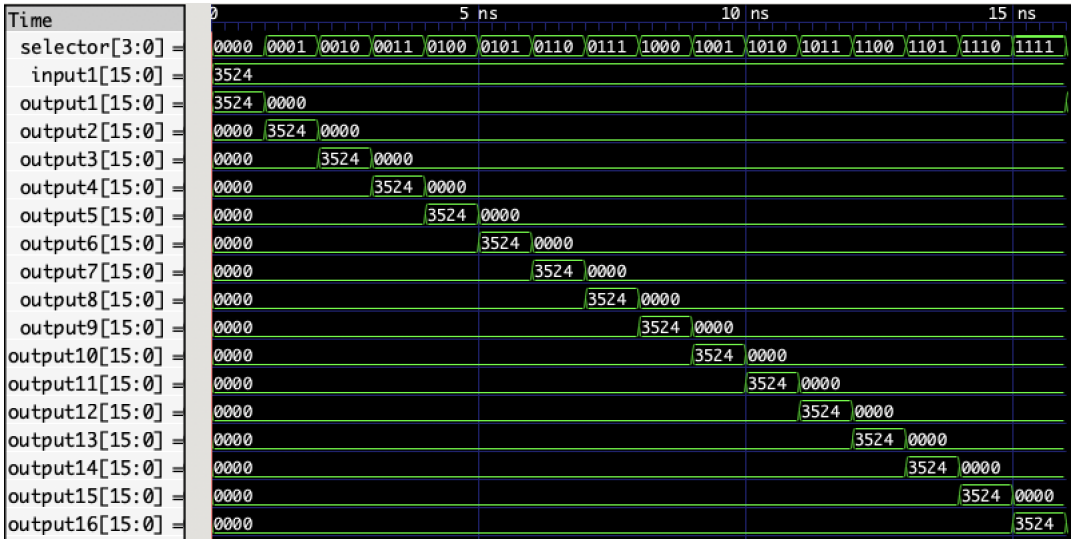
- Entrada: 1 input de 16 bits, 1 selector de 4 bits
- Salida: 16 outputs de 16 bits

Al primer DEMUX 8:1 se le asignarán los outputs 1-8 y al segundo los outputs 9-16. El input A y los tres primeros bits del selector se asignan a cada DEMUX 8:1, para que cada uno de ellos arroje outputs que luego serán evaluados. Al igual que en las secciones anteriores, cada valor binario del último bit del selector, que en este caso es el cuarto, será asignado a cada subdemux. Concorde a ello, cada subdemux arrojará 7 outputs con valor 0 y uno con valor 1, los cuales serán evaluados con el bit3 del selector en el DEMUX 16:1 para ver cuál demux 8:1 se tomará en cuenta: el que sea compatible con el valor del demux retornará en uno de sus outputs el input A, y el que no tendrá únicamente 0s. En la siguiente tabla, se ve evidenciado ello:

*Los números del 1-16 hacen referencia a los outputs

Time	Sel	In	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0000	A	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0001	A	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0010	A	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0011	A	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0
4	0100	A	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0
5	0101	A	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0
6	0110	A	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0
7	0111	A	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0
8	1000	A	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0
9	1001	A	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0
10	1010	A	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0
11	1011	A	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0
12	1100	A	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0
13	1101	A	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0
14	1110	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0
15	1111	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A

El bit0, bit1 y bit2 del selector tendrán una variación de la misma manera que la subsección anterior, esto es: bit0 cada unidad de tiempo para determinar el cambio necesario de cada DEMUX 2:1 que contiene cada DEMUX 4:1; bit1 cada dos unidades de tiempo para que el DEMUX 4:1 pueda seleccionar cuál DEMUX 2:1 tomará en cuenta para el output; y el bit2 cada 4 unidades de tiempo, para determinar cuál DEMUX 4:1 tomará el DEMUX 8:1. El bit3 cumplirá la función de seleccionar cuál subdemux 8:1 debe ser tomado en cuenta para arrojar la salida final y variará cada 8 unidades de tiempo:



4 DECODER 3-8

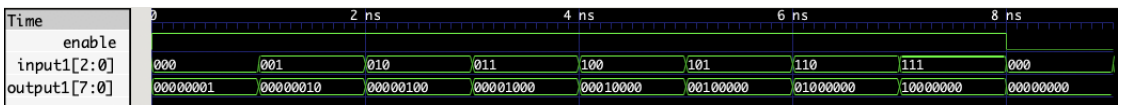
El decoder 3-8 está compuesto por lo siguiente:

- Entrada: 1 input de 3 bits, 1 enable de 1 bit
- Salida: 1 output de 8 bits

Lo que realiza la sección de código puede dividirse en dos tareas principales. La primera es evaluar cuál es el valor del enable: si es 0, no importa cuál es el input, el output será 0 en todas las posiciones; si el input es 1 se realizará la segunda tarea. Esta, en cambio, lo que hace es evaluar cada fila de la siguiente tabla:

Time	E	I[2]	I[1]	I[0]	O[7]	O[6]	O[5]	O[4]	O[3]	O[2]	O[1]	O[0]
0	1	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	1	0	0	0	0	0	0	1	0
2	1	0	1	0	0	0	0	0	0	1	0	0
3	1	0	1	1	0	0	0	0	1	0	0	0
4	1	1	0	0	0	0	0	1	0	0	0	0
5	1	1	0	1	0	0	1	0	0	0	0	0
6	1	1	1	0	0	1	0	0	0	0	0	0
7	1	1	1	1	1	0	0	0	0	0	0	0
8	0	x	x	x	0	0	0	0	0	0	0	0

Es posible observar a partir de la tabla anterior que, al convertir el input en un número decimal, la posición del bit correspondiente a él en el output será 1. Por ello, para cada posición se pregunta si es que el valor del input corresponde en decimal a dicha posición: si es así, el output en ella es 1; de lo contrario es 0. Para poder demostrar una salida en cada posición del bit, se tienen que asignar todos los valores posibles para un número binario de 3 bits. Por ello, se realizó en el input variaciones en función de unidades de tiempo, siendo cada unidad de tiempo para el bit0, cada dos unidades de tiempo para el bit1 y cada cuatro para el bit2. En cuanto al enable, se cambió su valor cada 8 unidades de tiempo. Se demostrará la ejecución de estas tareas mencionadas anteriormente por medio del siguiente esquema:



5 ENCODER 8-3

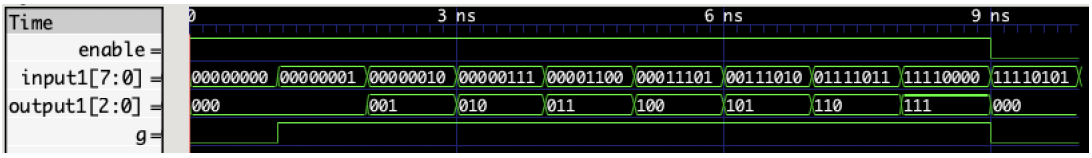
El encoder 8-3 está compuesto por lo siguiente:

- Entrada: 1 input de 8 bits, 1 enable de 1 bit
- Salida: 1 output de 3 bits

En este caso, la implementación de la solución se realiza nuevamente analizando los valores del input para encontrar un patrón y poder arrojar el output. El enable realiza una operación opuesta a la del decoder visto en la sección anterior. De igual manera que en este, lo primero a realizar es evaluar si es que el enable está encendido o apagado. Si es 0, el output del encoder 8-3 será 0 sin importar el input insertado; en el caso en el que el encoder sea 1, se tendrá que tomar en cuenta en qué posición está el primer 1 a partir del bit más significativo hacia el menos significativo. Como es una cuestión de "prioridad" hacia este primer bit 1 más significativo, es posible llenar con dont cares el valor de los bits menos significativos siguientes, ya que no influirán en decidir el output. El output se relaciona al número en binario de la posición en la que se encuentre el primer valor 1 desde el bit más significativo. Por otro lado, el output g debe variar dependiendo de las dos entradas que se tienen: si es que el enable es 0, no importa el valor del input, la salida de g será 0; si es que el enable es 1, g tomará el valor 0 únicamente si es que el valor del input es 0 también y, de lo contrario, será 1. Es posible ver las salidas en función a las diferentes opciones de input en la siguiente tabla:

Time	E	I[7]	OI[6]	I[5]	I[4]	I[3]	I[2]	I[1]	I[0]	O[2]	O[1]	O[0]	G
0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	1	0	0	0	1
2	1	0	0	0	0	0	0	1	x	0	0	1	1
3	1	0	0	0	0	0	1	x	x	0	1	0	1
4	1	0	0	0	0	1	x	x	x	0	1	1	1
5	1	0	0	0	1	x	x	x	x	1	0	0	1
6	1	0	0	1	x	x	x	x	x	1	0	1	1
7	1	0	1	x	x	x	x	x	x	1	1	0	1
8	1	1	x	x	x	x	x	x	x	1	1	1	1
9	0	x	x	x	x	x	x	x	x	0	0	0	0

Para poder obtener todas las salidas de la tabla previa, es necesario obtener combinaciones en los bits del input de tal manera en la que todos los casos sean contemplados. Por ello, las asignaciones en las distintas posiciones de los inputs varían: I[0] cada unidad de tiempo; I[1] cada dos unidades de tiempo; I[2] cada tres unidades de tiempo; I[3] cada cuatro unidades de tiempo; I[4] cada cinco unidades de tiempo; I[5] cada seis unidades de tiempo; I[6] cada siete unidades de tiempo; I[7] cada ocho unidades de tiempo; y el enable cada ocho unidades de tiempo. Se obtendría la siguiente simulación:



6 BARREL SHIFTER 32-BIT