Escuela de Ingeniería Electrónica
Licenciatura en Ingeniería Electrónica
EL3313 Taller de Diseño Digital
II Semestre 2021

TEC | Tecnológico de Costa Rica

Informe de Laboratorio

Lab. 2: Introducción a los lenguajes de descripción de hardware - Lógica combinacional

Bosco Richmond, Francis Guindon, Christian Arguedas

¿en qué tiempo se debe escribir?

Resumen

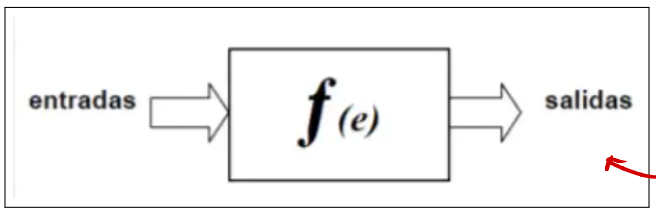
En este laboratorio se realizó una introducción al lenguaje de descripción de hardware Verilog utilizando lógica combinacional mediante la plataforma Vivado, donde se realizó la simulación pre y post síntesis de un multiplexor 4 a 1, un codificador/decodificador de binario a Gray, un sumador optimizado y una ALU, así como también sus respectivos testbench para comprobar el correcto funcionamiento de los mismos. Logrando diseños funcionales para los requerimientos establecidos en el instructivo de este laboratorio.

1. Cuestionario previo

1. Explique qué es el modelado de comportamiento y de estructura en diseño digital. Brinde un ejemplo de cada uno.

   El modelado de comportamiento por medio de bloques describen a los circuitos y/o sistemas de manera funcional contando con velocidad de ejecución a la hora de simular al poseer menos variables. En resumen importa la función que relaciona la entrada con la salida fig.1.

Figura 1



¿de dónde tomaron la imagen?

Figura 1: Modelado de comportamiento

El modelado de estructura en su parte en vez de representar procesos, muestra la estructura o el resultado del proceso donde jerarquizan con bloques más pequeños que se conectan en un circuito con su descripción partiendo de procesos inferiores a superiores.

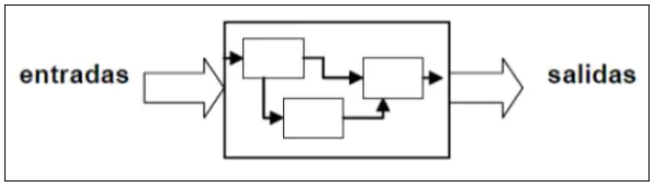


Figura 2: Modelado esctuctural

En las fig.3 y fig.4 se puede ver un ejemplo de código en Verilog para cada uno de los modelados.

```
architecture Algoritmico of Mux21 is
begin
   process (a,b,ctrl)
   begin
      if (ctrl = '0') then
         z <= a;
      else
         z <= b;
      end if;
   end process;
end Algoritmico;
```

Figura 3: Ejemplo de código con modelado de comportamiento

```
architecture Estructural of Mux21 is
   signal ctrl_n,n1,n2 : bit;
   component INV
      port (y : in bit;
            z : out bit);
   end component;
   component AND2
      port (x,y : in bit;
            z : out bit);
   end component;
   component OR2
      port (x,y : in bit;
            z : out bit);
   end component;
   begin
      U0: INV port map (ctrl,ctrl_n);
      U1: AND2 port map (ctrl_n,a,n1);
      U2: AND2 port map (ctrl,b,n2);
      U3: OR2 port map (n1,n2,z);
end Estructural;
```

Figura 4: Ejemplo de código con modelado de estructura

2. Explique el proceso de síntesis lógica en el diseño de circuitos digitales.

   El proceso de sintesis lógica consiste en la especificación de una tarea en instrucciones de alto nivel hacia una implementación de un nivel más bajo, por lo general llamado RTL (Register Transfer Level), la síntesis es una tarea vertical entre niveles de abstracción, del nivel más alto en la jerarquía de diseño se va hacia el más bajo nivel de la jerarquía [1].

   Se requiere un proceso de síntesis para pasar el algoritmo, de un lenguaje de nivel alto a nivel RTL. Al mismo tiempo que se implementa el código del sistema, se simula el sistema con los modelos de funcionamiento sintetizados a RTL del código, siendo esta tarea transparente al programador. De este modo se consigue el modelado de todo el sistema, con el código escrito en un lenguaje de alto nivel [1].

3. Investigue sobre la tecnología de FPGAs. Describa el funcionamiento de la lógica programable en general, así como los componentes básicos de una

   Las FPGAs (Field Programmable Gate Arrays) consisten en una matriz bidimencional de bloques configurables que se pueden conectar mediante recursos generales de interconexion los cuales incluyen segmentos de pista de diferentes longitudes y unos conmutadores programables para enlazar bloques a pistas o pistas entre sí [2].

   Existen cuatro grandes familias dependiendo de la estructura que adoptan los bloques lógicos que tengan definidos los cuales son los siguientes [2]:

   - Matriz simétrica.
   - Basada en canales.
   - Mar de puertas.
   - PLD jerárquica.

   La lógica programable general de las FPGAs (Field Programmable Gate Arrays) consta de las siguientes etapas basicas[2]:

   - Dividir el circuito en bloques básicos, asignándolos a los bloque configurables del dispositivo.
   - Conectar los bloques de lógica mediante los conmutadores necesarios.

   Los elementos básicos que constituyen una FPGA se pueden ver en la Fig. 5 y se enumeran a continuación [2]:

   - Bloques lógicos, cuya estructura y contenido se denomina arquitectura.
   - Recursos de interconexión, cuya estructura y contenido se denomina arquitectura de rutado.
   - Memoria RAM, que se carga durante el RESET para configurar bloques y conectarlos.

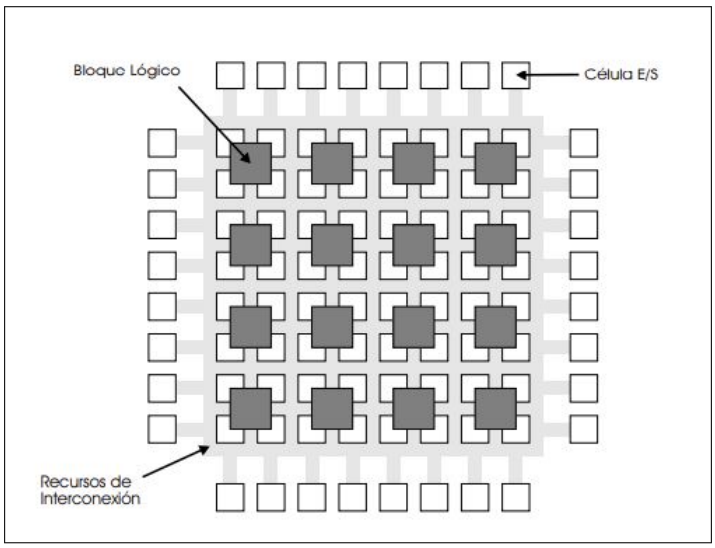


Figura 5: Componentes básicos de una FPGA [2] Tomado de...

Las FPGAs ofrecen dos principales ventajas las cuales son el bajo coste de prototipado y el corto tiempo de producción. Sus principales desventajas se encuentran su baja velocidad de operación y baja densidad lógica [2].

2. Problema 1. Multiplexor 4-to-1

Para la realización del diseño del multiplexor se establecieron las entradas en Verilog parametrizables de manera que se pueda cambiar el ancho de los datos de entrada para que este funcione para 4, 8 y 16 bits. La salida se asignó mediante la selección de las entradas binarias de dos bits.

En el testbench se realizó el cambio del parámetro del ancho de los datos para poder observar el funcionamiento del multiplexor con para 4, 8 y 16 bits. También se fijaron valores aleatorios para las entrada y un contador que funciona como entradas de selección para poder observar el comportamiento completo del multiplexor, para ver estos datos más cómodamente se realizó una tabla en la consola con el comando display.

¿Algún diagrama? ¿Qué resultados se obtuvieron?

3. Problema 2. Codificador / Decodificador

Se realizó el siguiente diagrama para desarrollar el codificador de gray a binario en la Tabla 1 a partir de la Figura 6 para el codificador binario a gray y la figura 7 para el decodificador gray a binario.

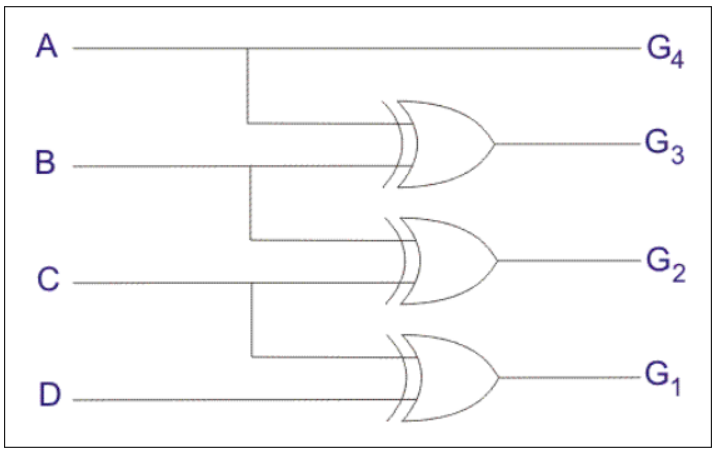


Figura 6: Esquema para el circuito lógico del codificador Binario a Gray

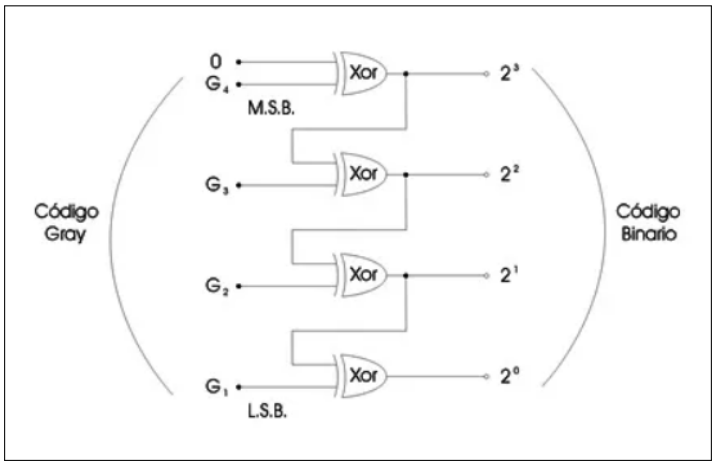


Figura 7: Esquema para el decodificador Gray a Binario extraido de [3]

Para el decodificador binario a gray se obtiene la tabla anterior desde la persepectiva de Gray llegando a los mismos resultados al decodificar a binario.

Al combinarse tanto el codificador y el decodificador se llega al mismo resultado en orden binario contando que en post sintésis se da un ligero retardo en la salida pero en la mayor parte del tiempo se respeta.

Faltan resultados acá que lo demuestren

Tabla 1: Tabla de verdad de la función lógica

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

4. Problema 3. Sumador optimizado

Para desarrollar el sumador completo de 1-bit se partió de la tabla de verdad de la salida de la suma y del acarreo de salida, las cuales se muestran en la Tabla 2 donde A y B representan las entradas a sumar, C el acarreo de entrada.

A	B	C	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabla 2: Tabla de verdad para el sumador de 1-bit

Habiendo obtenido la tabla se procedió a montar los mapas de Karnaugh, en la Fig. 8 se muestra el mapa para la salida de la suma y en la Fig. 9 el mapa para el acarreo de salida. Debido a que los mapas no reconocen la función XOR, partiendo de su definición (1), se realizó la simplificación de la función obtenida mediante algebra booleana como se muestra en los procedimientos para S y C_{out}.

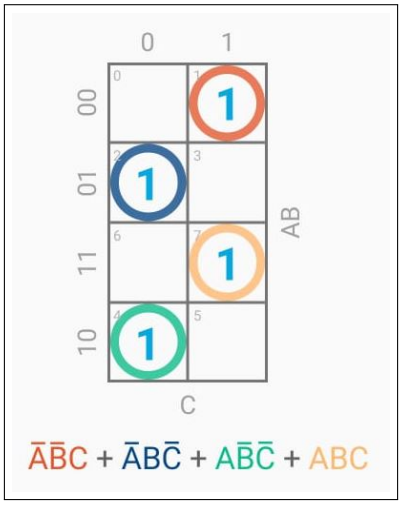


Figura 8: Mapa para S

$$A \oplus B = A\bar{B} + \bar{A}B = (A + B)(\bar{A} + \bar{B}) \tag{1}$$

$$\begin{aligned}
\bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC &= (\bar{A}\bar{B} + AB)C + (\bar{A}B + A\bar{B})\bar{C} \\
S &= (\overline{A + B} + \overline{\bar{A} + \bar{B}})C + (A \oplus B)\bar{C} \\
S &= \overline{(A + B)(\bar{A} + \bar{B})} + (A \oplus B)\bar{C} \\
S &= \overline{(A \oplus B)}\, C + (A \oplus B)\bar{C} \\
S &= (A \oplus B) \oplus C
\end{aligned}$$

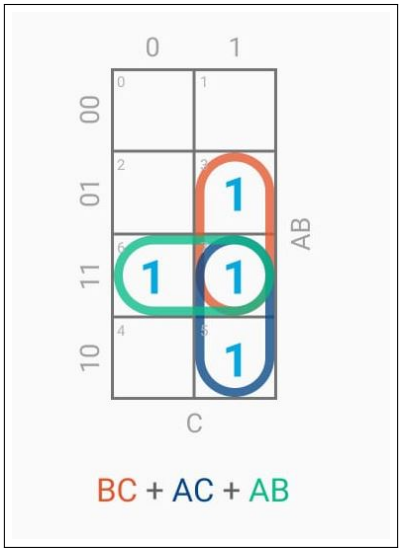


Figura 9: Mapa para C_{out}

$$BC + AC + AB = (B + A)\overline{AB}C + AB$$
$$C_{out} = (B + A)(\bar{A} + \bar{B})C + AB$$
$$C_{out} = (A \oplus B)C + AB$$

Con las funciones logicas obtenidas se procedió a armar el diagrama de bloques correspondiente para luego implementarlo en Vivado, este diagrama se puede observar en la Fig. 10

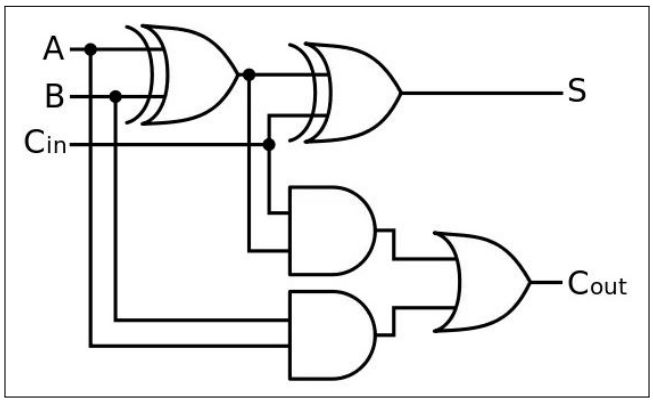


Figura 10: Diagrama del sumador de 1-bit

Luego se construyó el sumador completo de 8 bits al conectar 8 sumadores de 1 bit en cascada, como se muestra en la Fig. 11

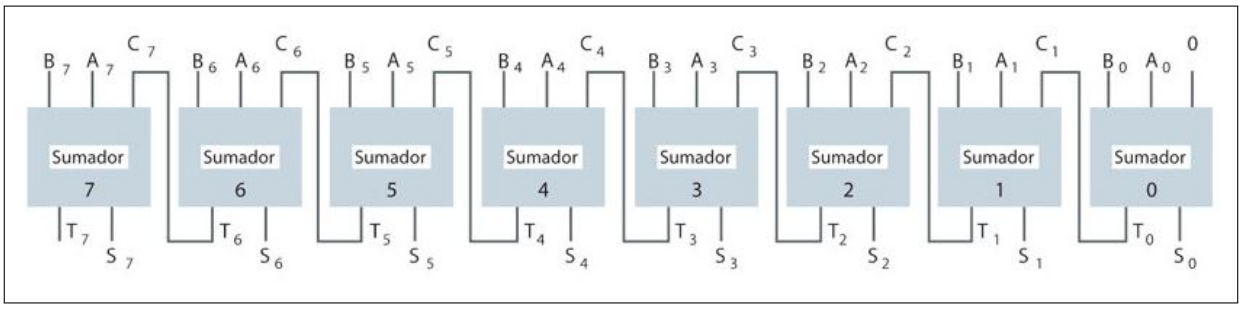


Figura 11: Sumador de 8 bits.

Finalmente se implementó el diseño en Vivado mediante Verilog y se realizó un testbench para evaluar el funcionamiento del sumador, tomando en cuenta los casos en los que el sumador produce overflow, para esto se realizó una tabla en la consola del programa se guardaron los datos de la simulación en un archivo .log el cual permite observar los 2^{17} resultados posibles, ya que no es posible observar esta cantidad en la consola de Vivado. En la Fig. 12 se observa parte del contenido del archivo creado el cual fue abierto en una terminal de Linux, a la izquierda se presentan los primeros datos y a la derecha los últimos donde se produce overflow.

Otra particularidad que ocurrió durante la implementación del sumador fue que al tener una gran cantidad de posibles resultados se tuvo que cambiar el tiempo de simulación ya que que Vivado tiene por defecto 100 μs y el tiempo estimado para la simulación con la resolución fijada de 10 ns fue de 1.31072 ms. En la Fig. 13 se muestra el cambio realizado en el tiempo de simulación.

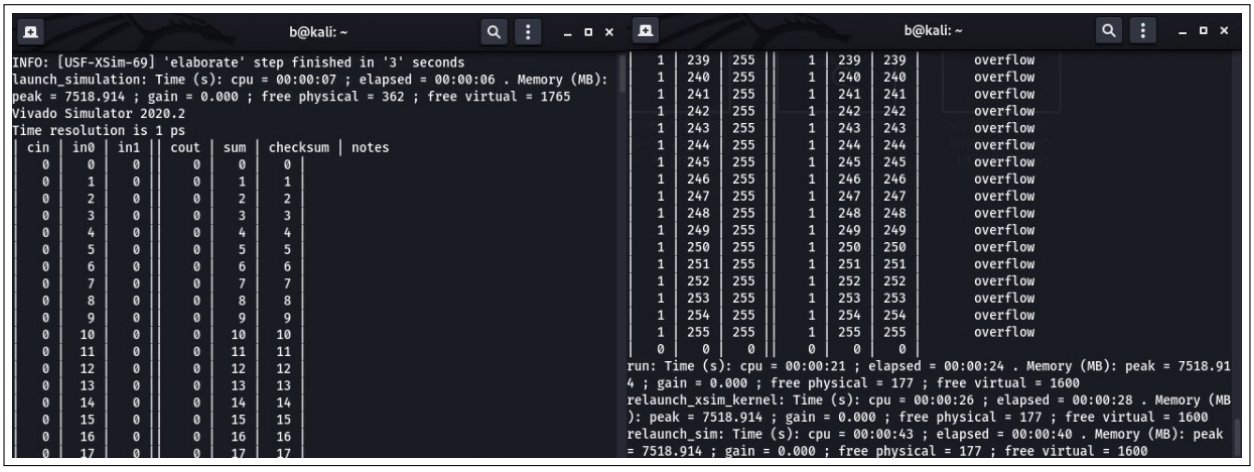


Figura 12: Datos del sumador en la terminal de Linux.

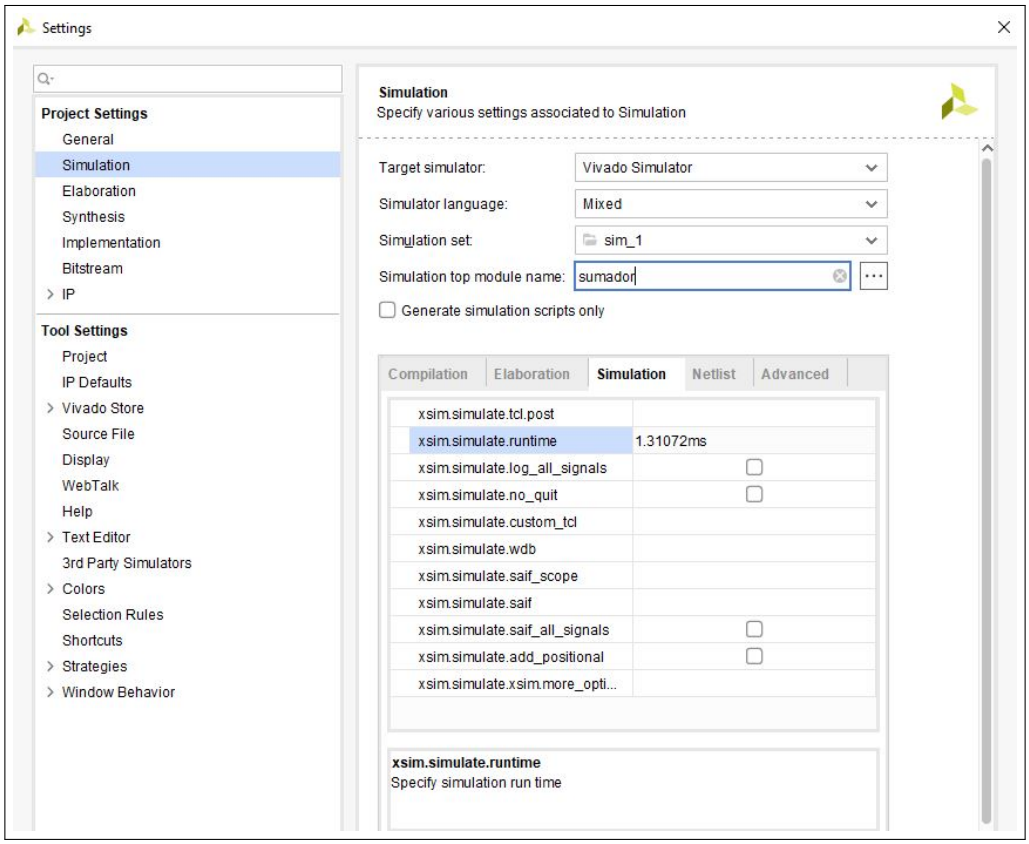


Figura 13: Especificación del tiempo de simulación en Vivado.

5. Problema 4. Unidad aritmética lógica (ALU)

Se realizó una ALU parcialmente parametrizable de n bits con el diagrama de alto nivel mostrado en la figura 14.

no hay otros diagramas que muestren los diferentes módulos internos.

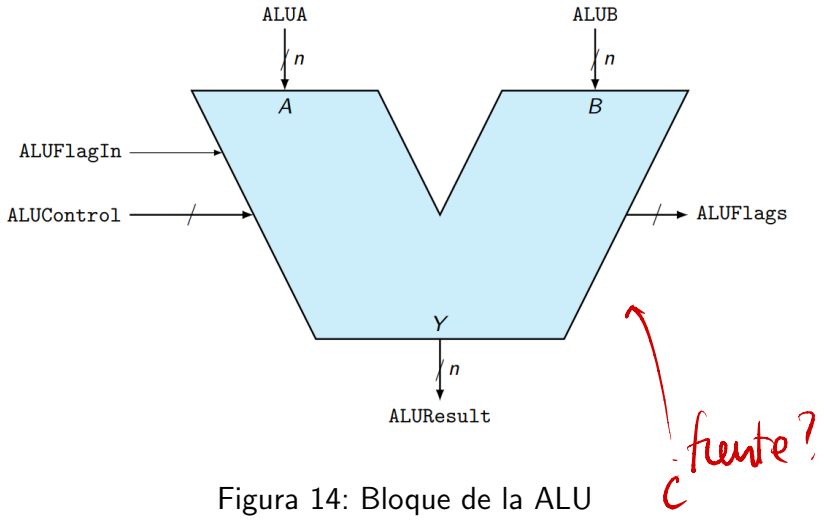


Figura 14: Bloque de la ALU

Las operaciones que se pueden realizar son las siguientes, según la opción de `ALUControl`:

- `0H - and`
- `1H - or`
- `2H - suma`
- `3H - incrementar en uno`
- `4H - decrementar en uno`
- `5H - not`
- `6H - resta`
- `7H - xor`
- `8H - corrimiento a la izquierda`
- `9H - corrimiento a la derecha`

Las operaciones parametrizables son las lógicas y de incremento/decremento. Las operaciones de suma y resta son dependientes de un sumandor de carry-lookahead interno limitado a 4 bits. El esquemático del sumador de carry-lookahead se muestra en la figura 15. Se utiliza este sumador para que la suma sea más rápido que realizarla por el método del sumador completo en cascada o sumandor de ripple-carry.

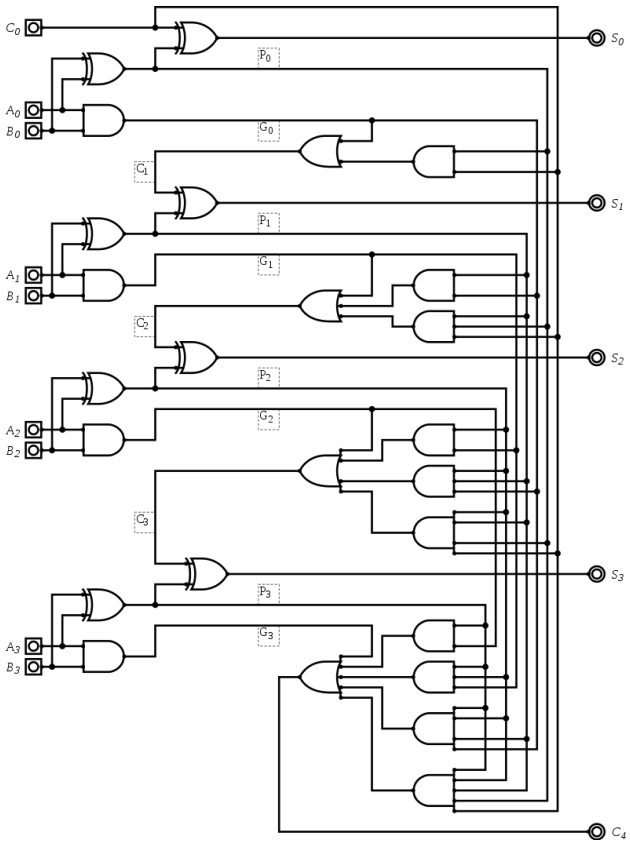


Figura 15: Esquemático de Sumador de Carry Lookahead

La ALU se implementó con lógica combinacional usando `assign` y el operador condicional `?:`. Los resultados del testbench imprimen a la consola una serie de pruebas para cada operación de la ALU. El resultado se muestra en la figura 16. Además los resultados incluyen una columna de valores esperados para la salida de la ALU, y en caso de haber una diferencia entre el valor resultante y el valor esperado, se imprime un mensaje de error al final de la línea.

```
    A |    B |     FlagIn |    Control ||  Resultado |   Esperado | C | Z |
AND---------------------------------------
 0000 | 1111 | 0000000000 | 0000000000 || 0000000000 | 0000000000 | 0 | 1 |
 0110 | 1101 | 0000000000 | 0000000000 || 0000000004 | 0000000004 | 0 | 0 |
OR----------------------------------------
 0000 | 1111 | 0000000000 | 0000000001 || 000000000f | 000000000f | 0 | 0 |
 0100 | 1101 | 0000000000 | 0000000001 || 000000000d | 000000000d | 0 | 0 |
SUMA--------------------------------------
 0100 | 0001 | 0000000000 | 0000000002 || 0000000005 | 0000000005 | 0 | 0 |
 1001 | 0011 | 0000000000 | 0000000002 || 000000000c | 000000000c | 0 | 0 |
 1101 | 1101 | 0000000000 | 0000000002 || 000000000a | 000000000a | 0 | 0 |
 0101 | 0010 | 0000000001 | 0000000002 || 0000000008 | 0000000008 | 0 | 0 |
 0001 | 1101 | 0000000001 | 0000000002 || 000000000f | 000000000f | 0 | 0 |
 0110 | 1101 | 0000000001 | 0000000002 || 0000000004 | 0000000004 | 0 | 0 |
INCREMENTAR-------------------------------
 0000 | 1111 | 0000000000 | 0000000003 || 0000000001 | 0000000001 | 0 | 0 |
 0100 | 1101 | 0000000001 | 0000000003 || 000000000e | 000000000e | 0 | 0 |
DECREMENTAR-------------------------------
 0000 | 1111 | 0000000000 | 0000000004 || 000000000f | 000000000f | 0 | 0 |
 0100 | 1101 | 0000000001 | 0000000004 || 000000000c | 000000000c | 0 | 0 |
NOT---------------------------------------
 0000 | 1111 | 0000000000 | 0000000005 || 000000000f | 000000000f | 0 | 0 |
 0100 | 1101 | 0000000001 | 0000000005 || 0000000002 | 0000000002 | 0 | 0 |
RESTA-------------------------------------
 1101 | 1100 | 0000000000 | 0000000006 || 0000000001 | 0000000001 | 0 | 0 |
 1001 | 0110 | 0000000000 | 0000000006 || 0000000003 | 0000000003 | 0 | 0 |
 0101 | 1010 | 0000000000 | 0000000006 || 000000000b | 000000000b | 0 | 0 |
 0101 | 0111 | 0000000001 | 0000000006 || 000000000f | 000000000f | 0 | 0 |
 0010 | 1111 | 0000000001 | 0000000006 || 0000000004 | 0000000004 | 0 | 0 |
 0010 | 1110 | 0000000001 | 0000000006 || 0000000005 | 0000000005 | 0 | 0 |
XOR---------------------------------------
 0000 | 1111 | 0000000001 | 0000000007 || 000000000f | 000000000f | 0 | 0 |
 0101 | 1100 | 0000000001 | 0000000007 || 0000000009 | 0000000009 | 0 | 0 |
CORRIMIENTO IZQUIERDA---------------------
 1010 | 0000 | 0000000000 | 0000000008 || 000000000a | 000000000a | 0 | 0 |
 1010 | 0001 | 0000000000 | 0000000008 || 0000000004 | 0000000004 | 1 | 0 |
 1010 | 0010 | 0000000000 | 0000000008 || 0000000008 | 0000000008 | 0 | 0 |
 1010 | 0011 | 0000000000 | 0000000008 || 0000000000 | 0000000000 | 1 | 1 |
 1010 | 0000 | 0000000001 | 0000000008 || 000000000a | 000000000a | 0 | 0 |
 1010 | 0001 | 0000000001 | 0000000008 || 0000000005 | 0000000005 | 1 | 0 |
 1010 | 0010 | 0000000001 | 0000000008 || 000000000b | 000000000b | 0 | 0 |
 1010 | 0011 | 0000000001 | 0000000008 || 0000000007 | 0000000007 | 1 | 0 |
CORRIMIENTO DERECHA-----------------------
 1010 | 0000 | 0000000000 | 0000000009 || 000000000a | 000000000a | 0 | 0 |
 1010 | 0001 | 0000000000 | 0000000009 || 0000000005 | 0000000005 | 0 | 0 |
 1010 | 0010 | 0000000000 | 0000000009 || 0000000002 | 0000000002 | 1 | 0 |
 1010 | 0011 | 0000000000 | 0000000009 || 0000000001 | 0000000001 | 0 | 0 |
 1010 | 0000 | 0000000001 | 0000000009 || 000000000a | 000000000a | 0 | 0 |
 1010 | 0001 | 0000000001 | 0000000009 || 000000000d | 000000000d | 0 | 0 |
 1010 | 0010 | 0000000001 | 0000000009 || 000000000e | 000000000e | 1 | 0 |
 1010 | 0011 | 0000000001 | 0000000009 || 000000000f | 000000000f | 0 | 0 |
```

Figura 16: Resultados de pruebas del testbench

Se puede observar que para las pruebas realizadas cada operación de la ALU funciona correctamente.

6. Conclusiones

Durante el laboratorio se logró corroborar el comportamiento del multiplexor tanto para 4,8 y 16 bits contabilizando un ligero error al inicio de la post síntesis en la salida. Eventualmente se codifico de binario a gray y viceversa empleando puertas XOR comprobando sé el resultado

deseado además de contar con un ligero retraso en la post síntesis en el resultado pero respetando la mayor parte del tiempo lo esperado con un ligero retraso. Para el sumador se empleo primero el desarrollo de un sumador completo para un bit y empleando una conexión de cascada se logro optimizar y obtener un sumador completo para 8 bits y finalmente el ALU se logró emplear las distintas operaciones según el valor el ALUControl con un retraso igualmente debido en la post síntesis. Se logró implementar todos los módulos de forma combinacional.

Referencias

[1] R. A. Buj, "Procedimiento de diseño de circuitos digitales mediante FPGAs," tech. rep., Universidad de Lleida, Mar. 2007. [Online] https://repositori.udl.cat/bitstream/handle/10459.1/45767/Buj.pdf?sequence=1isAllowed=y.

[2] M. L. López and J. L. Ayala, *FPGA: Nociones básicas e implementación*. Universidad Politécnica de Madrid, Apr. 2004. [Online] https://repositori.udl.cat/bitstream/handle/10459.1/45767/Buj.pdf?sequence=1isAllowed=y.

[3] Contaval, *Código numérico Gray*. Contaval, 2017. [Online] https://www.contaval.es/codigo-numerico-gray/.

60/

2. Informe presentado en el formato provisto para este curso — 24/40 %

   Se distribuirá el peso de reporte en:

   a) Cuestionario previo — 3/5 %
   b) Descripción funcional de unidades desarrolladas — 15/20 %
   c) Datos y resultados (tablas, gráficos, simulaciones, reportes de herramientas) — 3/5 %
   d) Análisis de datos y resultados, compaciones. — 1/5 %

      Detalle los errores y complicaciones del diseño, si se resolvieron y cómo se resolvieron. Si no se resolvieron, ¿cómo se resolverían?

   e) Conclusiones — 2/5 %

   Recuerde citar adecuadamente en su reporte cualquier información proveniente de fuentes bibliográficas.

OBSERVACIONES

1. No hay análisis de resultados. Además se presentan pocos resultados
2. Por lo tanto, se concluye sobre lo que no se analizó.
3. Faltan diagramas de bloques
4. Cuidar ortografía