# Memoria Sistemas Digitales
# Diciembre 2018

Rubén Palmer Pérez, 43474448D
Alejandro García Colombás, 49607577G

# 2.Introducción

El proyecto de este año eran dos ejercicios, el primero siendo una sistema combinacional y el segundo un sistema secuencial.

Esta breve introducción servirá para explicar en detalle cada uno de los ejercicios y sus posibles sub apartados utilizando imágenes para ayudar a la compresión de que circuito se habla en todo momento.

Al enfrentar cada uno de los ejercicios, lo primero que hicimos fue hacer algunos prototipo a papel, para entender a lo que no enfrentamos y hacer algunas sesiones de “brainstorming” donde presentaríamos posibles ideas, y por lo tanto soluciones, de los ejercicios sin preocuparnos mucho por su funcionamiento, ya que lo que se busca son soluciones originales o más fáciles juntando partes de muchas ideas.

El proceso de desarrollo de cada uno de los circuitos ha sido sin duda todo un desafío, sobre todo el combinacional ya que encontrábamos problemas de desarrollo cada segundo y teníamos que buscar maneras más interesantes de solucionar el problema o recurrir a hacer parches. Aunque todo nuestro esfuerzo se fue en hacer un circuito fácil de entender mediante procedimientos dados en clase, se nos hizo imposible hacer dicho circuito sin usar algunos parches, en concreto dos. El secuencial sin embargo fue algo más simple de entender y crear el diagrama de transiciones lo que aceleró drásticamente la velocidad de trabajo.

Esta vez, nosotros dos decidimos dividir el trabajo en dos partes muy obvias, secuencia y combinacional. García se encargaría de el trabajo de secuenciales mientras que Rubén se encargaría de combinacionales. Aunque han sido numerosas veces las que hemos quedado para intentar optimizar ambos sistemas, cada uno decidimos centrarnos en uno de los dos para intentar desarrollar una solución más densa ya que uno podría trabajar en un ambiente “familiar” hasta el final del proyecto.

# 3.Sistema combinacional

## a) Descripción de las partes identificadas.

1. Conversor a valor absoluto del input: Da el valor absoluto del input mediante multiplexores para dar su valor invertido y un sumador que le sumaría ‘1’ si el bit más significativo es 1

2. Módulo S/M: Si el bit más significativo del input es  y el selector del MUX M son, respectivamente ‘1’ y ‘0’, entonces este módulo le sumaría 1 al bit 5. En el caso de que sea 32, al estar ocupado para representar este bit, la suma daría un carry out de 1 al bit 6 expresando que el número 32 no se puede pasar a S/M con solo 6 bits

3. BCD: Consiste en un conjunto de sub-circuitos creados a base de los mapas de karnaugh

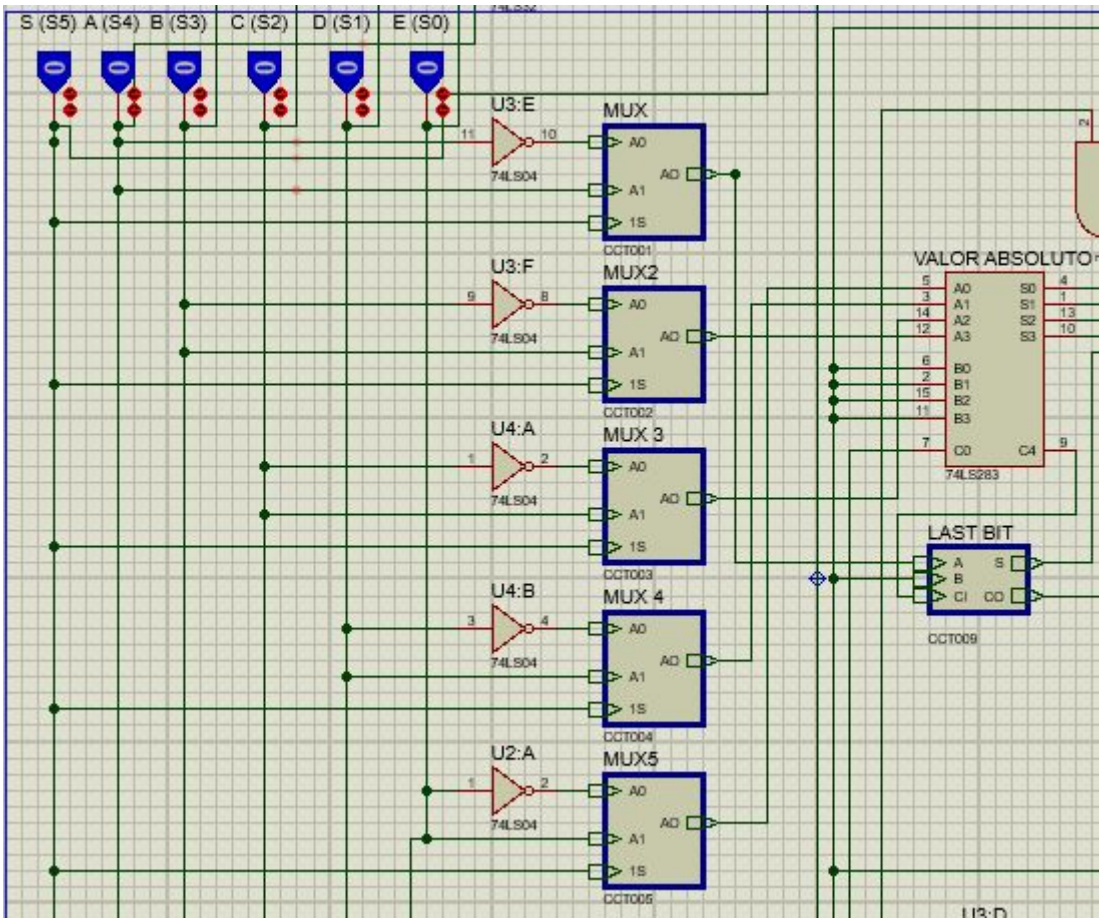
4. Sumador con el valor anterior: Consiste de dos fases. La primera determinar cuál es el valor anterior del input y la segunda un sumador entre el número anteriormente mencionado y el input.
    a. Valor anterior: Mediante un conjunto de multiplexores y un bit selector( el más significativo ), podemos determinar si sumarle ‘1’ o restarle 1 al input en valor absoluto, -1 si el selector es 0 y 1 si el selector es 1, ya que para el número -16 buscamos -17, no -15.
    b. Sumador A con A’: Un simple sumador entre el valor anterior y el valor del input en valor absoluto codificado en C2. Lo único a nombrar especial de este circuito es su bit de overflow, pues podemos tener overflow o “doble” overflow cuando el valor del output pasa de 64. Una simple puerta OR soluciona este problema ya que nos dice si hay overflow en el caso de que sea único o doble sin indicarnos cómo es este

5. Excepciones: Existen dos excepciones o también llamados parches, que arreglan la solución en los momentos en los que no sabíamos cómo arreglarlo sin rediseñar todo el sistema. En las imágenes, estas excepciones estarán colocadas en el apartado de la excepción ( 5.a en el apartado 4 y 5.b en el apartado 3 ) Para no liar y tener todo el contenido de una parte en un mismo sitio
    a. La primera existe en el sumador del valor anterior cuando el input es 0. Por como está diseñado el programa, un adder de 6 bits se coloca para que le sume o le reste ‘1’ al input mediante un bit selector ( el más significativo ). Si este es 0, se le restará 1 y si es 1 se le sumaría uno. En el caso de que el input sea ‘0’, se le restaría uno dándonos un resultado no deseado. Para arreglarlo, se crea dicha excepción y se le conecta a un multiplexor tal que, si la excepción es falsa, se toma el programa como tal pero, si la excepción es verdadera se tomará como si el bit selector fuese ‘1’ dándonos el valor absoluto de -1.
    b. La segunda surge en el módulo BCD, donde cuando el input es 32 exacto, vuelve a salirnos un valor indeseado. Parecido al anterior caso, creamos una excepción conectada a un multiplexor, si esta es falsa se toma el programa como tal y si es verdadera se fuerza el resultado deseado al output.

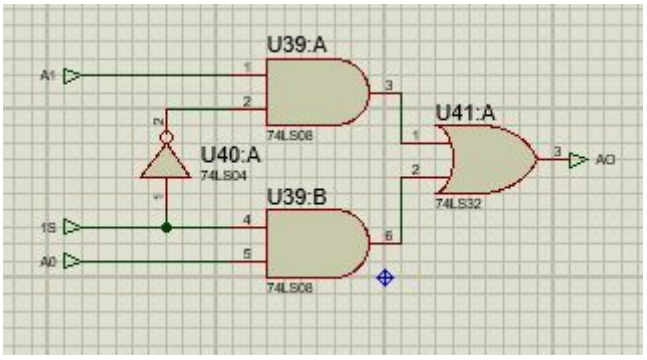
6. M Multiplexor: Finalmente encontramos un simple multiplexor que nos selecciona el output dependiendo de dos bit selectores entre 4 posibles outputs.

# b) Circuito de cada una de las partes ( y sus subpartes )

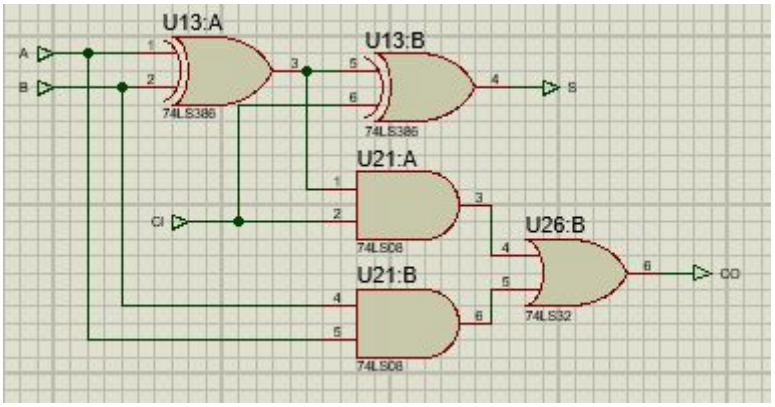
## 1.Conversor a valor absoluto



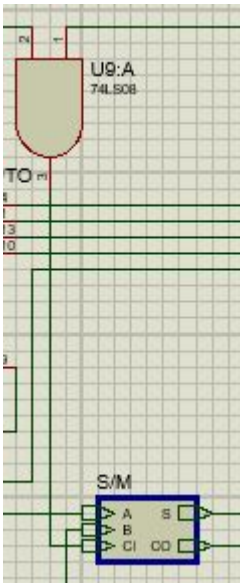

### 1.a) MUX ( todos tienen el mismo circuito interno )



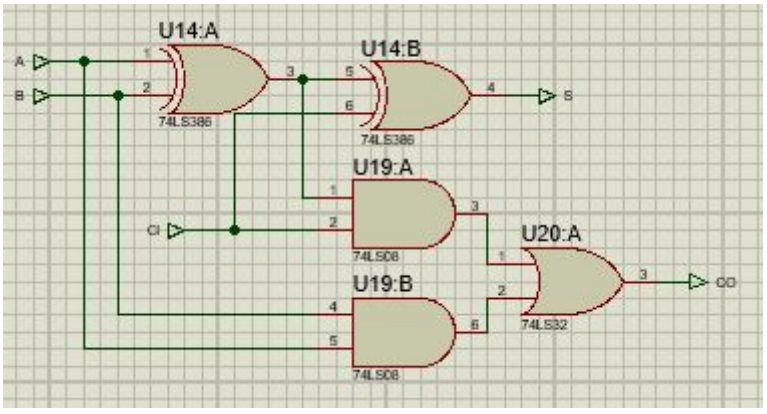

### 1.b) LAST BIT



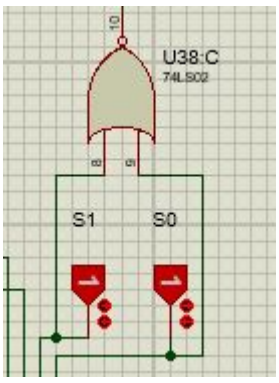
## 2. Modulo S/M:



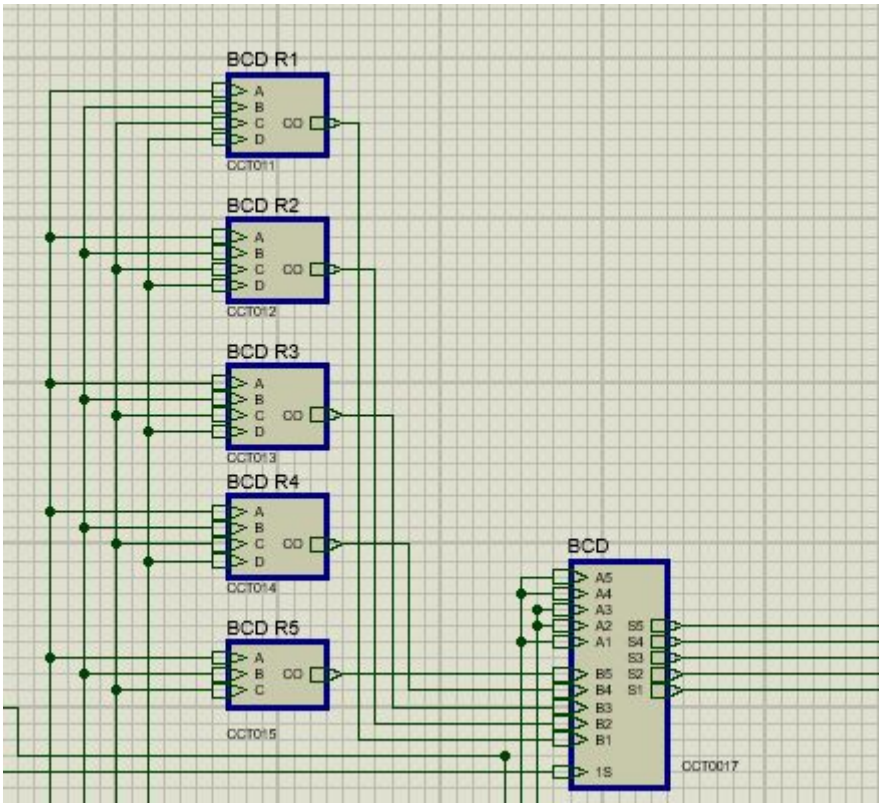

2.a) S/M



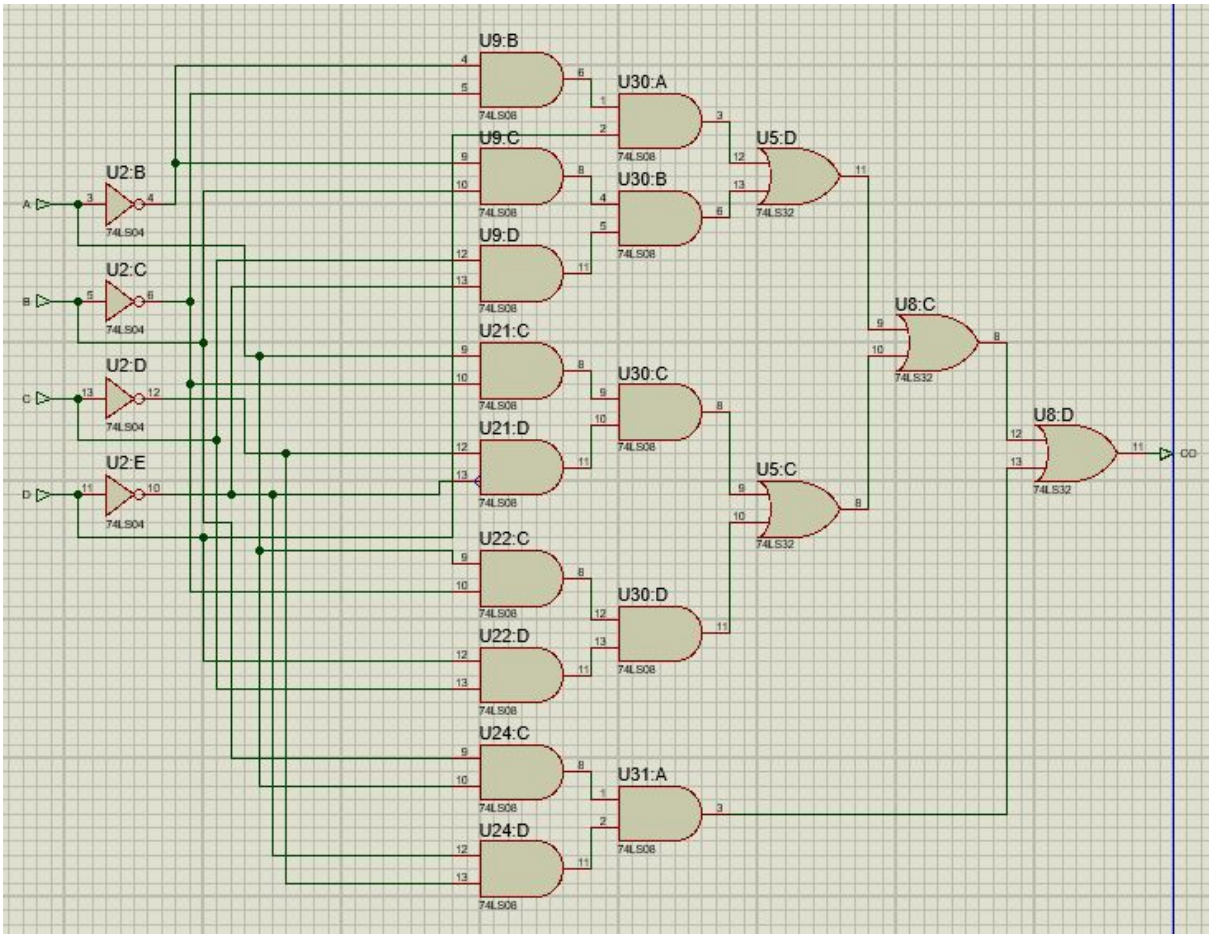

2.b) Lo que permite que la puerta U9.A sea verdadero con respecto a los bit selectores de MUX M



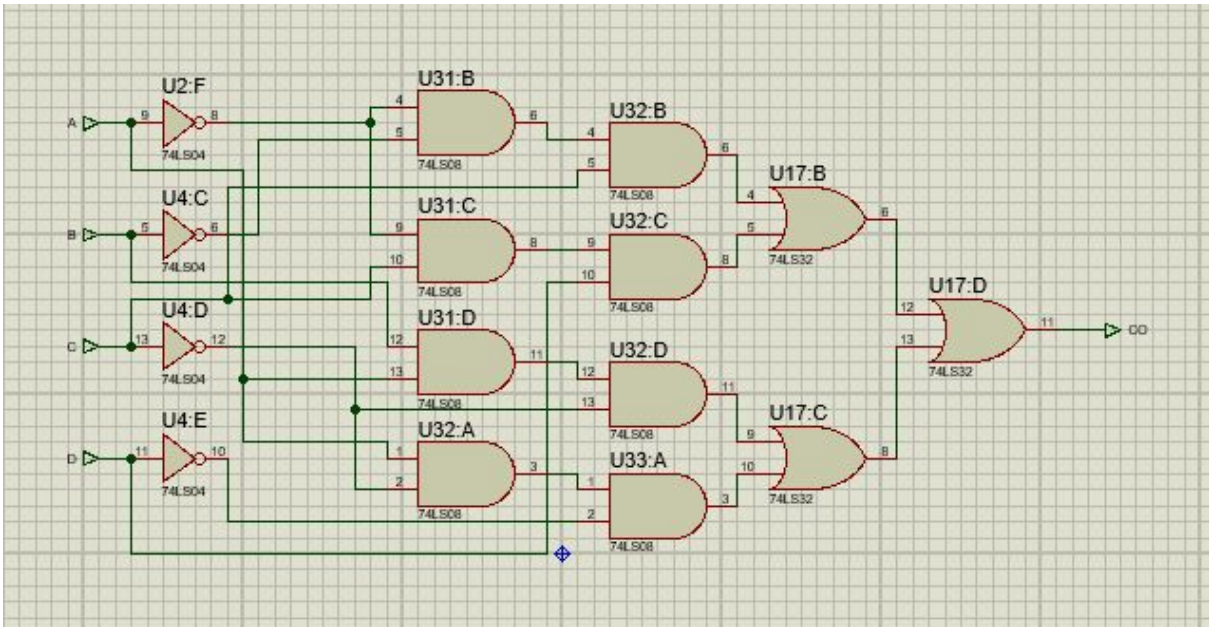
## 3.Módulo BCD



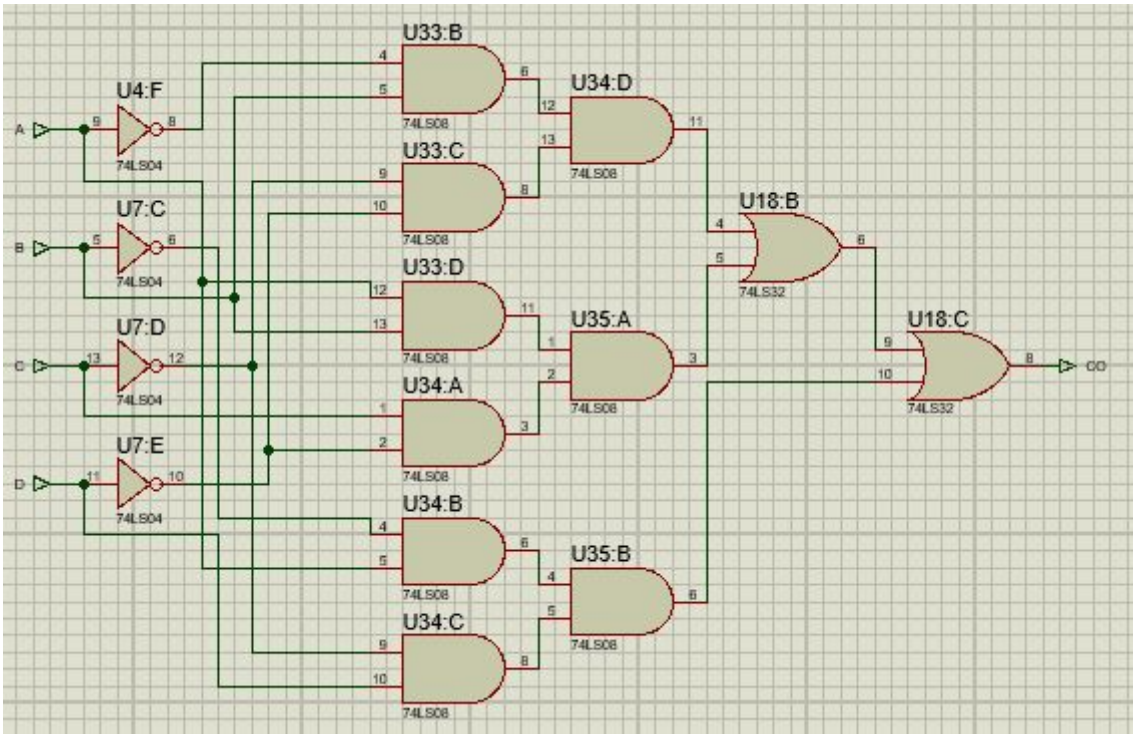

### 3.a)BCD R1



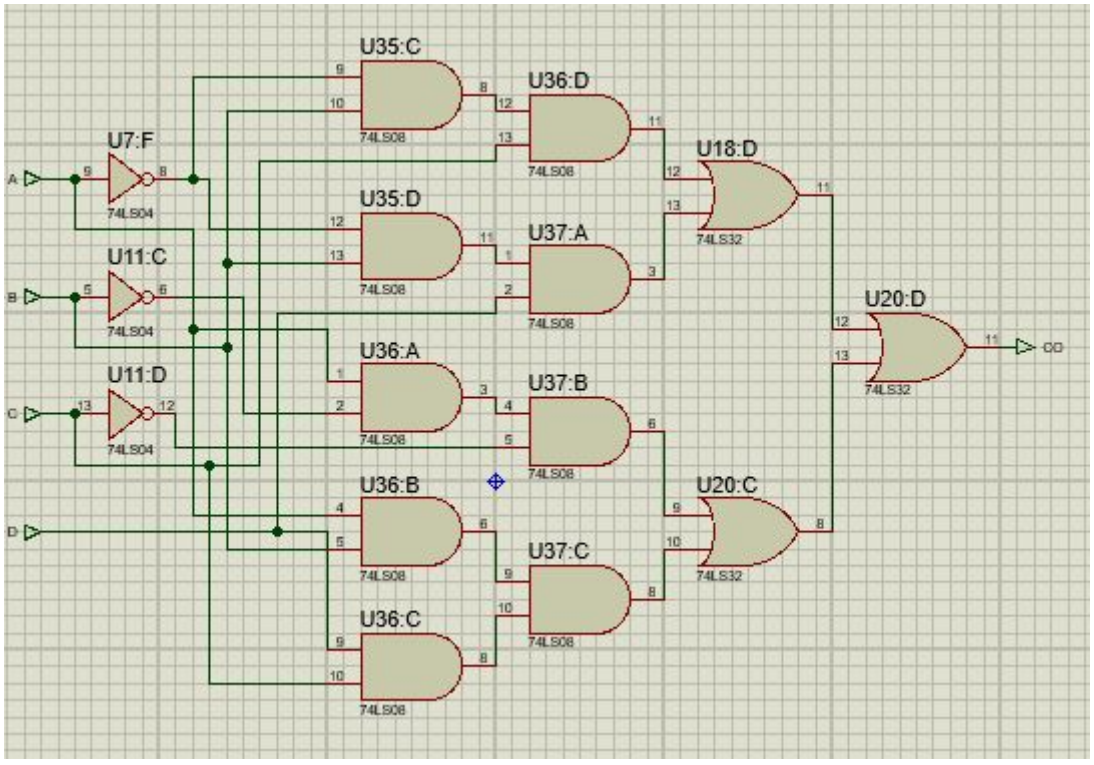
3.b) BCD R2



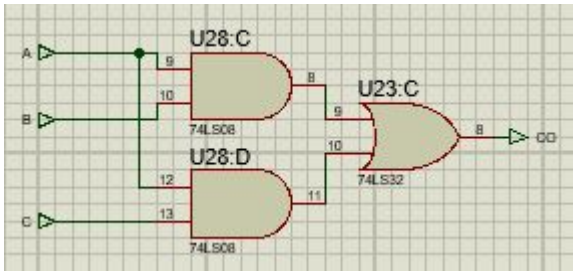

3.c) BCD R3



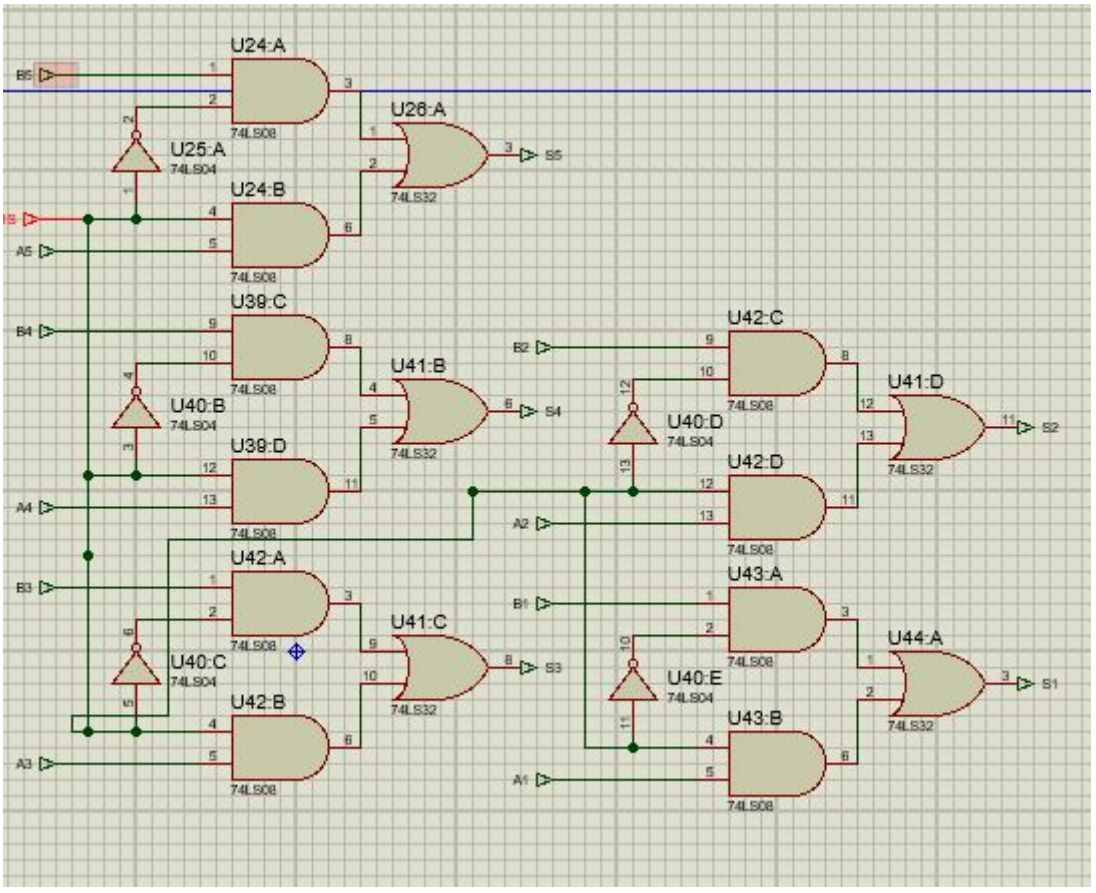
3.d) BCD R4



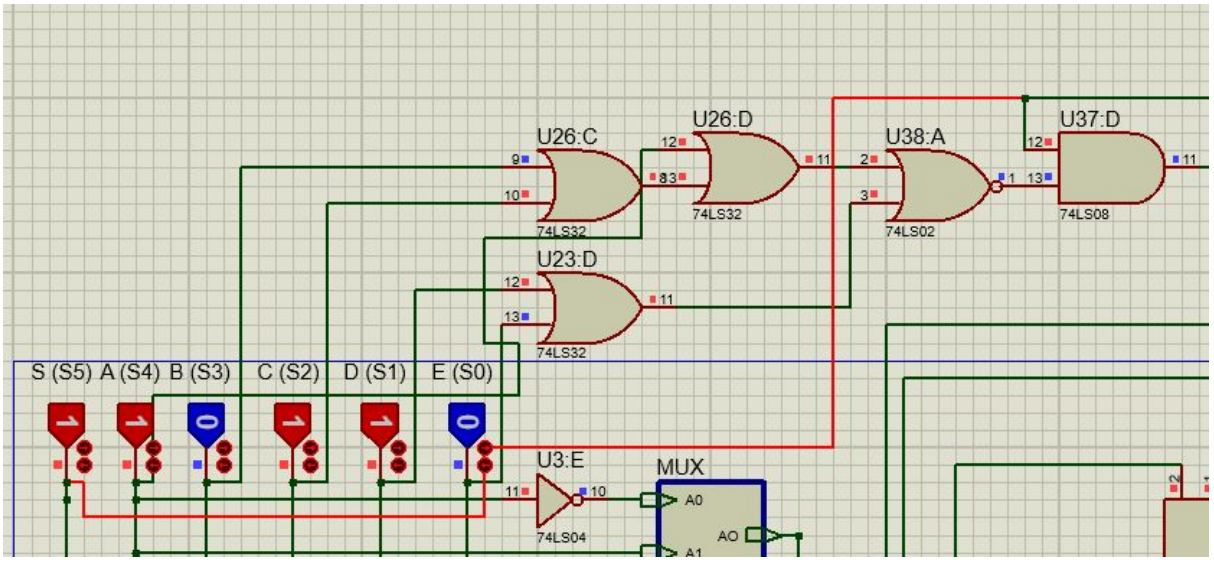

3.e)BCD R5




3.f) BCD

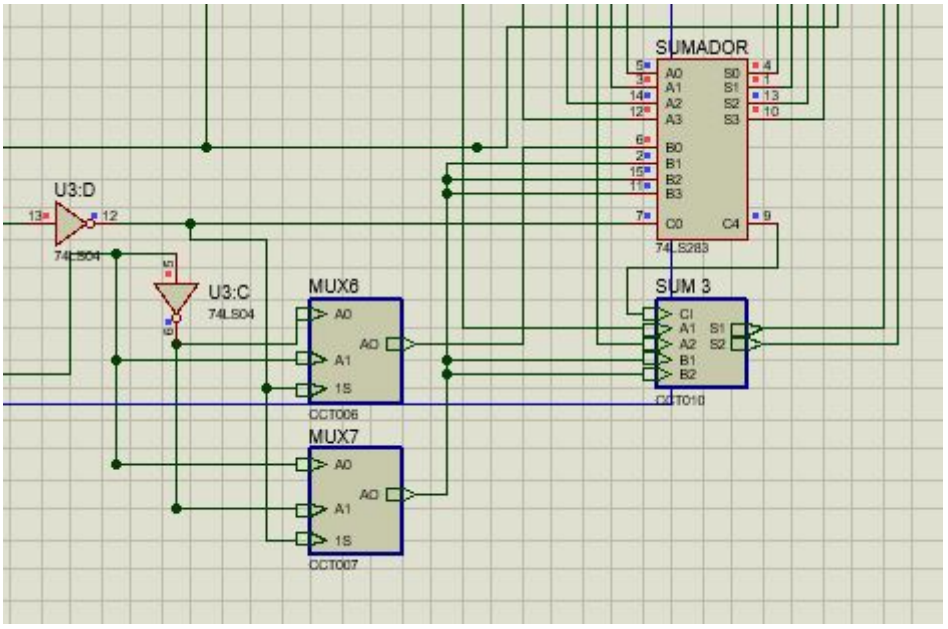


5.a) Excepcion del BCD

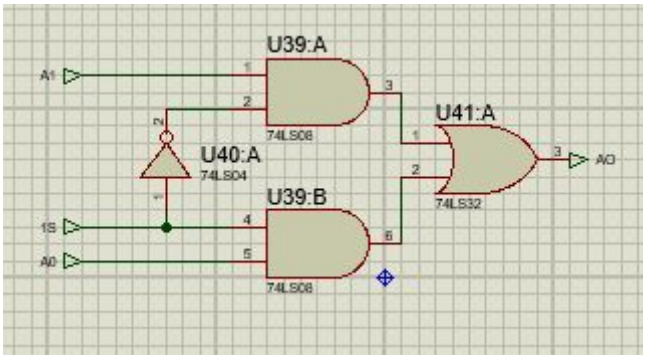



## 4) Sumador con el valor anterior

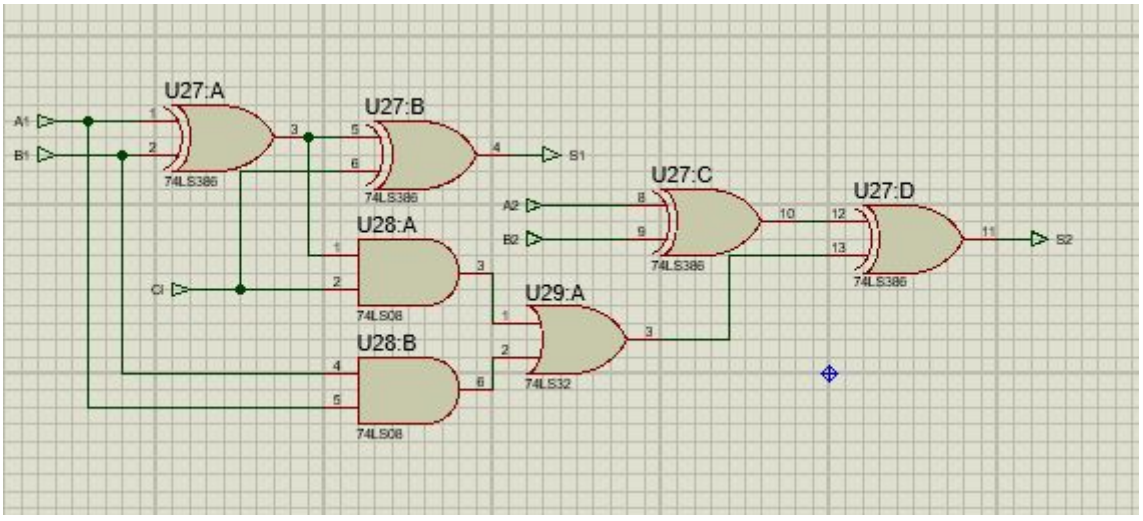
a) Primera parte: Determinar el valor anterior



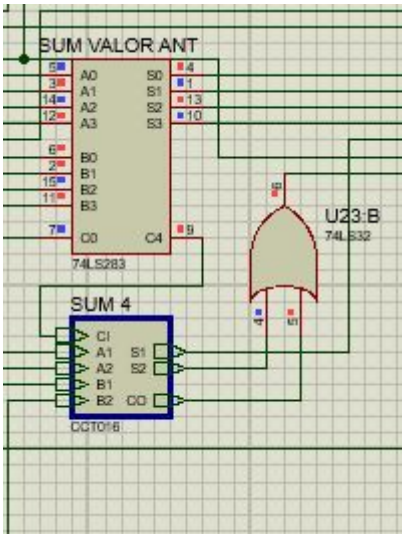

a.1) MUX ( Todos son iguales )



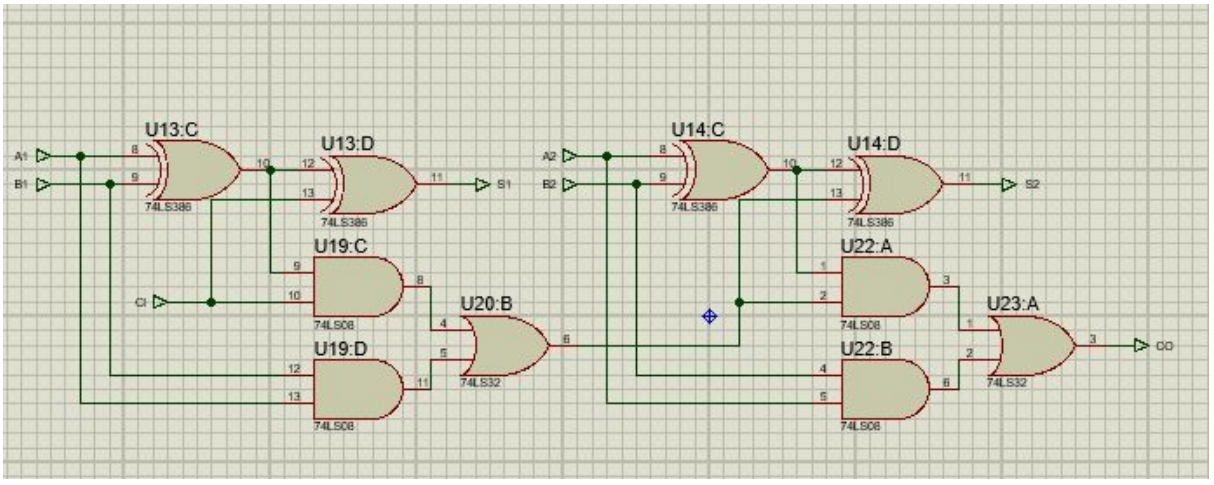
a.2) SUM 3



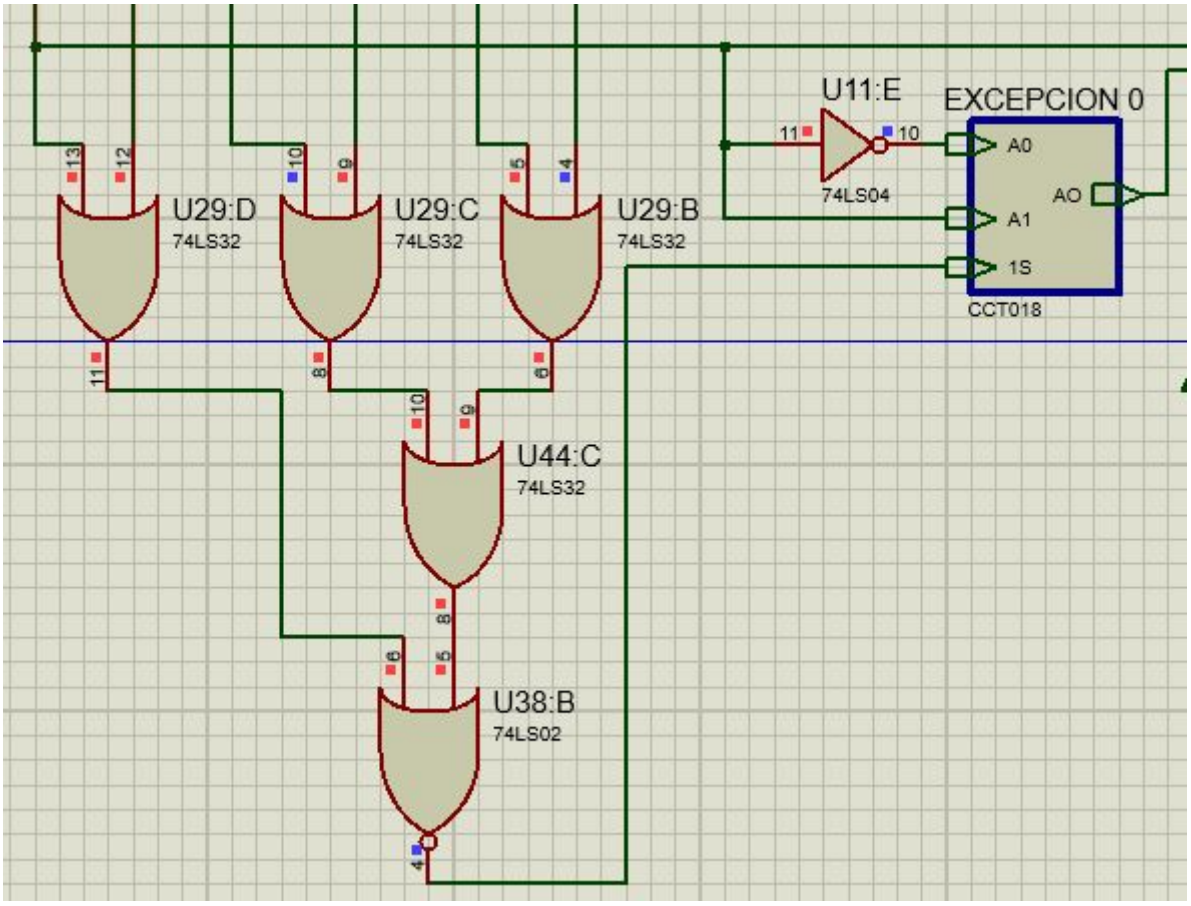

b) Segunda parte: Sumador del input y su valor anterior.
(La puerta OR nos indica Overflow)



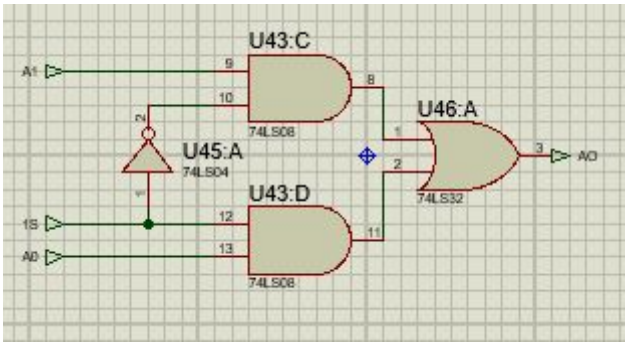

b.1)



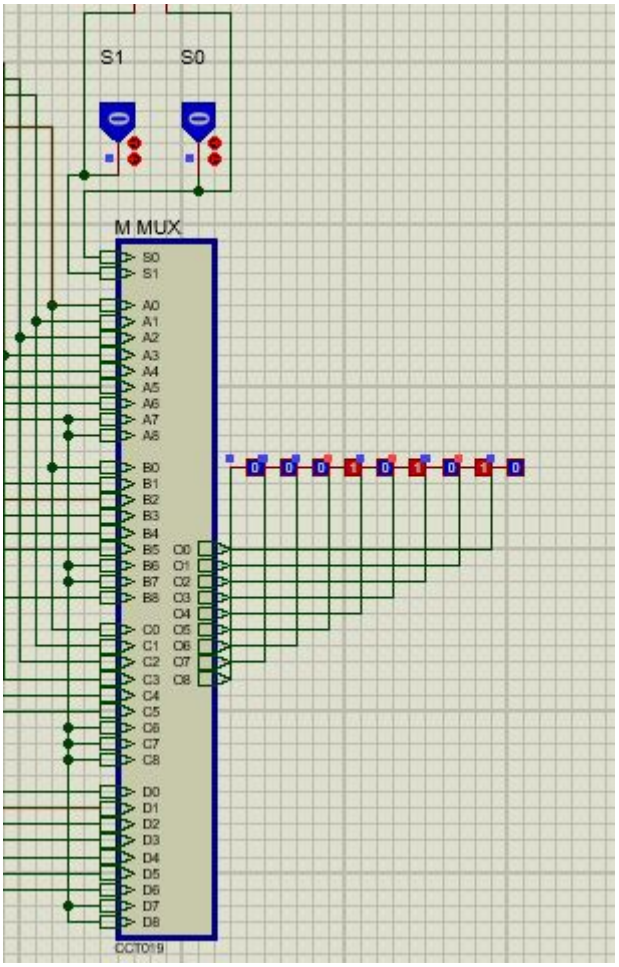
5.b) Excepción en el caso de que el input sea 0



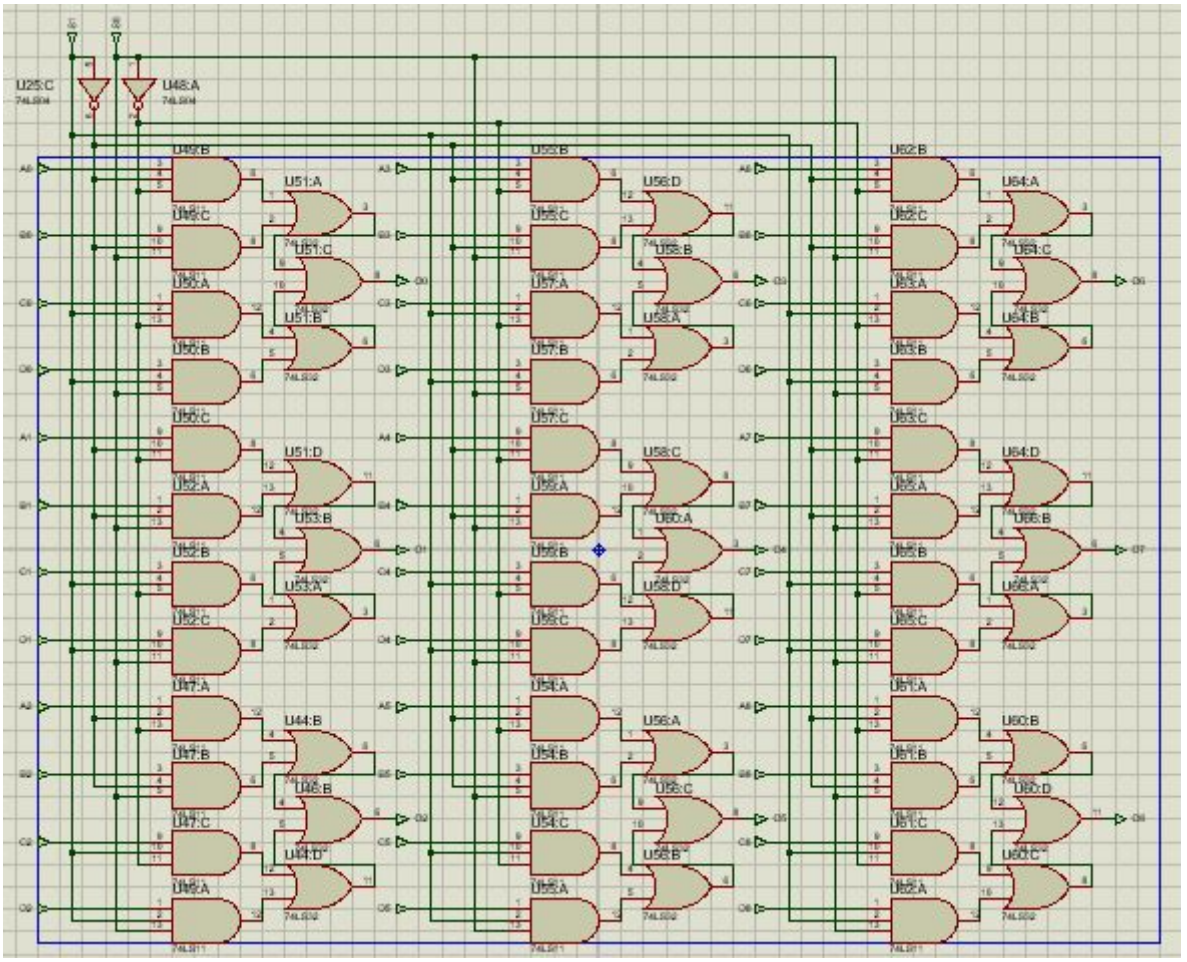

5.b.1) Excepción 0 ( es un MUX )



## 6. M MUX




6.a) M MUX ( Interior )



# c) Pruebas realizadas.

Daremos un ejemplo para cada uno de los casos y daremos especial importancia a las excepciones para entender cómo funcionan. Empezaremos por orden del multiplexor ( S/M, BCD, etc ). Para todos ejemplos se tomará el valor 10 en binario (1010) como caso general y se tomarán otros más “interesantes” para observar cómo cambian los circuitos.

1) S/M, Caso 0 0
   El input en este caso es (001010), gracias a que ninguno de los bits de selección son “1” la puerta NOR ( Referida en 2.b) del apartado b) ) Se encuentra en estado “1”, lo que permite que una entrada de la puerta AND ( Referida en 2 del apartado b) ) sea también “1” pero ya que el bit más significativo no es “1” la puerta AND se queda en “0”. En este caso pues, y ya que el bit más significativo es “0”, el circuito 1, no hace nada al dar los valores tal cual que entran como output. Con un ejemplo diferente, el -10 ( 110110 ) sucede algo parecido. En este caso el bit más significativo si que es uno lo que permite a la puerta AND ser “1”. Este valor positivo añade un bit de “carry in” en el circuito referenciado en 2.a) del apartado anterior. Ya que el bit más significativo es “1” el circuito 1 de la input negada ( 001001 ) y le añade un bit mediante un adder ( 001010 ), esto nos da el valor absoluto del input sin estar codificado en c2. Finalmente se cambia el bit 5 por “1” mediante el adder.
   Otro caso interesante es el de -32. Mirando todo el procedimiento el output antes de convertirlo en S/M es de (100000) ya que no estamos codificando en c2. Al sumar un bit al bit 5 ya ocupado, obtenemos un carry out al 6ºbit, lo que nos informa que no se puede pasar a S/M el número -32 por falta de bits.

2) BCD, Caso 0 1
   No hay mucho que decir para este caso en particular ya que se utilizó karnaugh. Lo único interesante a mencionar es la excepción 5.a) que es verdadera si el input es exactamente (100000) lo que hace cambiar el Mux 3.f) para forzar el output al deseado.

3) Valor Absoluto, Caso 1 0
   Por como está diseñado este circuito, pasar a valor absoluto el input fue algo necesario para que otros circuitos funcionasen, como el caso 0 0 ( S/M ). Si el bit más significativo es 0, copiara el input en el output directamente, ya que no presenta ningún valor negativo. En el momento que el 5º Bit sea “1”, los multiplexores referenciados 1.a) darán el valor invertido de tal. En el caso de -10 ( 110110 ) daría ( 001001 ) y un sumador actuaría ante el bit 0, añadiendo “1” ( 001010 ).

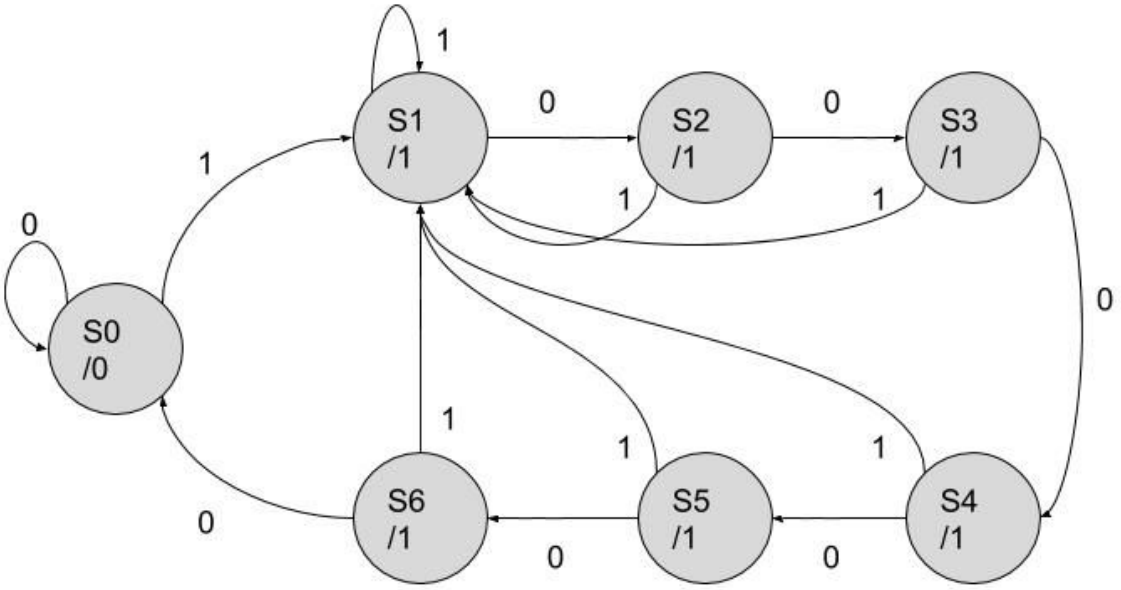
4) La suma en c2 del valor de la entrada con su anterior, Caso 1 1
   Sin duda, el más complicado. Otra vez, utilizaremos los valores 10 y -10 para expresar las dos posibilidades del circuito. Lo primero a tener en cuenta es si el valor es negativo, ya que el valor anterior de -10 ( -11 ) y 10 ( 9 ) no son el mismo. Este problema se resuelve con un conjunto de multiplexores, que actúan con un bit selector ( 5º Bit ) que conecta con un sumador. En el caso que el número sea negativo, se le sumaría “1” al su valor absoluto en binario natural ( reutilizamos el caso 3 ) y si es positivo se le restaría 1 ( para 001010 → 001001 y para 110110 → 001010 → 001011) . Con esto ya tendríamos los valores anteriores mediante los sumadores 4.a.2)

En la segunda parte sumariamos ambos valores con un sumador. Lo único a tener en cuenta es el overflow, ya que podemos tener overflow de 32 o de 64. Para solucionar este caso, utilizamos una puerta OR ( 4.b) ) para que nos indique si existe cualquier tipo de overflow.
En este apartado encontramos la segunda excepción cuando el input es 000000. Por como funciona el circuito, el 0 lo toma como si fuese un valor positivo, restando “1”, que en el adder es 01111. En este caso encontramos un output no deseado ( 0001111) por lo que se hace una excepción conectada a un multiplexor ( 5.b) y 5.b.1) ). Cuando el input es 00000, la excepción es verdadera y el multiplexor da como salida “1”. Esto, esencialmente, hace que el valor 00000 lo tome como uno negativo sumándole 1 al input, dándonos el resultado deseado, el valor absoluto de -1 ( 00001 ). Seguidamente, se suman 00000 y 00001 dándonos 00001.

5) Lo último a tener en cuenta es el multiplexor final, que simplemente elige el output dependiendo de dos bits selectores.

# 4.Sistema secuencial

## a) Diagrama de transición de estados




## b) Codificación de estados

| | Q2 | Q1 | Q0 |
|---|---|---|---|
| S0 | 0 | 0 | 0 |
| S1 | 0 | 0 | 1 |
| S2 | 0 | 1 | 0 |
| S3 | 0 | 1 | 1 |
| S4 | 1 | 0 | 0 |
| S5 | 1 | 0 | 1 |
| S6 | 1 | 1 | 0 |

## c) Justificación del tipo de máquina empleado ( Mealy o Moore )

Para este ejercicio hemos empleado una máquina de Moore. Hemos utilizado este tipo de máquina debido a que la salida del circuito depende únicamente del estado en que nos encontramos. En este problema nuestra salida Z solo dependia de si había o no, un contenedor en la cinta, por lo tanto, esta máquina es más intuitiva que una de Mealy, en este caso en concreto.

## d) Tabla de transición de estados y de salida

| Q2 | Q1 | Q0 | X | Q2’ | Q1’ | Q0’ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | X | X | X | X |
| 1 | 1 | 1 | X | X | X | X |

## e) Minimización de funciones

Tabla de entrada de los Flip-Flop

| Z | J2 | K2 | J1 | K1 | J0 | K0 |
|---|---|---|---|---|---|---|
| 0 | 0 | X | 0 | X | 0 | X |
| 0 | 0 | X | 0 | X | 1 | X |
| 1 | 0 | X | 1 | X | X | 1 |
| 1 | 0 | X | 0 | X | X | 0 |
| 1 | 0 | X | X | 0 | 1 | X |
| 1 | 0 | X | X | 1 | 1 | X |
| 1 | 1 | X | X | 1 | X | 1 |
| 1 | 0 | X | X | 1 | X | 0 |
| 1 | X | 0 | 0 | X | 1 | X |
| 1 | X | 1 | 0 | X | 1 | X |
| 1 | X | 0 | 1 | X | X | 1 |
| 1 | X | 1 | 0 | X | X | 0 |
| 1 | X | 1 | X | 1 | 0 | X |
| 1 | X | 1 | X | 1 | 1 | X |
| X | X | X | X | X | X | X |
| X | X | X | X | X | X | X |

J2=Q1 Q0 X^
K2=Q1+X
J1=Q0 X^
K1=Q2+Q0+X
J0=X+Q2^ Q1+Q2 Q1^
K0=X^

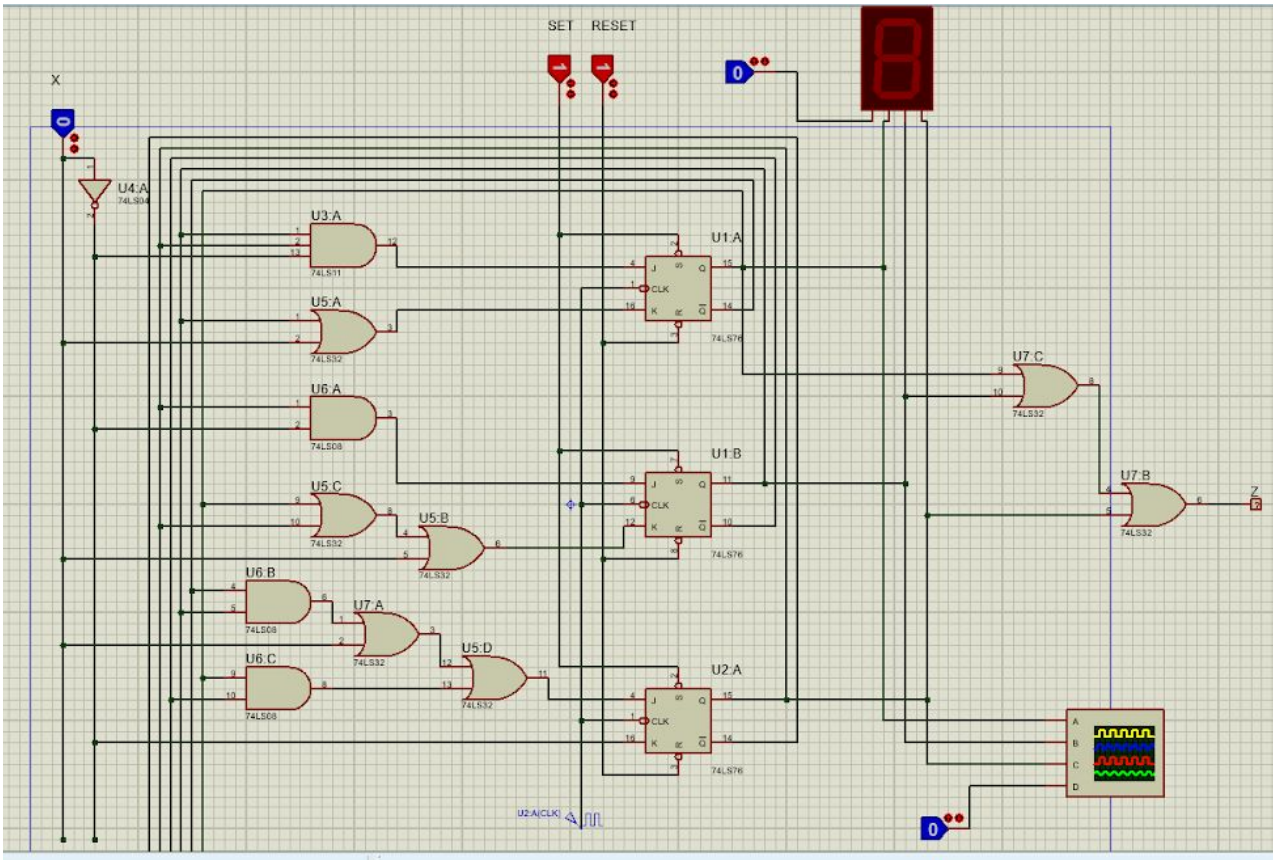
Tabla de salida(Z)

| Q2 | Q1 | Q0 | Z |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |

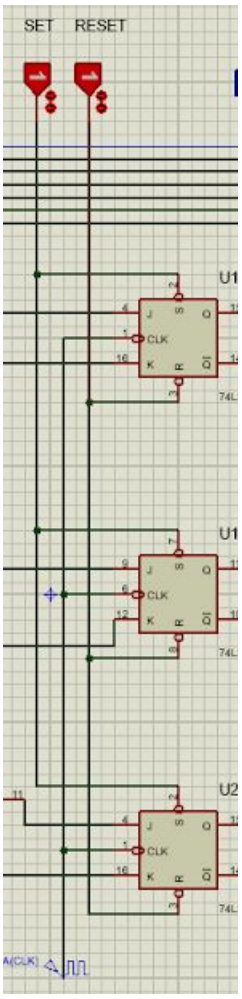
Z=Q2+Q1+Q0 (no depende de X)

## f) Implementación con Proteus

1.Estructura completa de la implementación con Proteus

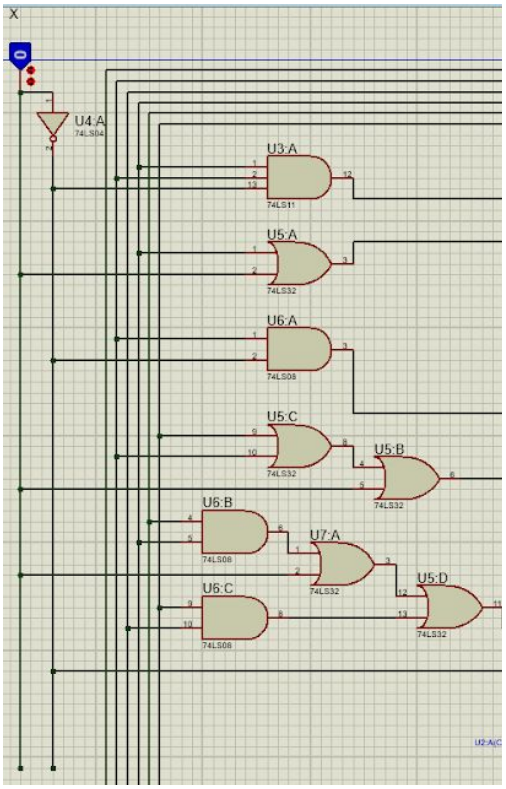


2.Tres Flip-Flops JK, uno para cada bit de estado



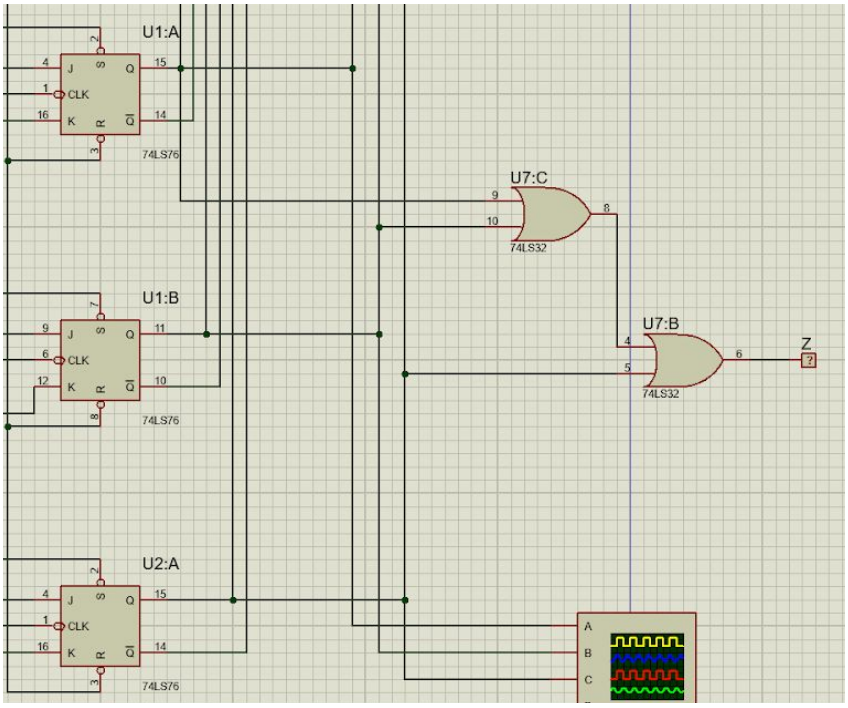

Cada Flip-Flop está conectado a un CLK, que regula los pulsos de reloj del circuito, y a una entrada de SET y otra de RESET.

3.Funciones simplificadas de cada una de las entradas de los Flip-Flop

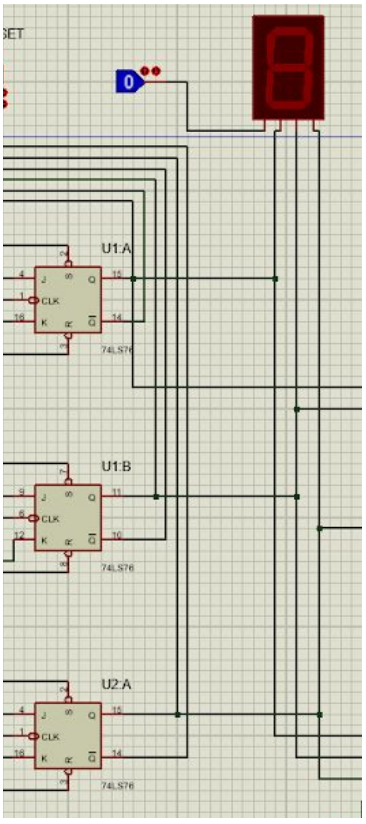



Funciones de las entradas J y K de los Flip-Flop. Simplificadas en forma de suma de productos utilizando mapas de Karnaugh.

4.Simplificación de la función de salida



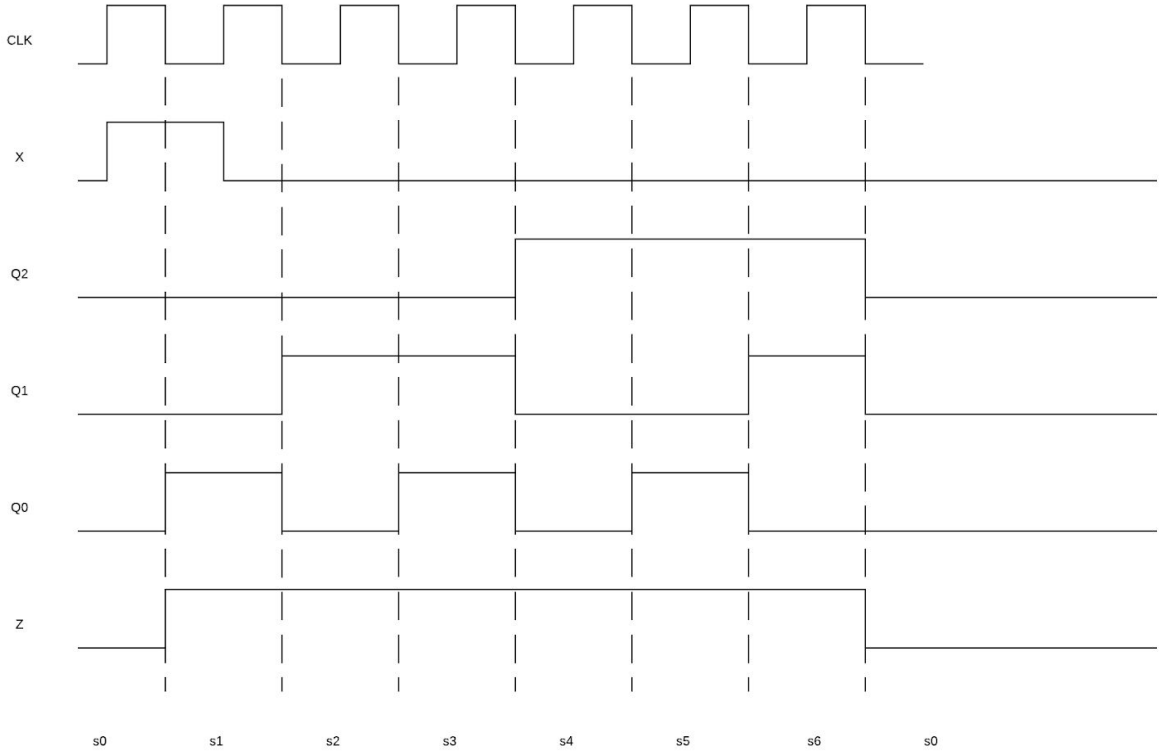

5.Display BCD 7-segments




Las salida del los Flip-Flop se conectan al Display BCD para indicar en qué posición de la cinta se encuentra el último contenedor que entró en la cinta, es  decir, en qué estado del diagrama nos encontramos.

### g) Juego de pruebas y cronograma de las salidas para un ejemplo

La prueba que vamos ha realizar es muy sencilla. Veremos que pasa cuando solo ponemos una caja sobre la cinta y no ponemos ninguna en los siguientes 7 pulsos de reloj.
Con esta prueba vemos como la caja recorre cada una de las posiciones de la cinta hasta que sale de ella y el motor de la cinta se detiene.




# 5.Conclusión

Durante el desarrollo de estos proyectos hemos adquirido una gran cantidad de conocimientos y cualidades. Ha sido una experiencia muy satisfactoria en muchos momentos del desarrollo pero no han faltado los momentos de frustración y preocupación.

Para empezar hemos mejorado nuestro trabajo en equipo, debido a que hemos tenido que coordinar en buena medida nuestro trabajo. Hemos tenido que lidiar con la manera de pensar de otro para sacar la mejores conclusiones posibles sobre los temas y problemas del trabajo.

Sobre la asignatura de sistemas digitales y del programa Proteus, hemos fijado los conocimientos de las clases teóricas y prácticas, incluso los hemos ampliado. Además, este trabajo nos ha ayudado a coger soltura en la simplificación de funciones y el entendimiento de muchos circuitos.