Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

`IE0411` Microelectrónica -G01-

Tarea 3: Potencia

Jorge Muñoz Taylor (A53863) - `jorge.munoztaylor@ucr.ac.cr`

II-2020

Índice

1. Desarrollo

1.1. Diseño del contador

El sumador de rizado de 8 bits se diseño juntando 8 sumadores completos de 1 bit, estos sumadores completos se crearon de forma estructural utilizando la biblioteca de componentes *libreria.v* y el diagrama que se muestra en la figura 1.

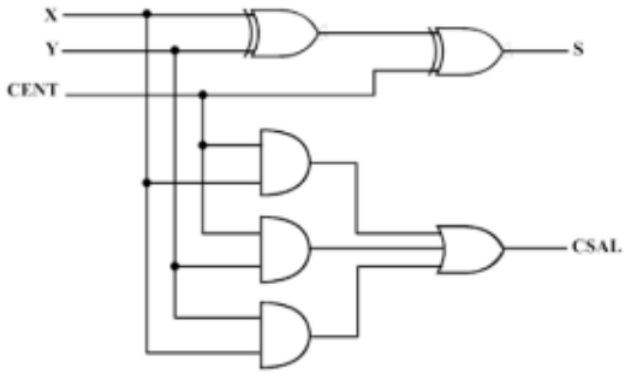

Figura 1: Diseño estructural que se implementó para crear el sumador completo de 1 bit, la entrada *CENT* corresponde a un bit de *carry* procedente de otro sumador completo, *CSAL* corresponde al bit de *carry* que produce la suma.

Para implementar los 8 sumadores completos en el sumador de rizado se utilizó el comando *generate* de verilog[1], de esta forma se simplificó el proceso de escritura del código. Como se puede ver en el código 1 esta función se encarga de replicar el circuito la cantidad de veces que sea necesario, solo hay que tener el cuidado de conectar todas las salidas y entradas entre cada dispositivo instanciado, por ejemplo, la línea *assign in_ci[i+1] = out_co[i]* comple la función de conectar los *carrys* entre los sumadores completos.

Código 1: Uso de generate para simplificar el diseño

```
genvar i;
generate

  for (i=0; i<8; i=i+1)
  begin

    if ( i != 7 )
      assign in_ci[i+1] = out_co[i];

    sumador_completo #(PwrC) BLOCK
    (
      .x    ( a[i] ),
      .y    ( b[i] ),
      .cent ( in_ci[i] ),
      .s    ( s[i] ),
      .csal ( out_co[i] )
    );
  end

endgenerate
```

1.2. Diseño de la prueba para los sumadores

El diseño del *testbench*

- Se instanciaron los 3 sumadores (el de rizado, el lógico y el de *lookahead*), durante toda la prueba se les colocarán los mismos valores de entrada.
- Para cada una de las 3 partes del proyecto se correrá la misma prueba, solo variarán los retardos y las compuertas para el sumador de rizado.
- Primero se ejecutarán 50 sumas seguidas con una semilla pseudoaleatoria, esto se utiliza para verificar que los 3 sumadores arrojen los mismos resultados.
- Se ejecutarán 500, 1000, 2000 y 5000 con 3 semillas diferentes para cada sumador, es decir, para el sumador de rizado se ejecutarán primero 500 sumas con una semilla, luego 500 sumas con otra semilla y otras 500 sumas con otra semilla, luego se hará el mismo procedimiento con 1000 sumas, después con 2000 y 5000, lo mismo con los demás sumadores. Esto con el fin de encontrar la potencia consumida por cada sumador.
- Por último se miden los retardos de cada sumador, el procedimiento que se siguio fué el siguiente:

  a. Se resetea la salida del sumador con 1's, esto se consigue sumando FF a 0.
  b. Se mide el tiempo en el que se encuentra la simulacion.
  c. Se ponen las entradas corespondientes.
  d. Se espera a que la salida llegue al valor que debe dar.

e. Se mide el tiempo en el que se llega al valor correcto y se le resta el tiempo inicial.

f. Se repite el proceso con otro sumador, así hasta que se analicen los 3 casos solicitados:

f.1. oprA == 00 oprB == 00

f.2. oprA == 00 oprB == 01

f.3. oprA == FF oprB == 01

1.3. Composición de la prueba

Se compone de 6 archivos:

- **contador_Transicion.v:** Lleva el conteo de las transiciones bajo a alto de las compuertas que componen a los sumadores, esto permite calcular la potencia consumida por el dispositivo.
- **definiciones.v:** Contiene los valores de potencia y retardo de cada compuerta en la biblioteca.
- **definiciones2.v:** Contiene los valores de potencia y retardo (modificados para la parte 2) de cada compuerta en la biblioteca.
- **libreria.v:** Contiene la definición de las compuertas lógicas que se utilizarán para definir los sumadores.
- **sumador_logico.v:** Definición estructural del sumador lógico.
- **sumador_look.v:** Definición estructural del sumador de lookahead.
- **sumador_rizado.v:** Definición estructural del sumador completo y del sumador de rizado.
- **sumador_rizado2.v:** Definición estructural del sumador completo (modificado para la parte 3) y del sumador de rizado.
- **BancoPruebas.v:** Contiene las pruebas realizadas a los sumadores.

2. Resultados

2.1. Parte 1

En la figura 2 se muestra una captura de la terminal donde puede apreciarse como los tres sumadores tienen el mismo resultado para las mismas entradas (la salida del sumador de rizado está etiquetada como *Sumador_1*), esto verifica que el sumador se comporta como debe y por lo tanto el diseño estructural es correcto.

```
jorge@jorge-Latitude-7350:~/Escritorio/microelectronica_tarea3$ make
VCD info: dumpfile ./bin/Sumadores.vcd opened for output.
No. Suma =           1: Operador A =   0, Operador B =  54, Sumador_1 =  54, Sumador_2 =  54, Sumador_3= 54
No. Suma =           2: Operador A = 221, Operador B =  68, Sumador_1 =  33, Sumador_2 =  33, Sumador_3= 33
No. Suma =           3: Operador A = 139, Operador B = 209, Sumador_1 =  92, Sumador_2 =  92, Sumador_3= 92
No. Suma =           4: Operador A = 141, Operador B = 185, Sumador_1 =  70, Sumador_2 =  70, Sumador_3= 70
No. Suma =           5: Operador A = 214, Operador B =  19, Sumador_1 = 233, Sumador_2 = 233, Sumador_3=233
No. Suma =           6: Operador A = 226, Operador B = 110, Sumador_1 =  80, Sumador_2 =  80, Sumador_3= 80
No. Suma =           7: Operador A =  72, Operador B =  22, Sumador_1 =  94, Sumador_2 =  94, Sumador_3= 94
No. Suma =           8: Operador A = 101, Operador B =  53, Sumador_1 = 154, Sumador_2 = 154, Sumador_3=154
No. Suma =           9: Operador A = 177, Operador B =  53, Sumador_1 = 230, Sumador_2 = 230, Sumador_3=230
No. Suma =          10: Operador A = 192, Operador B =  84, Sumador_1 =  20, Sumador_2 =  20, Sumador_3= 20
No. Suma =          11: Operador A = 162, Operador B = 199, Sumador_1 = 105, Sumador_2 = 105, Sumador_3=105
No. Suma =          12: Operador A =  45, Operador B = 246, Sumador_1 =  35, Sumador_2 =  35, Sumador_3= 35
No. Suma =          13: Operador A = 143, Operador B =  85, Sumador_1 = 228, Sumador_2 = 228, Sumador_3=228
No. Suma =          14: Operador A =  37, Operador B = 171, Sumador_1 = 208, Sumador_2 = 208, Sumador_3=208
No. Suma =          15: Operador A = 247, Operador B = 154, Sumador_1 = 145, Sumador_2 = 145, Sumador_3=145
No. Suma =          16: Operador A =  76, Operador B =  65, Sumador_1 = 141, Sumador_2 = 141, Sumador_3=141
No. Suma =          17: Operador A =  53, Operador B =  59, Sumador_1 = 112, Sumador_2 = 112, Sumador_3=112
No. Suma =          18: Operador A = 206, Operador B = 152, Sumador_1 = 102, Sumador_2 = 102, Sumador_3=102
No. Suma =          19: Operador A =  12, Operador B = 175, Sumador_1 = 187, Sumador_2 = 187, Sumador_3=187
No. Suma =          20: Operador A =  30, Operador B = 198, Sumador_1 = 228, Sumador_2 = 228, Sumador_3=228
No. Suma =          21: Operador A = 142, Operador B =  93, Sumador_1 = 235, Sumador_2 = 235, Sumador_3=235
No. Suma =          22: Operador A = 104, Operador B =  32, Sumador_1 = 136, Sumador_2 = 136, Sumador_3=136
No. Suma =          23: Operador A = 103, Operador B =  82, Sumador_1 = 185, Sumador_2 = 185, Sumador_3=185
No. Suma =          24: Operador A = 133, Operador B = 131, Sumador_1 =   8, Sumador_2 =   8, Sumador_3=  8
No. Suma =          25: Operador A =  57, Operador B = 153, Sumador_1 = 210, Sumador_2 = 210, Sumador_3=210
No. Suma =          26: Operador A = 216, Operador B = 119, Sumador_1 =  79, Sumador_2 =  79, Sumador_3= 79
No. Suma =          27: Operador A =  10, Operador B = 166, Sumador_1 = 176, Sumador_2 = 176, Sumador_3=176
No. Suma =          28: Operador A = 222, Operador B = 126, Sumador_1 =  92, Sumador_2 =  92, Sumador_3= 92
No. Suma =          29: Operador A = 115, Operador B = 202, Sumador_1 =  61, Sumador_2 =  61, Sumador_3= 61
No. Suma =          30: Operador A = 214, Operador B =  47, Sumador_1 =   5, Sumador_2 =   5, Sumador_3=  5
```

Figura 2: Captura de la terminal cuando se ejecuta la simulación de la parte 1 (*make parte1*), note que las entradas *Operador A* y *Operador B* son las mismas para todos los sumadores.

También puede verificarse por medio de *Gtkwave*, en la figura 3 se muestran algunos resultados para varias entradas, en efecto las salidas son las correctas pero puede notarse un pequeño detalle, el resultado no aparece directamente, esto ocurre debido al retardo de los componentes que provocan que las señales no sean procesadas al mismo tiempo por las compuertas, sino que las compuertas procesan valores diferentes hasta que la señal se estabilice.

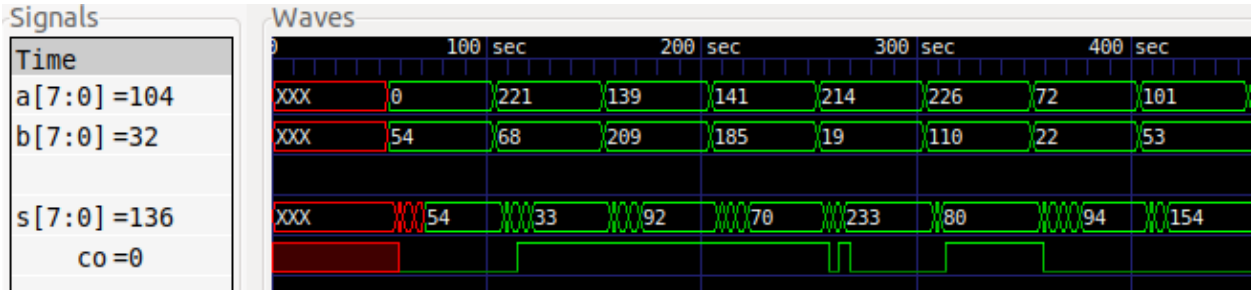

Figura 3: Diagrama de señales del sumador de rizado procesando algunos valores de entrada para verificar su comportamiento.

En la tabla 1 se muestran los resultados de la simulación, en la misma se puede ver que el promedio de transiciones bajo-alto más reducido corresponde al sumador lookahead para los 4 casos (500, 1000, 2000 y 5000 sumas) mientras que el sumador lógico es el que más potencia consume. También puede notarse que para esta parte el sumador con menor retardo es el sumador lógico (se tomó el valor de retardo más alto de cada sumador y de ellos se seleccionó el menor).

Resulta interesante observar el cambio en el retardo cuando se introducen las entradas $A = FF$ y $B = 01$ respecto a las otras dos combinaciones de entradas, si se analiza con cuidado este comportamiento se da porque, cuando se da la suma $A = FF + B = 01$ todos los bits en la salida deben esperar a la entrada C_{out} del sumador completo anterior, esto hace que haya que esperar a que el sumador completo anterior termine de procesar las entradas para así desplegar la salida correcta en el sumador completo actual, mismo que introduce un retardo al sumador completo siguiente, y así con todos los sumados completos.

		Sumador de Rizado		
	Sumas	**seed 1**	**seed 2**	**seed 3**
Potencia	**500**	488820	503640	494880
	1000	1022040	1009860	977340
	2000	1958460	2002740	1998240
	5000	5085480	5070660	5042580
	Entradas			
Retardo	**A=B=0**	4.0		
	A=0, B=1	4.0		
	A=FF, B=1	37.0		

		Sumador lógico		
	Sumas	**seed 1**	**seed 2**	**seed 3**
Potencia	**500**	564660	576780	568380
	1000	1167300	1168860	1151580
	2000	2278020	2306220	2284320
	5000	5837520	5804040	5803920
	Entradas			
Retardo	**A=B=0**	6.0		
	A=0, B=1	6.0		
	A=FF, B=1	22.0		

		Sumador lookahead		
	Sumas	**seed 1**	**seed 2**	**seed 3**
Potencia	**500**	476100	496320	483720
	1000	1004400	998460	986460
	2000	1945920	1970160	1981860
	5000	5052840	5011680	5015820
	Entradas			
Retardo	**A=B=0**	6.0		
	A=0, B=1	6.0		
	A=FF, B=1	23.0		

Cuadro 1: Tabla que recopila la salida de la terminal al ejecutar el código de la parte 1, aquí están representados los sumadores de rizado, lógico y de lookahead respectivamente.

2.2. Parte 2

A pesar del cambio en los retardos de los componentes los sumadores continúan funcionando correctamente, en la figura 4 se puede ver como el sumador de rizado despliega correctamente el resultado adecuado.

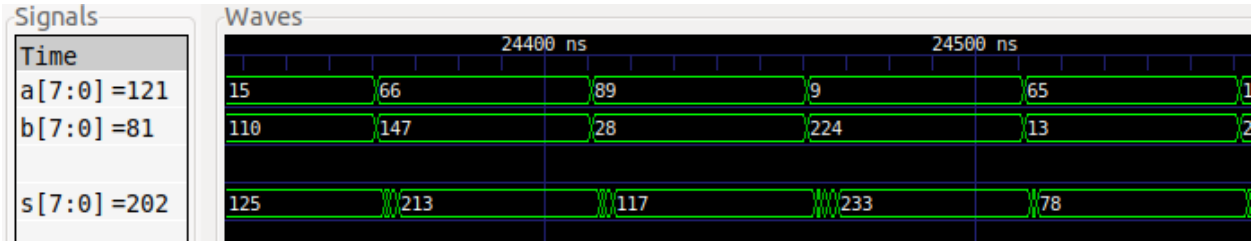

Figura 4: Diagrama de señales del sumador de rizado que muestra la salida del mismo en base a las entradas A y B.

En este caso todos los componentes tienen el mismo valor de retardo del inversor, al ejecutar la misma prueba de la parte 1 se obtuvieron los resultados que se muestran en la tabla 2, ahora el sumador que menos potencia promedio consume es el sumador de rizado mientras que el que más consume es el sumador lógico. Ahora el sumador lookahead es el que presenta un menor retardo (utilizando el mismo criterio que se utilizó en la parte 1, se tomó el mayor valor de retardo que presenta cada compuerta para cada valor de entrada y se tomó el sumador que presenta el valor más pequeño).

		Sumador de Rizado		
	Sumas	**seed 1**	**seed 2**	**seed 3**
Potencia	**500**	488820	503640	494880
	1000	1022040	1009860	977340
	2000	1958460	2002740	1998240
	5000	5085480	5070660	5042580
	Entradas			
Retardo	**A=B=0**	2.0		
	A=0, B=1	2.0		
	A=FF, B=1	15.0		

		Sumador lógico		
	Sumas	**seed 1**	**seed 2**	**seed 3**
Potencia	**500**	615180	622380	620100
	1000	1266480	1254120	1238760
	2000	2471460	2498760	2476560
	5000	6325320	6296520	6292620
	Entradas			
Retardo	**A=B=0**	3.0		
	A=0, B=1	3.0		
	A=FF, B=1	6.0		

		Sumador lookahead		
	Sumas	**seed 1**	**seed 2**	**seed 3**
Potencia	**500**	524640	546420	528300
	1000	1102560	1102560	1078440
	2000	2130240	2159820	2171460
	5000	5547660	5502840	5493060
	Entradas			
Retardo	**A=B=0**	2.0		
	A=0, B=1	2.0		
	A=FF, B=1	6.0		

Cuadro 2: Tabla que recopila la salida de la terminal al ejecutar el código de la parte 2, aquí están representados los sumadores de rizado, lógico y de lookahead respectivamente.

2.3. Parte 3

En la figura 5 se verifica que el sumador de rizado funciona correctamente a pesar del cambio hecho en la definición estructural.

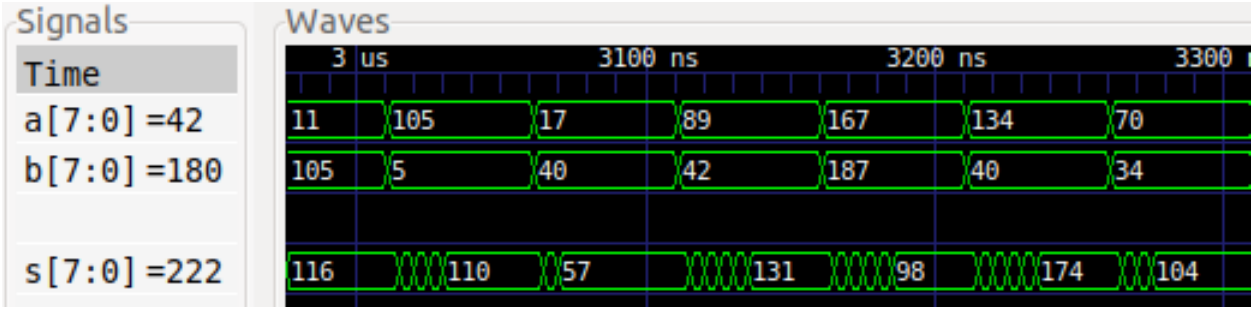

Figura 5: Diagrama de señales que muestra el comportamiento del sumador de rizado en base a las señales de entrada A y B, durante esta simulación se utilizaron los retardos originales de la biblioteca de componentes.

Ahora los valores de retardo son los originales pero se cambió la estructura del sumador de rizado, donde se sustituyeron dos compuertas XOR de 2 entradas por una de 3 entradas, luego se aplicó la prueba de la parte 1. Nuevamente el sumador de lookahead es el que menos potencia consume y el sumador lógico el que más consume además de ser el sumador con menor retardo (22.0).

Aquí puede notarse que el retardo del sumador de rizado es mayor (39.0) que el valor obtenido en la parte 1, esto ocurre porque no necesariamente menos etapas implican menor tiempo de retardo, de hecho el esfuerzo lógico explica que es más eficiente tener varias etapas (las óptimas, es decir, una infinitud de etapas NO implica retardo nulo) ya que el procesamiento de las señales se divide entre todas las etapas que componen el camino disminuyendo así el retardo y aumento la capacitancia de salida.

		Sumador de Rizado		
	Sumas	**seed 1**	**seed 2**	**seed 3**
Potencia	**500**	555240	571800	561840
	1000	1158480	1146360	1108080
	2000	2218320	2274960	2266200
	5000	5767680	5755560	5720640
	Entradas			
Retardo	**A=B=0**	4.0		
	A=0, B=1	4.0		
	A=FF, B=1	39.0		

		Sumador lógico		
	Sumas	**seed 1**	**seed 2**	**seed 3**
Potencia	**500**	564660	576780	568380
	1000	1167300	1168860	1151580
	2000	2278020	2306220	2284320
	5000	5837520	5804040	5803920
	Entradas			
Retardo	**A=B=0**	6.0		
	A=0, B=1	6.0		
	A=FF, B=1	22.0		

		Sumador lookahead		
	Sumas	**seed 1**	**seed 2**	**seed 3**
Potencia	**500**	476100	496320	483720
	1000	1004400	998460	986460
	2000	1945920	1970160	1981860
	5000	5052840	5011680	5015820
	Entradas			
Retardo	**A=B=0**	6.0		
	A=0, B=1	6.0		
	A=FF, B=1	23.0		

Cuadro 3: Tabla que recopila la salida de la terminal al ejecutar el código de la parte 3, aquí están representados los sumadores de rizado, lógico y de lookahead respectivamente.

3. Conclusiones

- El sumador se pudo diseñar en base a las especificaciones dadas en el enunciado del proyecto.
- Se pudo verificar que la respuesta de entrada-salida del sumador cambió (aunque poco) cuando se modificó ligeramente su definición estructural.
- El sumador lógico es el que presenta mayor consumo de potencia en todos los casos, esto es así porque el diseño estructural de este sumador contiene más compuertas lógicas que los de los otros sumadores
- El sumador lógico es el que presenta el menor retardo (si tomamos el peor retardo mas pequeño) excepto cuando se modificaron los tiempos de retardo de los componentes a 1 (parte 2, en este caso el de menor retardo fué el sumador de *lookahead*).

4. Recomendaciones

- Cuando se utilice el comando *generate* de verilog hay que tener cuidado con las conexiones de los componentes generados ya que *generate* no lo hace automáticamente.
- Hay que tener en cuenta que el consumo de potencia se mide únicamente en las transiciones de bajo a alto, el valor que despliega la terminal para la sección de consumo de potencia es en efecto el conteo de dichas transiciones.
- Repasar el concepto de retardo en las compuertas lógicas y como este puede afectar la salida de una función lógica, esto es pecialmente importante cuando se tienen circuitos sincrónicos.

Referencias

[1] D. K. Tala. (2010) Verilog tutorial. http://classweb.ece.umd.edu/enee359a/verilog_tutorial.pdf.