Parcial

1)

a.

El número más grande sería -8 = “1000” (el rango va de -8 a 7), cuyo valor al cuadrado es: 64 = “0100_000”. Por lo que la salida, considerando el bit de signo, tendrá que ser de 8 bits.

b.

A3	A2	A1	A0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	1	1	0	0	1
0	1	1	0	0	0	1	0	0	1	0	0
0	1	1	1	0	0	1	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	1	1	0	0	0	1
1	0	1	0	0	0	1	0	0	1	0	0
1	0	1	1	0	0	0	1	1	0	0	1
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	0	0	1	0	0	1
1	1	1	0	0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0	0	0	0	1

c.

Minimización:

$Y_7 = 0$

$Y_6 = A_3 . /A_2 . /A_1 . /A_0$

Y5					
		A1 A0			
		00	01	11	10
A3A2	00	0	0	0	0
	01	0	0	1	1
	11	0	0	0	0
	10	0	1	0	1

$$Y_5 = /A_3 . A_2 . A_1 + A_3 . /A_2 . /A_1 . A_0 + A_3 . /A_2 . A_1 . A_0$$

Y4					
		A1A0			
		00	01	11	10
A3A2	00	0	0	0	0
	01	1	1	1	0
	11	0	0	0	0
	10	0	1	1	1

Rojo + Naranja + Amarillo + Verde

$$Y_4 = A_3 . /A_2 . A_0 + /A_3 . A_2 . A_0 + A_3 . /A_2 . A_1 + /A_3 . A_2 . /A_1$$

Y3					
		A1 A0			
		00	01	11	10
A3 A2	00	0	0	1	0
	01	0	1	0	0
	11	0	1	0	0
	10	0	0	1	0

$$Y_3 = /A_2 . A_1 . A_0 + A_2 . A_1 . A_0$$

Y2					
		A1 A0			
		00	01	11	10
A3 A2	00	0	0	0	1
	01	0	0	0	1
	11	0	0	0	1
	10	0	0	0	1

$Y_2 = A_1 . A_0$

$Y_1 = 0$

Y0					
		A1 A0			
		00	01	11	10
A3 A2	00	0	1	1	0
	01	0	1	1	0
	11	0	1	1	0
	10	0	1	1	0

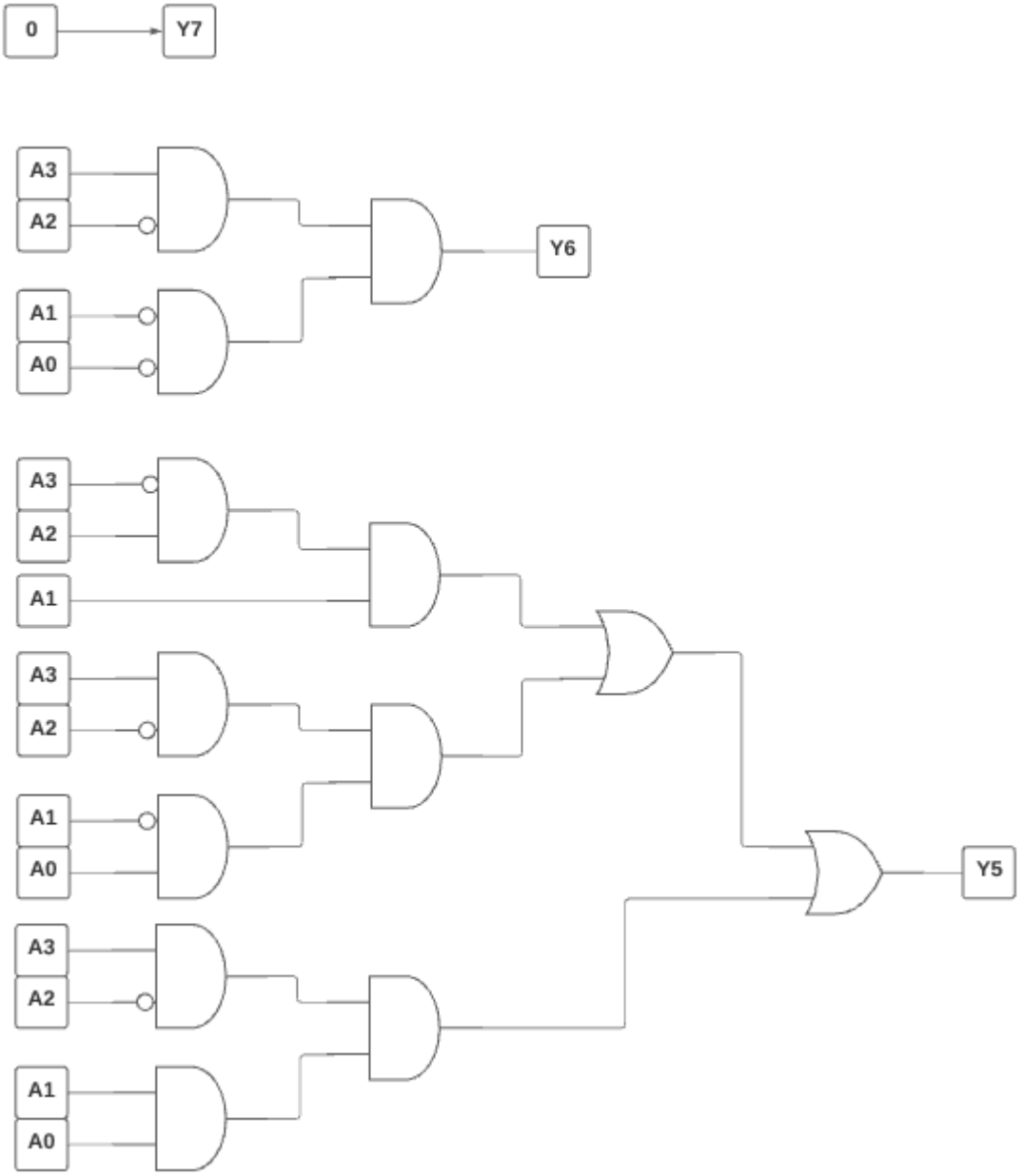
$Y_0 = A_0$

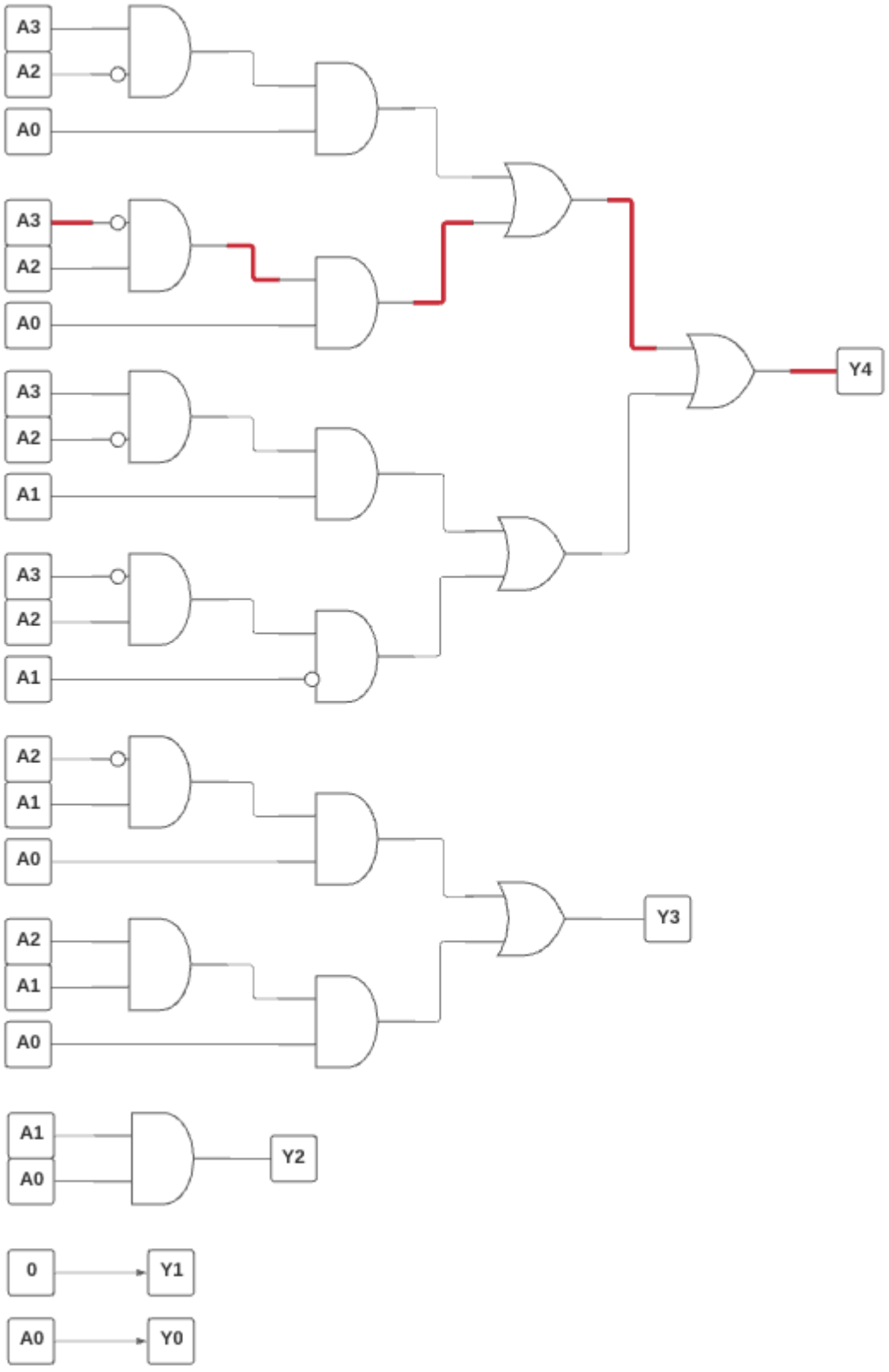
Según como lo veo, ambas operaciones son iguales:

$W = A.B.C.D = (A.B).(C.D) = ((A.B).C).D$

Sin embargo, el primer caso requiere hacer dos operaciones AND, y luego hacer AND en el resultado (2 propagaciones); mientras que en el segundo caso tengo que hacer una AND entre A y B, luego a ese resultado una AND con C, y luego a ese resultado una AND con D (3 propagaciones).

Para mi circuito, voy a usar el esquema de 2 propagaciones.


0
Y7
A3
A2
A1
A0
Y6
A3
A2
A1
A3
A2
A1
A0
A3
A2
A1
A0
Y5



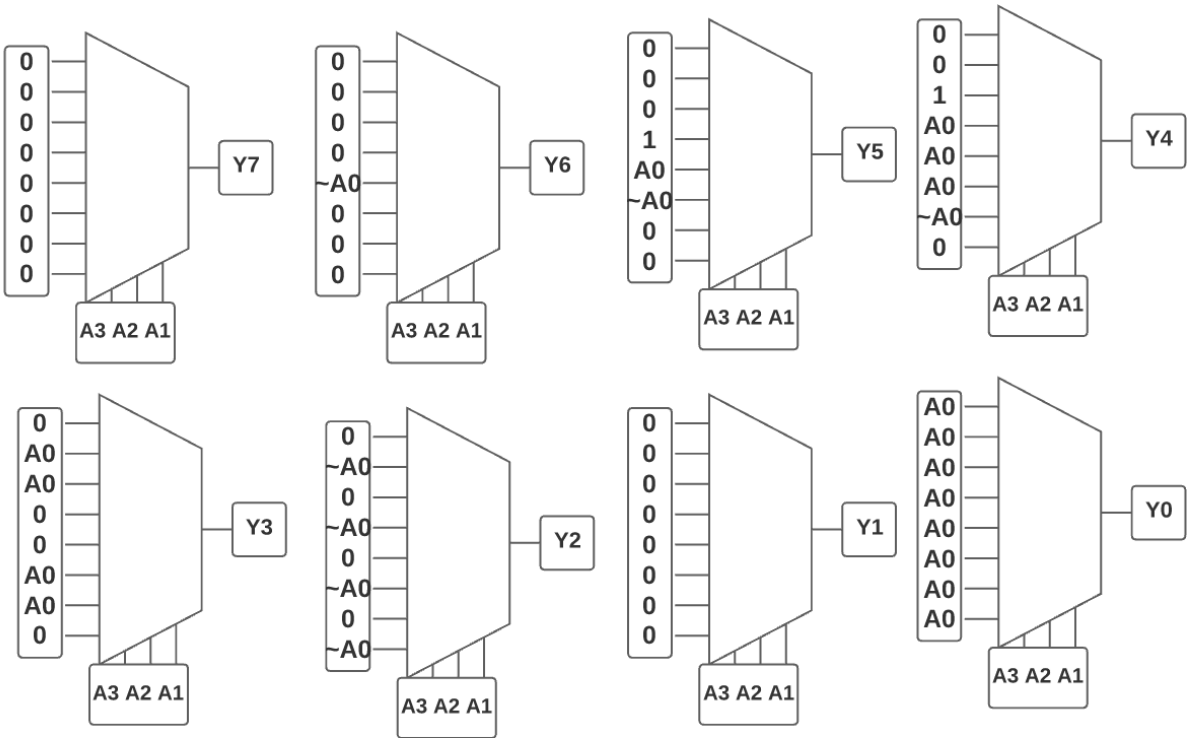
d.

Se marco el camino crítico con rojo. Hay muchos caminos que utilizan la misma cantidad de compuertas, ese es solo uno.

$$Demora\ máxima\ = t_g\ *\ (inversor\ +\ AND\ +\ AND\ +\ OR\ +\ OR)$$
$$Demora\ máxima\ =\ 0.5ns\ *\ (5)\ =\ 2.5ns$$

e.

A3	A2	A1	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	A0
0	0	1	0	0	0	0	A0	~A0	0	A0
0	1	0	0	0	0	1	A0	0	0	A0
0	1	1	0	0	1	A0	0	~A0	0	A0
1	0	0	0	~A0	A0	A0	0	0	0	A0
1	0	1	0	0	~A0	A0	A0	~A0	0	A0
1	1	0	0	0	0	~A0	A0	0	0	A0
1	1	1	0	0	0	0	0	~A0	0	A0



2)

Primero niego el MSB de los 3 números para comparar las magnitudes (podría usar comparadores signados, pero ya que estamos vamos a hacer las cosas completas).

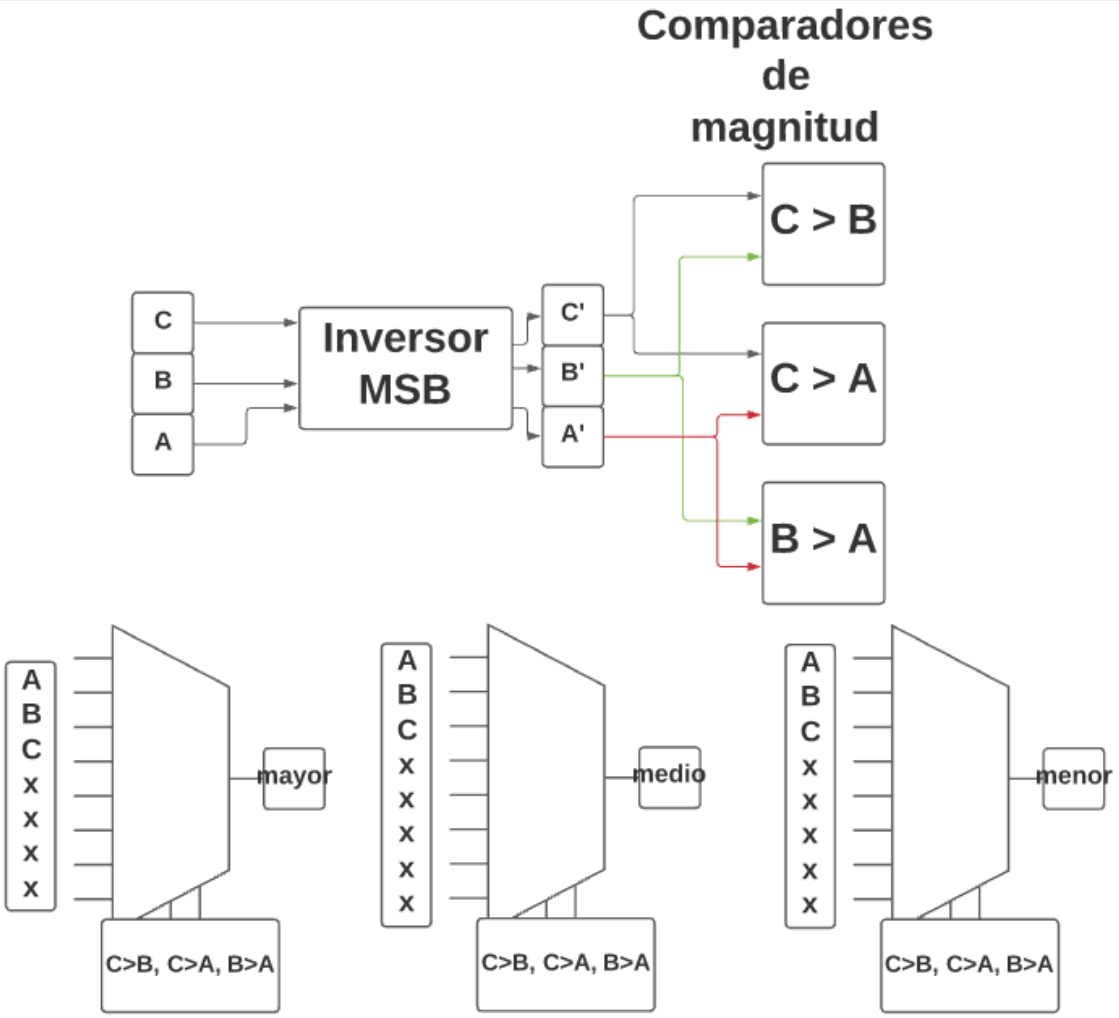
Luego hay solamente tres relaciones que me interesan:
C > B C > A B > A

Las relaciones entre C y B, C y A, B y A. El resto lo puedo obtener negando los resultados (por ejemplo C < B = ~(C > B)).

Pudiendo construir una tabla de la verdad del siguiente estilo:
(Nota, los casos no están hechos correctamente, es a modo de ejemplo).

Esta tabla puede ser implementada con 3*N multiplexores de 3 vías de control, uno para cada bit de cada salida.

C>B	C>A	B>A	Mayor	Menor	Medio
0	0	0	A	B	C
0	0	1	B	C	A
0	1	0	A	B	C
0	1	1	A	B	C
1	0	0	C	B	A
1	0	1	B	A	C
1	1	0	A	B	C
1	1	1	C	B	A


Comparadores de magnitud
C
B
A
Inversor MSB
C'
B'
A'
C > B
C > A
B > A
A
B
C
x
x
x
x
x
mayor
C>B, C>A, B>A
A
B
C
x
x
x
x
x
medio
C>B, C>A, B>A
A
B
C
x
x
x
x
x
menor
C>B, C>A, B>A

3)

El overflow funciona así. Suponga que yo tengo dos números de N bits signados.
El MSB indica el signo de cada número. Al sumar dos números de N bits, la suma puede dar **hasta** N+1 bits (ejemplo: 0110 + 0010 = 01000). Sin embargo, si queremos que la suma nos devuelva un número de N bits, igual que los sumandos, tomando el ejemplo anterior: 0110 + 0010 = 0 1000. El resultado sería “1000”, que es el -8, un número negativo. (6 + 2 = -8!!).

La suma, si el resultado no es de N+1 bits, es como una rueda, donde, para 4 bits, el número más positivo es el “+7” y el siguiente a ese es el “-8”, el más negativo.

En resumen, podemos identificar que hubo overflow si:

- Al sumar dos números positivos (MSB=0), el resultado es negativo (MSB=1)
- Al sumar dos números negativos (MSB = 1), el resultado es positivo (MSB = 0).

$$OVS = (not\ A(N-1)\ and\ not\ B(N-1)\ and\ S(N-1))\ or$$
$$(A(N-1)\ and\ B(N-1)\ and\ not\ S(N-1()$$

Suponiendo que A(N-1) es el MSB, y que S es el resultado de la suma, en N bits.

El carry es “el complemento” de la suma actual, traído de la suma anterior.
Por ejemplo en decimal. La suma 19 + 23 se realiza así:

Unidades: 3 + 9 = 1 2 (le paso “1” de carry a las decenas)
Decenas: 1 + 2 + 1(el carry anterior) = 4
19 + 23 = 42

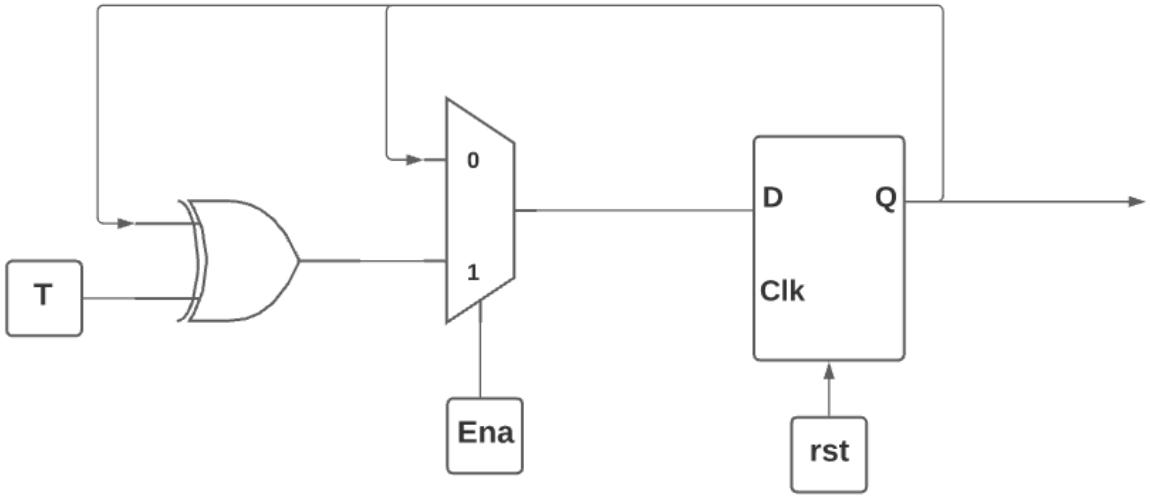
Este mismo concepto se puede llevar a cualquier base numérica, no sólo en base 10.
Trabajando en base 2 (binario, magnitudes): (se indicará con una “S” el resultado de la suma actual, con “Co” el carriage output, y con “Ci” el carriage input.

```
        0       1       0       1
+       0       1       1       1
                ----------------------
                        1(Co)   0(S)

                        1(Ci)
        0       1       0       1
+       0       1       1       1
                ----------------------
                1(Co)   0(S)    0

                1(Ci)
        0       1       0       1
+       0       1       1       1
                ----------------------
```

1(Co) 1(S) 0 0 ⇒ Resultado final = 1100

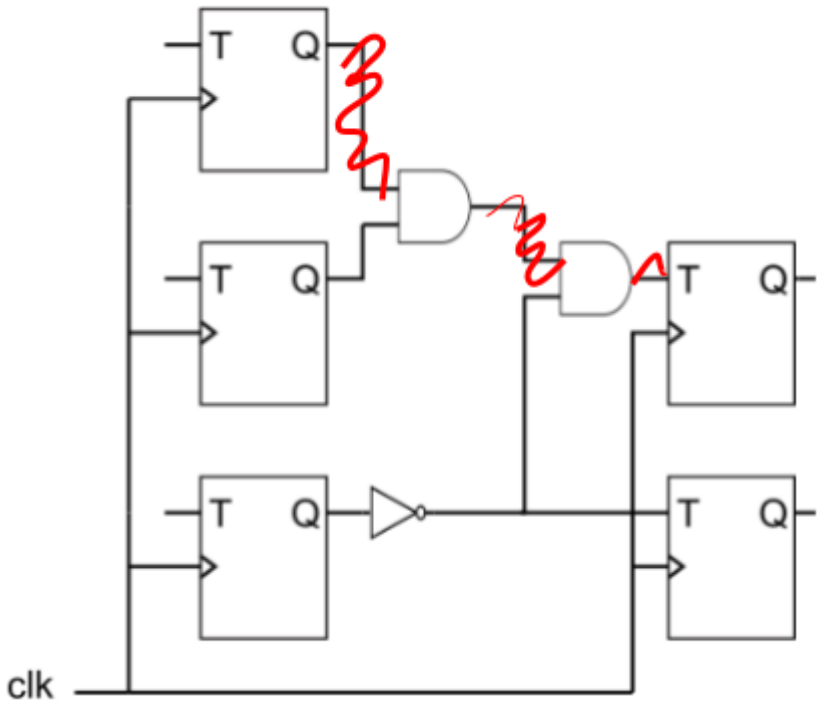
4)

Perdón, pero me marea lo de “activo en alto”. Yo lo considero así, cualquier cosa sería negar el valor de “rst” y de “ena”.



5)

En rojo se marca el camino crítico (uno de varios), siendo aquel que tiene 2 compuertas con sus tiempos de gate entre un flip-flop y el siguiente.



a.

Que el circuito funcione correctamente significa que ambos slacks son mayores a 0.
$Si\ F = 150Mhz \implies T = 6.67ns$

$$t_{slack-s} = T_{min} - (t_{cqA-Max} + t_{g-Max} + t_{sB-Max} - t_{sk-Min}) > 0$$

$$t_{slack-s} = 6.67 - (3 + 2 * 1.5 + 1)$$

$t_{slack-s} = -0.33\ ns$ El slack de setup es negativo.

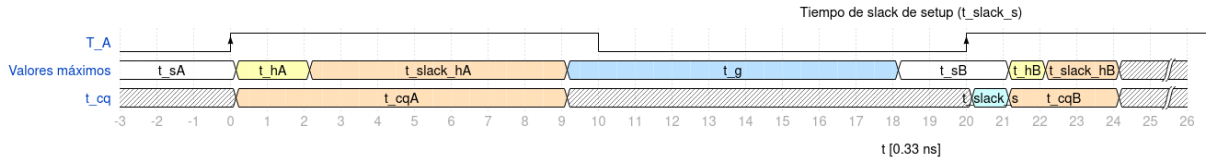
$$t_{slack-hAB} = t_{cqA-Min} + t_{g-Min} - t_{hB-Max} - t_{sk-Max} > 0$$

$$t_{slack-hAB} = 2 + 2 * 1.5 - 0.5$$

$t_{slack-hAB} = 4.5ns$ El slack de hold es positivo.

Para el slack de setup, vemos en el diagrama temporal como el tiempo de setup de B no llega a “entrar” dentro del ciclo de clock, luego del tiempo de gate, resultando en que le “faltan” 0.33 ns” para llegar al tiempo de setup.
Importante: la escala del gráfico es de “0.33 ns” por tick.

[Código de WaveDrom]



b.

Para $F = 100Mhz \implies T = 10ns$

$$t_{slack-s} = T_{min} - (t_{cqA-Max} + t_{g-Max} + t_{sB-Max} - t_{sk-Min}) > 0$$

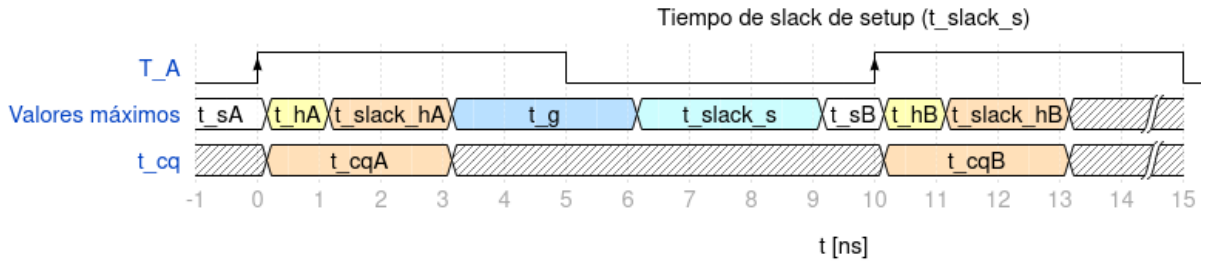
$t_{slack-s} = 10 - (7) = 3ns$ Es positivo

El slack de hold no depende de la frecuencia, así que su valor sigue siendo positivo

$t_{slack-hAB} = 4.5ns$

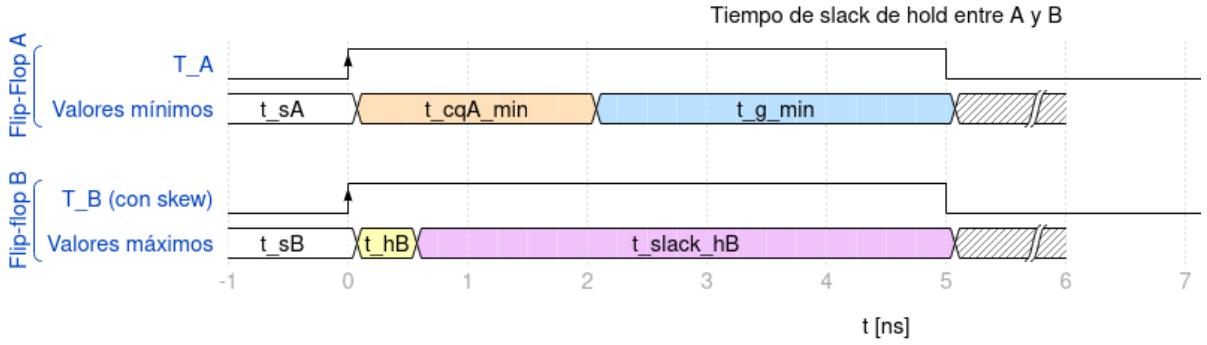
En el gráfico vemos ahora como entran todos los tiempos perfectamente en 10ns de período, sobrándome 3ns hasta el tiempo de setup del flip-flop B. (Nota T_ha y T_slack_hA no están perfectamente a escala).

[Codigo WaveDrom]



El gráfico del slack de hold lo hago acá porque es más fácil tener un período múltiplo de "10" para graficar. Acá vemos que la diferencia entre lo que tarda en procesar el nuevo dato el flip-flop A y lo que tarda el flip-flop B en estar listo para recibirlo es de ns. EL hecho de que justo termine en el flanco descendente del clock no tiene nada que ver.

[Código de WaveDrom]



C.

Con tsk = 0 y t_slack_s = 0 (frecuencia máxima es cuando estoy al borde del funcionamiento, sin margen de nada).

$$T_{mín} = t_{cqA-Max} + t_{g-Max} + t_{slack-s} + t_{sB-Max} - t_{sk-Min}$$

$$T_{mín} = 3 + 1.5 * 2 + 1$$
$$T_{mín} = 7ns \implies F_{MAX} = 142.86\,MHz$$

Que se condice con el resultado del punto a), donde la frecuencia era de 150MHz, mayor a la máxima obtenida en este punto.