Electronica III
Trabajo Practico de Laboratorio N 1

Martin Rodriguez Turco
Juan Martin Laguinge
Tobias Scala
Guido Panaggio

5 de Septiembre de 2018

1-Rango y Resolución

0.1. Objetivo

Desarrollar un programa que calcule el Rango y Resolución de un número de punto fijo.

El mismo recibira el signo, cantidad de bits de la parte entera y la cantidad de bits de la parte fraccionaria.

0.2. Calculo de Rango y Observaciones

Para poder hacer el calculo del Rango se deben tomar en cuenta los 3 parametros mencionados. Esto se debe a que se debe distinguir si el número es de tipo signado (SIGNO = 1) o no (SIGNO = 0). Esto parece ser estrictamente necesario. Sin embargo más adelante veremos que no lo es.

Comenzamos por enunciar la definición de Rango:

Es la diferencia entre la magnitud representable más positiva y la magnitud representable más negativa

0.2.1. Ejemplo de cálculo de Rango

Introduciremos el metodo de cálculo mediante un ejemplo:

Supongamos que deseamos calcular el rango **R** de un número binario *signado* con 3 *bits* de parte entera y 2 *bits* de parte fraccionaria entonces tenemos un número con la siguiente forma:

2	1	0	-1	-2
x	x	x	x	x

Para calcular el maximo número representable M :

0	1	1	.1	1

$$0^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} = 3{,}75$$

Para calcular el minimo número representable m :

1	0	0	.0	0

$$m = -2^2 = -4$$

Por lo tanto el rango:

$$R = 3{,}75 - (-4)$$
$$R = 7{,}75$$

Ahora podemos preguntarnos Cómo realizar el cálculo sí el número es no *signado*.

La maxima denominación vendra dada por:

1	1	1	.1	1

$$M' = 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2}$$
$$M' = 7{,}75$$

La minima denominación vendra dada por:

0	0	0	.0	0

Lo cual claramente indicia que el minimo número representable es el 0. Por lo tanto:

$$m = 0$$

Entonces:

$$R' = M' - m'$$
$$R' = 7{,}75$$

Notamos que $R = R'$. Esto nos da indicios de que el rango de un número de punto fijo *signado* ó *no signado* tiene n el mismo rango. A continuación demostraremos que esto es de hecho así.

0.2.2. Formula general para el cálculo del Rango

Dado un número binario B en punto fijo con k bits en su parte entera y j bits para su parte fraccionaria. Primero trataremos a B como si se tratase de un número *no signado*.Para ambos casos separaremos el problema en dos partes. Por un lado nos encargaremos de calcular la parte entera y luego la fraccionaria.

Parte entera:

$$P_k = 1 + 2 + \cdots + 2^k \tag{1}$$

$$2P_k = 2 + 2^2 + \cdots + 2^k + 2^{k+1} \tag{2}$$

Restandolas:

$$2P_k - P_k = 2^{k+1} - 1 \tag{3}$$

$$P_k = 2^{k+1} - 1 \tag{4}$$

Para realizar el calculo de la parte fraccionaria procedemos de forma analoga:

$$F_j = \frac{1}{2} + \frac{1}{4} + \cdots + \left(\frac{1}{2}\right)^j \tag{5}$$

$$\frac{1}{2}F_j = \frac{1}{4} + \frac{1}{8} + \cdots + \left(\frac{1}{2}\right)^j + \left(\frac{1}{2}\right)^{j+1} \tag{6}$$

Restandolas:

$$F_j - \frac{1}{2}F_j = \frac{1}{2} - \left(\frac{1}{2}\right)^{j+1} \tag{7}$$

De lo cual obtenemos

$$F_j = 1 - \left(\frac{1}{2}\right)^j \tag{8}$$

Luego, como estamos tratando a B como un número *no signado*. Usando 4 y 7 :

$$R = P_k + F_j = \left(2^{k+1} - 1\right) + \left(1 - \left(\frac{1}{2}\right)^j\right) - 0$$

Obtenemos la formula para el Rango de un número *no signado*:

$$R = 2^{k+1} - \left(\frac{1}{2}\right)^j$$

Para el caso de un número *signado* se tienen en cuenta $k + j - 1$ bits para calcular el número más positivo representable. Es decir:

$$P'_k = P_{k-1} = 2^{(k+1)-1)} - 1$$

$$P'_k = 2^k - 1$$

En el caso de la parte fraccionaria es analogo al primero tratado

$$F'_j = F_j F'_j = 1 - \left(\frac{1}{2}\right)^j$$

Para obtener la minima representación posible:

$$m''_k = -2^k \tag{9}$$

Por lo tanto ya tenemos todo lo necesario para calcular R' el rango de un número signado:

$$\begin{aligned} R' &= P'_k + F'_j - m''_k \\ R' &= \left(2^k - 1\right) + \left(1 - \left(\frac{1}{2}\right)^j\right) - \left(-2^k\right) \end{aligned}$$

Reordenando y agrupando convenientemente

$$R' = 2^{k+1} - \left(\frac{1}{2}\right)^j$$

Finalmente hemos demostrado que el Rango de un número en punto fijo no depende de si el número es *signado* o *no signado* Esto presenta una gran ventaja a nivel computacional ya que no son necesarias consideraciones extra y simplifica el cálculo a una sola expresión

0.2.3. Formula general para el cálculo de la Resolución

Un número en punto fijo permite expresar números reales con cierto grado de precisión. Esta caracteristica esta limitada por la cantidad de bits que se dediquen a la parte fraccionaria.

La resolución vendra dada por:

$$Resolución = \left(\frac{1}{2}\right)^j \tag{10}$$

Y es indistinto si el número es *signado* o es *no signado*

2-Mapas de Karnaugh y Algebra de Boole

0.3. Objetivo

Obtener la representación en mapas de Karnaugh de una función dada. Esta misma debera ser simplifacada aplicando las tecnicas de agrupación vistas en clase. Además se debera realizar mediante el método tradicional del Algebra de Boole.

Finalmente modelaremos el sistema resultante con su correspondiente circuito logico. Luego, dado se expresara este mismo haciendo uso unicamente de compuertas *nand*.

0.3.1. Obtención del mapa de Karnaugh

Dada la siguiente expresión en maxtérminos se pudo interpretar el mapa de Karnaugh de donde provinieron

$$f(a,b,c,d) = \prod (M_0, M_1, M_5, M_7, M_8, M_{10}, M_{14}, M_{15})$$

ab \ cd	00	01	11	10
00	0	0	1	1
01	1	0	0	1
11	1	1	0	0
10	0	1	1	0

Mediante la selección de dichos grupos se obtuvo la siguiente función logica simplificada mediante mintérminos:

$$f(a,b,c,d) = b\overline{cd} + a\overline{c}d + \overline{b}cd + \overline{a}c\overline{d}$$

Dado que se han escogido 4 grupos se ha llegado de forma inmediata a dicha formulación

0.3.2. Aplicación del algebra de Boole

En este apartado desarrollaremos paso por paso las operaciones necesarias para obtener la expresión logica simplificada del sistema dado. Nuestro primer paso es plantear la suma de de los mintérminos:

$$f(a,b,c,d) = \underbrace{\overline{a}\overline{b}cd}_{m_3} + \underbrace{\overline{a}\overline{b}c\overline{d}}_{m_2} + \underbrace{\overline{a}b\overline{c}\overline{d}}_{m_4} + \underbrace{\overline{a}bc\overline{d}}_{m_6} + \underbrace{a\overline{b}\overline{c}\overline{d}}_{m_{12}} + \underbrace{a\overline{b}\overline{c}d}_{m_{13}} + \underbrace{a\overline{b}c\overline{d}}_{m_9} + \underbrace{a\overline{b}cd}_{m_{11}}$$

El resultado de su simplificación dependera de cómo se sumen los terminos. Primero aplicaremos ciegamente las reglas de simplificación:

$$f(a,b,c,d) = \underbrace{\overline{a}\overline{b}c\underbrace{\left(d+\overline{d}\right)}_{1}}_{m_3+m_2} + \underbrace{\overline{a}b\overline{d}\underbrace{(c+\overline{c})}_{1}}_{m_4+m_6} + \underbrace{ab\overline{c}\underbrace{\left(d+\overline{d}\right)}_{1}}_{m_{12}+m_{13}} + \underbrace{a\overline{b}d\underbrace{(c+\overline{c})}_{1}}_{m_9+m_{11}}$$

La expresión simplificada:

$$f(a,b,c,d) = \overline{a}\overline{b}c + \overline{a}b\overline{d} + ab\overline{c} + a\overline{b}d$$

Esta expresión tiene la misma cantidad de terminos que la obtenida mediane el mapa de Karnaugh. Ahora se realizara de nuevo la suma pero esta vez se agruparan tal como fueron escogidos los grupos en el mapa.

$$f(a,b,c,d) = \underbrace{\overline{a}\overline{b}cd + a\overline{b}cd}_{m_3+m_{11}} + \underbrace{\overline{a}b\overline{c}\overline{d} + ab\overline{c}\overline{d}}_{m_4+m_{12}} + \underbrace{a\overline{b}\overline{c}d + a\overline{b}c\overline{d}}_{m_{13}+m_9} + \underbrace{\overline{a}\overline{b}c\overline{d} + \overline{a}bc\overline{d}}_{m_2+m_6}$$

Simplificando:

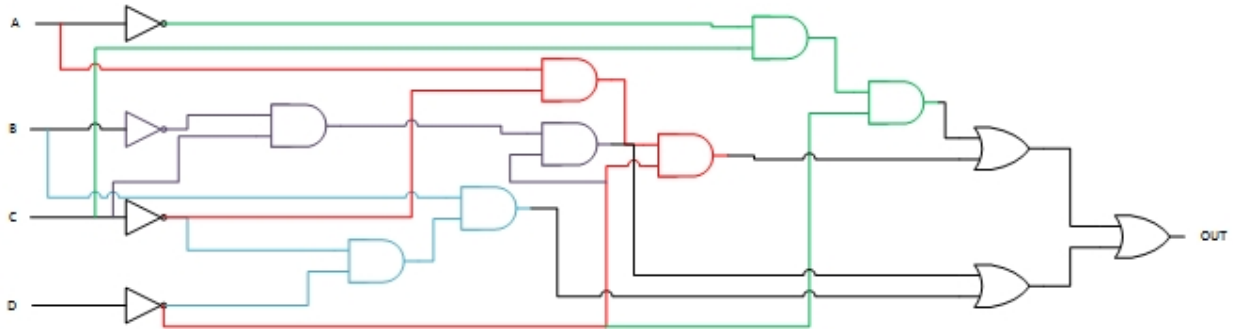
$$f(a,b,c,d) = \underbrace{\overline{b}cd\underbrace{(\overline{a}+a)}_{1}}_{m_3+m_{11}} + \underbrace{b\overline{c}\overline{d}\underbrace{(\overline{a}+a)}_{1}}_{m_4+m_{12}} + \underbrace{a\overline{c}d\underbrace{(\overline{c}+c)}_{1}}_{m_{13}+m_9} + \underbrace{\overline{a}b\overline{d}\underbrace{\left(\overline{b}+b\right)}_{1}}_{m_2+m_6}$$

Finalmente, reacomodando los terminos:

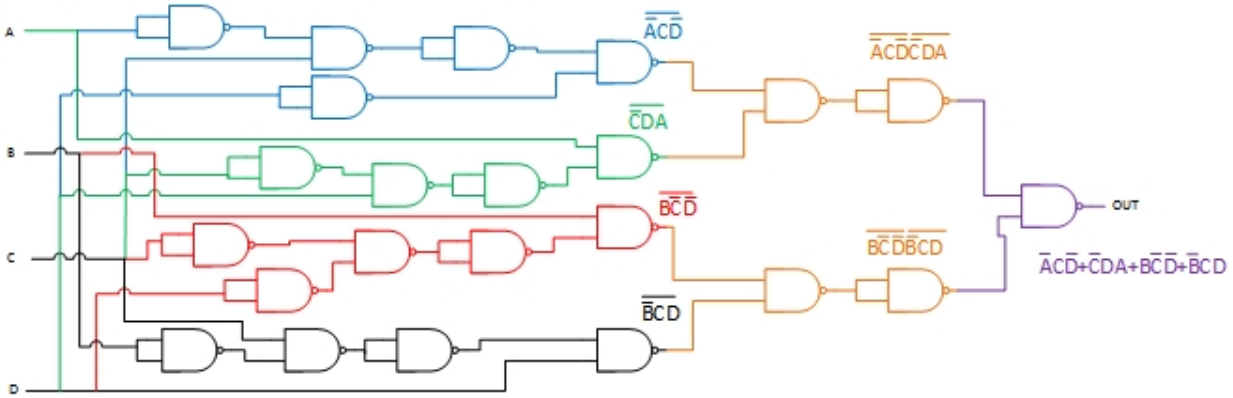
$$f(a,b,c,d) = b\overline{c}\overline{d} + a\overline{c}d + \overline{b}cd + \overline{a}c\overline{d}$$

Llegamos nuevamente a la expresión deducida mediane el mapa de Karnaugh

0.4. Circuito Logico Utilizando compuertas AND,OR y NOT



0.5. Circuito Logico Utilizando compuertas NAND



3-Encoder y Demux

0.6. Encoder

0.6.1. Objetivo

Desarrollar un módulo en Verilog el cual funcione como un encoder de 4 entradas.

0.6.2. Explicación y Desarrollo

Un encoder tiene la funcionalidad de tomar los datos de entrada y los codifica para que las salidas representen (en número en binario) la entrada cuyo valor es 1 como se puede ver en la siguiente tabla de verdad.

Cuadro 1: Tabla de Verdad del Encoder

INPUT	**OUTPUT**
a b c d	**x y**
0 0 0 0	z z
0 0 0 1	0 0
0 0 1 0	0 1
0 0 1 1	z z
0 1 0 0	1 0
0 1 0 1	z z
0 1 1 0	z z
0 1 1 1	z z
1 0 0 0	1 1
1 0 0 1	z z
1 0 1 0	z z
1 0 1 1	z z
1 1 0 0	z z
1 1 0 1	z z
1 1 1 0	z z
1 1 1 1	z z

Como las únicas salidas válidas del encoder son aquellas cuando hay una única entrada que contiene un 1, las demás combinaciones de entradas se consideran como comodines para las salidas. Las cuales se puede elegir su valor de tal forma de facilitar los cálculos para obtener la función lógica. Para poder obtener la fución lógica del mismo, se hizo uso del mapa de Karnaugh el cual se muestra a continuación.

ab \ cd	00	01	11	10
00	z	1	z	1
01	0	z	z	z
11	z	z	z	z
10	0	z	z	z

ab \ cd	00	01	11	10
00	z	0	z	1
01	0	z	z	z
11	z	z	z	z
10	1	z	z	z

El mapa de Karnaugh de la izquierda corresponde la salida X y el de la derecha la salida Y. Empezando con el mapa correspondiente a la salida X, se eligieron los valores de los comodines (z) de forma conveniente para poder agrupar lo más posible. Este mapa se lo resolverá utilizando mintérminos.

ab \ cd	00	01	11	10
00	1	1	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

Ahora, para poder armar la función lógica, se debe ver qué variable (dentro de cada grupo) no cambia su valor. Y si dicha variable contiene el valor 0, entonces se lo niega ya que estamos trabajando con mintérminos. Se obtiene la siguiente función lógica para la salida X.

$$X = \text{Not(c)Not(d)} + a$$

Cuya tabla de verdad se muestra a continuación.

Ahora centrándose en el mapa correspondiente a la salida Y, se eligieron nuevamente los valores de los comodines (z) de forma conveniente para poder agrupar lo más posible. Este mapa se lo resolverá utilizando maxtérminos.

Cuadro 2: Tabla de Verdad del Encoder

INPUT	**OUTPUT**
a c d	**x**
0 0 0	1
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

ab \ cd	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	0	0
10	1	1	0	0

Para poder armar la función lógica, se debe ver nuevamente qué variable (dentro de cada grupo) no cambia su valor. Y si dicha variable contiene el valor 1, entonces se lo niega ya que estamos trabajando con maxtérminos. Se obtiene la siguiente función lógica para la salida Y.

$$Y = a\bar{c} + c\bar{a}$$

Cuya tabla de verdad se muestra a continuación.

Se puede notar, a través de la tabla de verdad, que

$$Y = xor(a, c)$$

0.7. Demux

0.7.1. Objectivo

Desarrollaar un módulo en Verilog el cual implemente el concepto de Demux.

Cuadro 3: Tabla de Verdad del Encoder

INPUT	**OUTPUT**
a c	**y**
0 0	0
0 1	1
1 0	1
1 1	0

0.7.2. Criterio e implementación

Dado que un componenete del tipo Demux tiene por finalidad dirigir su entrada hacia una salida en particular se realizo la siguiente observación:

Si la entrada es nula su salida sera nula en todas las terminales

Teniendo dicho conocimiento podemos evitar realizar cualquier tipo de calculo dado que al conocer la entrada la salida queda completamente determinada. No es así en el caso de que la entrada sea 1 dado que cualquiera de los 4 terminales de salida puede ser esocgido para reflejar el valor de entrada. Para su implementación en Verilog se decidio implementar mediante shifteos de 1 bit. Es decir que para evitar definir la solución de manera anticipada (dado que desconocemos las consecuencias de su implementación en la vida real). En otras palabra, definir una constante puede incurrir en un gasto de energía constante mientras que la realización de un calculo cuando es necesario puede ser beneficioso.

Ejercicio 4

Para lograr ver los mintérminos que poseen los bits de salida realizamos la siguinte tabla de verdad:

Cuadro 4: Tabla de Verdad

INPUT	**OUTPUT**
0 0 0 0	0 0 0 0
0 0 0 1	1 1 1 1
0 0 1 0	1 1 1 0
0 0 1 1	1 1 0 1
0 1 0 0	1 1 0 0
0 1 0 1	1 0 1 1
0 1 1 0	1 0 1 0
0 1 1 1	1 0 0 1
1 0 0 0	1 0 0 0
1 0 0 1	0 1 1 1
1 0 1 0	0 1 1 0
1 0 1 1	0 1 0 1
1 1 0 0	0 1 0 0
1 1 0 1	0 0 1 1
1 1 1 0	0 0 1 0
1 1 1 1	0 0 0 1

Siendo X el primer bit de la izquierda,Y el segundo, W el tercero y Z el ultimo del OUTPUT; y siendo A el primer bit de la izquierda,B el segundo,C el tercero y D el ultimo. Obtenemos las siguentes salidas dadas en funcion de los minterminos:

$$X = m1 + m2 + m3 + m4 + m5 + m6 + m7 + m8$$

$$Y = m1 + m2 + m3 + m4 + m9 + m10 + m11 + m12$$

$$W = m1 + m2 + m5 + m6 + m9 + m10 + m13 + m14$$

$$Z = m1 + m3 + m5 + m7 + m9 + m11 + m13 + m15$$

Reemplazando los minterminos mi por sus respectivos valores nos queda la siguiente repuesta:

$$X = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D+\overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}+\overline{A}\cdot\overline{B}\cdot C\cdot D+\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}+\overline{A}\cdot B\cdot\overline{C}\cdot D+\overline{A}\cdot B\cdot C\cdot\overline{D}+\overline{A}\cdot B\cdot C\cdot D+A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}$$

$$Y = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D+\overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}+\overline{A}\cdot\overline{B}\cdot C\cdot D+\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}+A\cdot\overline{B}\cdot\overline{C}\cdot D+A\cdot\overline{B}\cdot C\cdot\overline{D}+A\cdot\overline{B}\cdot C\cdot D+A\cdot B\cdot\overline{C}\cdot\overline{D}$$

$$W = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D+\overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}+\overline{A}\cdot B\cdot\overline{C}\cdot D+\overline{A}\cdot B\cdot C\cdot\overline{D}+A\cdot\overline{B}\cdot\overline{C}\cdot D+A\cdot\overline{B}\cdot C\cdot\overline{D}+A\cdot B\cdot\overline{C}\cdot D+A\cdot B\cdot C\cdot\overline{D}$$

$$Z = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D+\overline{A}\cdot\overline{B}\cdot C\cdot D+\overline{A}\cdot B\cdot\overline{C}\cdot D+\overline{A}\cdot B\cdot C\cdot D+A\cdot\overline{B}\cdot\overline{C}\cdot D+A\cdot\overline{B}\cdot C\cdot\overline{D}+A\cdot B\cdot\overline{C}\cdot D+A\cdot B\cdot C\cdot D$$

Procedemos a hacer las tablas de Karnaugh de todas las salidas:

Cuadro 5: Tabla de Karnaugh para X

C D \ A B	**0 0**	**0 1**	**1 1**	**1 0**
0 0	0	1	0	1
0 1	1	1	0	0
1 1	1	1	0	0
1 0	1	1	0	0

Cuadro 6: Tabla de Karnaugh para Y

C D \ A B	0 0	0 1	1 1	1 0
0 0	0	1	1	0
0 1	1	0	0	1
1 1	1	0	0	1
1 0	1	0	0	1

Cuadro 7: Tabla de Karnaugh para W

C D \ A B	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	1	1	1	1
1 1	0	0	0	0
1 0	1	1	1	1

Cuadro 8: Tabla de Karnaugh para Z

C D \ A B	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	1	1	1	1
1 1	1	1	1	1
1 0	0	0	0	0

Luego de seleccionar los grupos correspondientes, las simplificaciones nos quedan:

$$X = A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B + \overline{A} \cdot C + \overline{A} \cdot D$$

$$Y = B \cdot \overline{C} \cdot \overline{D} + \overline{B} \cdot C + \overline{B} \cdot D$$

$$W = \overline{C} \cdot D + C \cdot \overline{D}$$

$$Z = D$$

De las simplificaciones obtenemos el siguente circuito logico que fue probado y armado en https://logic.ly/demo/.

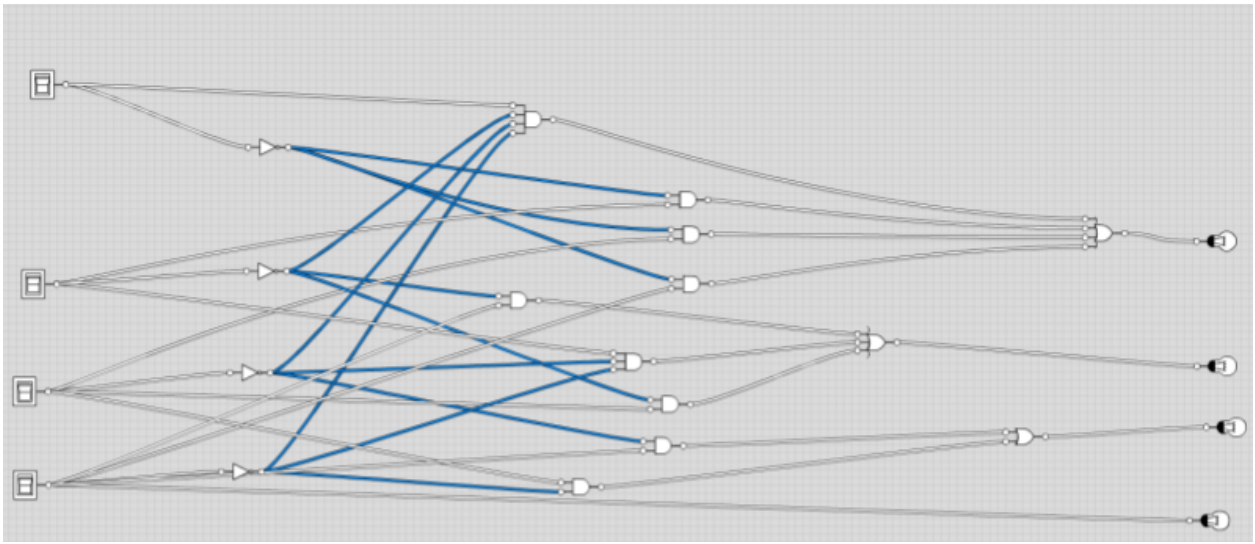


Figura 1: Circuito logico

5 - Sumador BCD

0.8. Objetivo

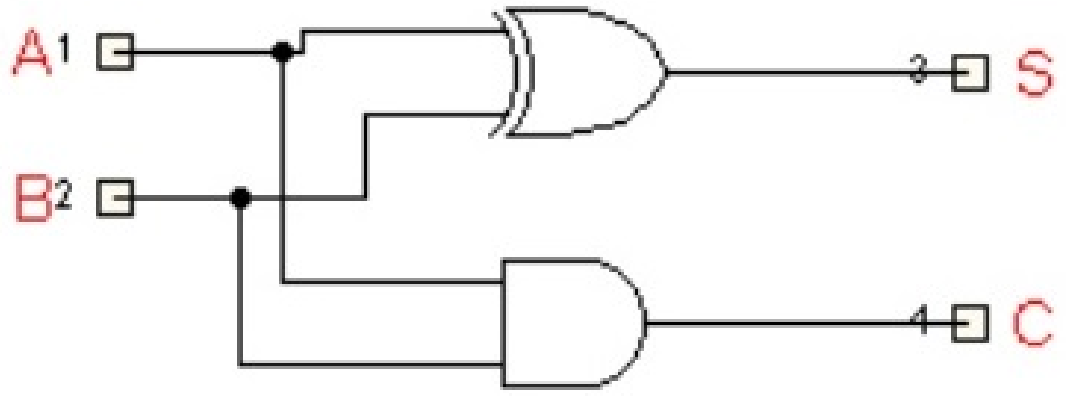
Un sumador BCD posee una función bastante autoexplicativa, esta es la de tomar 2 valores en expresión binaria y presentar la suma de esta en un formato BCD.

0.9. Implementaciíon

A la hora de desarrollar el programa se han tenido en cuenta los conceptos de modularización y reutilización de codigo por lo que este ha comprendido distintas etapas con sus propias funcionalidades. Por lo tanto la implementación final del codigo ha comprendido distitas estapas las cuales se aboradaran a continuación.

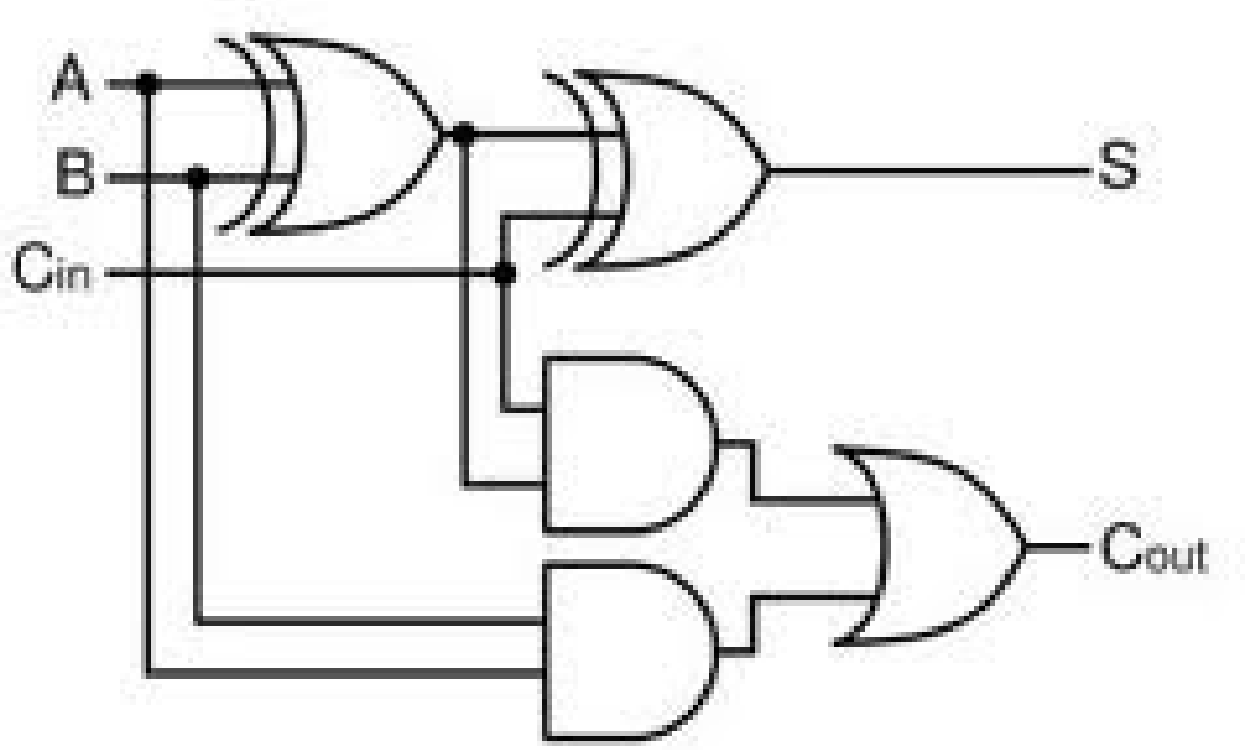
0.9.1. Adder de 1bit

El primer circuito lógico necesario para todas las siguientes implementaciones fue sumador de 1 bit, este tiene la función de realizar la suma de 2 valores de un bit arrojando como parametros de salida tanto el bit resultante como el bit *carry* generado por estos. este circuito se logra realizando una función AND a los parametros de entrada, llamese estos A y B, obteniendo de ese modo el bit de salida Y y paralelamente realizando una funcion XOR a los mismos y definiendo de ese modo el valor carry de salida. Este circuito se puede ver representado en el siguiente diagrama:



0.9.2. Full-Adder de 1bit

Una vez obtenido el Adder se implementa este para lograr el Full-Adder de un bit el cual cumple las mismas funciones basicas con la salvedad que este es capaz de recibir ademas un bit carry como parametro de entrada. Las funciones que cumple el Adder de un bit en el funcionamiento del Full Adder son las siguientes, en primer lugar se tratan los bit de entrara A y B, estos son aplicados al Adder de ese modo obteniendo un bit de carry que llamaremos *Co1* y una salida *Y1*, lo siguiente sera realizar un Adder con los valores Y1 y el carry de entrada *Cin*, una vez en este punto ya tendremos el bit de salida deasado junto con un segundo bit carry de salida, llamese *Co2*, por lo que resta definir cual sera el bit de carry a la salida del circuito. Este ultimo se obtiene realizando una función OR a los 2 bits carry *Co1* y *Co2*. A continuación se muestra el diagrama del Full-Adder de 1 bit: adder.jpg ad-



der.jpg

0.9.3. Full-Adder de 4 bits

Contando ya con el Full Adder de 1 bit se escala el funcionamiento de este para lograr realizar la función de suma para valores binarios de 4 bits. Para ello el procedimiento que se realiza es el de tomar de a pares los bits que componen los nibbles con los que se trabaja comenzando por los menos significativos, estos son tratados con el Full Adder (siendo el bit carry de entrada 0) y de aqui se obtiene el bit menos significativo del nibble de salida, por otro lado el bit carry que se obtinene sera el mismo ingresado como el de entrada al Full Adder al que se ingresaran los siguientes bits de los nibbles de entrada. Asi se obtendran los bits de salida yendo del menos significativo al mas significativo y, a su vez, el bit carry de salida producto del ultimo Full Adder de 1 bit implementado sera el que determine si se produjo carry al finalizar la suma.

0.9.4. Diseño del Sumador BCD

Una vez contando con todas estas herramientas nos disponemos a desarrollar el sumador BCD, para esto el procedimiento a seguir es el siguiente. El suamdor recibira como parametros de entrada 2 nibbles que se llamaran A y B estos seran ingresados a un bloque Full-Adder de 4 bits, el cual como se sabe devolvera el nibble de salida junto con el bit carry generado, con estos datos se veran si se cumplen alguna de las siguientes condiciones: por un lado si el nibble de salida comprende

un numero decimal mayor a 9, o si se ha producido carry en la suma, esto determinara el valor de una señal que actuara como selector de un dispositivo MUX, el cual dependiendo del estado del selector arrojara a la salida un valor decimal de 6, el cual sera el factor de corrección que convierta el valor en expresión binaria a BCD, o un valor nulo. Por último se tomara el nibble de salida del bloque Adder anterior y el valor a la salida del MUX y se haran pasar estos por un nuevo bloque Full-Adder de 4 bits el cual devolvera un nuevo nibble de salida y el carry de la suma. El criterio que se siguio fue el de devolver el nibble menos significtivo de la suma y un bit carry que indique si el mas significativo comprende un 1 o 0, debido a que la suma de 2 números dara un valor por debajo de 20. Se aclara ademas que la operación se suma de hizo robusta de tal menera que también informara si los parametros de entrada no son validos

6-ALU

0.10. Objetivo

Implementar una ALU que cumpla con las operaciones basicas *SUMA, RESTA, AND, OR, NOT, XOR, CA2, SHFT2*.

0.10.1. Criterio e implementación

Siguiendo las recomendaciones de la comunidad de Verilog la cual recomienda escribir el codigo de la manera más explicita posible para el compilador. La ALU tiene como entradas un selector de operaciones de $3bits$ y dos registros A y B de $4bits$ cada uno. Se implementaron las operaciones haciendo uso de las funciones ya aportadas por el lenguaje. Para la selección de operacion se hizo uso de un bloque switch que junto con el bloque @always refresca el estado del programa cada vez que ocurre un cambio en el selector de acción. Se implemento una CLI en Python con el fin de aumentar la interacción con la misma sin sacrificar su funcionalidad. Esto ha sido realizado con fines experimentales. Notese que esta interfaz no se entromete con el codigo fuente original de Verilog, simplemente lo utiliza.