Taller de Lógica Digital

Organización del Computador 1

Primer Cuatrimestre 2023

1. Enunciado

El Taller consistirá en desarrollar todos los circuitos combinatorios necesarios para realizar una ALU de 4 *bits*.

Para la realización de este taller deberá utilizar el archivo `TallerLogica-Combinatorios.circ` que puede descargar de la página de la materia.

2. Logisim Evolution

2.1. El simulador

El simulador[1] se encuentra instalado en cada computadora de los laboratorios. Para ejecutarlo, abrir una terminal y escribir el comando: `logisim-evolution`.

El simulador opera en dos modos. Edición y Simulación:

- En el modo edición podremos definir el funcionamiento del circuito con todas las entradas y salidas, las compuertas lógicas o los componentes que lo componen, más el aspecto físico que tendrá el componente a la hora de ser utilizado por otros circuitos.
- El modo simulación nos permitirá testear el funcionamiento del componente, asignando valores a las entradas y testeando el valor de las salidas.

2.2. Detalles adicionales

a) Las entradas (*inputs*) se simbolizan con un cuadrado y tienen el comportamiento de una llave. (Prendido = 1; Apagado = 0). Las mismas cambian de estado de forma manual. Se pueden definir de más de un *bit*.

b) Las salidas (outputs) se simbolizan con un círculo y serán "prendidas" cuando por su entrada haya un 1, o "apagadas" cuando por su entrada haya un 0. Se pueden definir de más de un *bit*.

c) En caso de ser necesario que el circuito tenga una entrada fija en algún valor, podremos utilizar el componente "Wiring/Constant".

d) Se recomienda *fuertemente* el uso de Etiquetas para facilitar la comprensión del funcionamiento de cada componente y que las Etiquetas tengan exactamente el mismo nombre que el enunciado. Para ello, seleccionar el componente y cambiar su atributo "Etiqueta".

e) El simulador permite usar cables de más de un bit. Para ello, pueden utilizar el componente "Wiring/Splitter", que permite unir o separar cables. Además, pueden usar el Multiplexor que se encuentra en "Plexers".

[1] https://github.com/logisim-evolution/logisim-evolution/

3. Antes de empezar

Completar la siguiente tabla, calculando **Res = Op1 + Op2** (es decir, la suma *bit a bit*). Interpretar los operandos y el resultado tanto en Sin Signo como en Complemento a 2, y decidir en cada caso si hubo *Overflow*.

Operandos		Sin signo					Complemento a 2				
Op1	Op2	$Op1_{10}$	$Op2_{10}$	Res (*bits*)	Res_{10}	V?	$Op1_{10}$	$Op2_{10}$	Res (*bits*)	Res_{10}	V?
1111	0001	15	1	0000	0	1	-1	1	0000	0	0
0001	1111	1	15	0000	0	1	1	-1	0000	0	0
0101	0101	5	5	1010	10	0	5	5	1010	-6	1
1000	0111	8	7	1111	15	0	-8	7	1111	-1	0
0110	1010	6	10	0000	0	1	6	-6	0000	0	0

Repetir el ejercicio anterior para **Res = Op1 - Op2**.

Operandos		Sin signo					Complemento a 2				
Op1	Op2	$Op1_{10}$	$Op2_{10}$	Res (*bits*)	Res_{10}	V?	$Op1_{10}$	$Op2_{10}$	Res (*bits*)	Res_{10}	V?
1000	0010	8	2	0110	6	0	-8	2	0110	6	1
0001	1111	1	15	0010	2	1	1	-1	0010	2	0
0101	0101	5	5	0000	0	0	5	5	0000	0	0
1000	0111	8	7	0001	1	0	-8	7	0001	1	1
0110	1010	6	10	1100	12	1	6	-6	1100	-4	1

4. Ejercicios

Completar el esqueleto provisto de los componentes que se enumeran a continuación. **Sólo podrán utilizarse compuertas lógicas básicas de un bit de dos entradas (AND, OR, XOR) y la compuerta NOT, de una entrada, *splitters* y multiplexores. Deberán desarrollar todos los componentes necesarios.** Deberán diseñar cada uno de los mismos de manera modular, incorporando una funcionalidad por componente.

Importante: Durante la resolución del taller se encontrarán con 3 *checkpoints*. Al llegar a cada uno de ellos deberán validar lo realizado hasta ese punto con algún docente antes de continuar.

a) **Sumador simple de 1 *bit***. El mismo tendrá dos entradas **a** y **b** y dos salidas: **S** que representa la suma de a y b y **Cout** que representará el acarreo de dicha suma.

b) **Sumador completo de 1 *bit***. El mismo tendrá tres entradas **a**, **b** y **Cin**. Este último representa el carry de entrada. Y dos salidas: **S** que representa la suma de a y b considerando el acarreo, y **Cout** que representará el acarreo de dicha suma.

c) **Sumador de 4 *bits***. El mismo tendrá dos entradas de cuatro *bits* **A** y **B**, que representarán los numerales a sumar y otra **Cin**, con la misma interpretación anterior. Por otra parte, el mismo tendrá dos salidas **S** (de cuatro *bits*), donde se deberá ver reflejado el resultado y **Cout** que representará el acarreo final de la suma.

Checkpoint 1

d) **Comparador con Cero, de 4 *bits***. El mismo tendrá una entrada **A** de cuatro *bits* y una salida **Z** que deberá encenderse cuando la entrada es 0000.

e) **Sumador de 4 *bits* con Flags ZCVN**. Extender el sumador de 4 *bits* creando un componente similar pero que tenga salidas que reflejen lo sucedido durante la suma binaria (*bit* a *bit*):

- el resultado es cero $\Leftrightarrow$ **Z** vale 1
- la suma binaria produjo acarreo $\Leftrightarrow$ **C** vale 1

- la suma interpretada en complemento a 2 dio overflow ⇔ **V** vale 1
- el resultado interpretado en complemento a 2 es negativo ⇔ **N** vale 1

f) **Incrementador de 4 *bits***. El mismo tendrá una entrada **A** de cuatro *bits* y una salida: **S**. La salida deberá expresar el resultado de incrementar la entrada en una unidad.

g) **Inversor de 4 *bits***. El mismo tendrá dos entradas, **A** de cuatro *bits* e **INV** y una salida **S** de cuatro *bits*. Si **INV** es 0, la salida deberá expresar el mismo valor que la entrada, mientras que si **INV** es 1, deberá expresar el **inverso aditivo** de la entrada en complemento a dos (cuando éste exista).

Checkpoint 2

h) **ALU de 4 *bits***. La misma tendrá tres entradas **A**, **B** y **OP**, ésta última de dos *bits*. Además, tendrá cinco salidas: **S** más **Z**, **C**, **V** y **N**. La salida deberá expresar el resultado de:

- A + B ⇔ **OP** vale `00`
- A - B ⇔ **OP** vale `01`. Aquí **C** deberá informar si la resta binaria produjo *borrow* (dame uno).
- A AND B (bit a bit) ⇔ **OP** vale `10`
- A OR B (bit a bit) ⇔ **OP** vale `11`

En las últimas dos operaciones, los flags **C** y **V** deberán tomar el valor `0`. En todos los casos los *flags* reflejan el resultado de operación entre operandos en complemento a 2.

i) ¿Es posible utilizar esta ALU con operandos interpretados como sin signo?¿Cómo detectaría el *overflow*?

Es posible utilizar esta ALU para operandos interpretados como sin signo, ya que el resultado en bits es el mismo. (esto es para la suma, para la resta el resultadotiene sentido si el el re

El overflow es el inverso del borrow en la resta, y el valor del carry en la suma.

5. Validación de los resultados

Completar la siguiente tabla indicando los resultados utilizando la ALU de 4 *bits*. Compare los resultados obtenidos con los calculados en las tablas del ejercicio 3.

Operandos		Sumador					Restador				
A	B	S	Z	C	V	N	S	Z	C	V	N
1111	0001	0000	1	1	0	0	1110	0	0	0	1
0001	1111										
0101	0101										
1000	0111										
0110	1010										

Para validar sus resultados, les proveemos un script. Para utilizarlo, correr el siguiente comando en una terminal: `sh verificador.sh TallerLogica-Combinatorios.circ`

Checkpoint 3