Mattius Alexander Cardona Franco

Daniel Felipe Rivera Arroyave

Informe de laboratorio

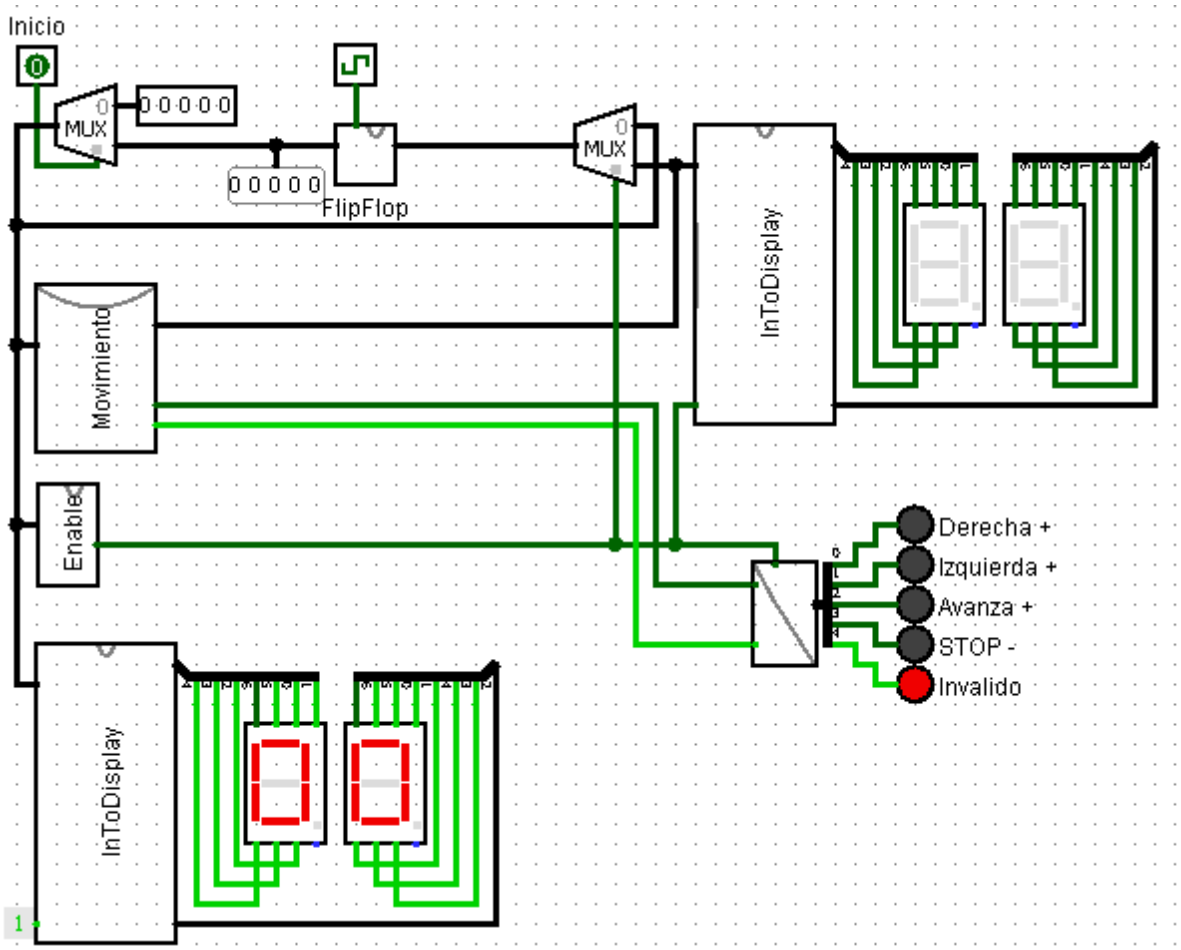
2508355-

Arquitectura de computadores y laboratorio

Fredy Alexander Rivera Velez

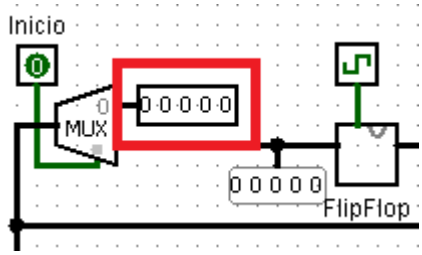
2018-1

La descripción del proceso de creación, la implementación de las tablas de verdad y los mapas de Karnaugh se harán en orden de “entrada” y siguiendo el recorrido de los datos para mantener un orden.



(Figura 1: Circuito completo funcionando)

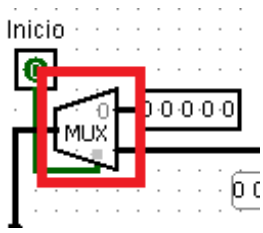
Para empezar, el sistema cuenta con un “PIN” que funciona como entrada de datos, el cual recibe 5 bits formando así un número entero entre el 0 y el 31 escrito en binario:



(Figura 2: PIN de entrada de datos)

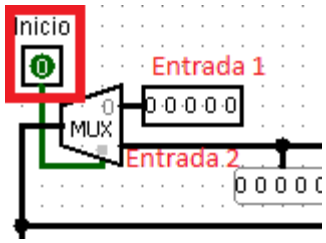
Dicho pin envía por medio de un bus los 5 bits como señales individuales correspondiendo cada una a un “1” o un “0”.

Esta señal entra a un **Multiplexor** que consta de dos entradas y un seleccionador de salida, La otra entrada que recibe el “MUX” es el “número de salida” o “posición siguiente” que obtendremos más adelante:



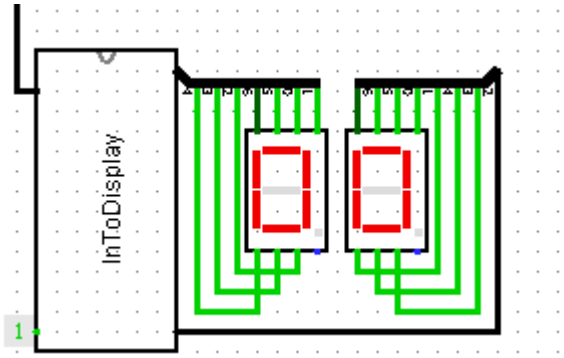
(Figura 3: Multiplexor 1)

La señal del seleccionador que recibe este **multiplexor** al estar en “0” o apagada permite el paso de la **entrada 1 (**correspondiente al PIN de entrada), mientras que si lo encendemos para que la señal enviada sea “1”, bloquea la **entrada 1** y abre paso a la **entrada 2** (correspondiente al número de salida o posición siguiente):



(Figura 4: Seleccionador)

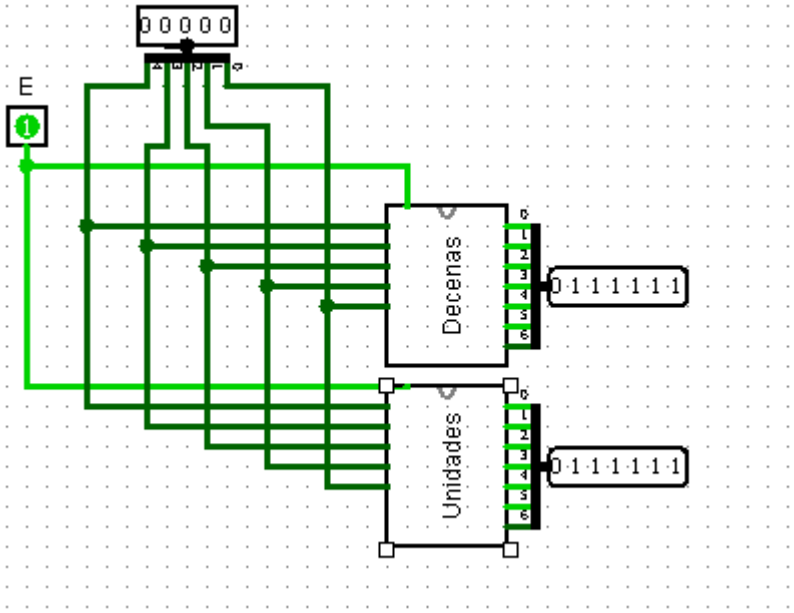
El bus que sale del multiplexor y que lleva consigo la señal de 5 bits con el **número de inicio**, se dirige en primer lugar y sin ningún tipo de filtro hacia un circuito al cual llamamos “**InToDisplay**”, cuya función es convertir el número de entrada que está en binario, en una imagen gráfica y legible en decimal:



(Figura 5: InToDisplay con entrada 00000)

A este **InToDisplay** le asignamos dos entradas y dos salidas, una entrada es el bus que lleva consigo el número de inicio en binario y la otra es un habilitador que en este caso es una constante que lo mantiene en funcionamiento; Una de las salidas lleva consigo un bus con las **decenas** del número que ingresamos y la segunda otro bus con las **unidades** de dicho número.

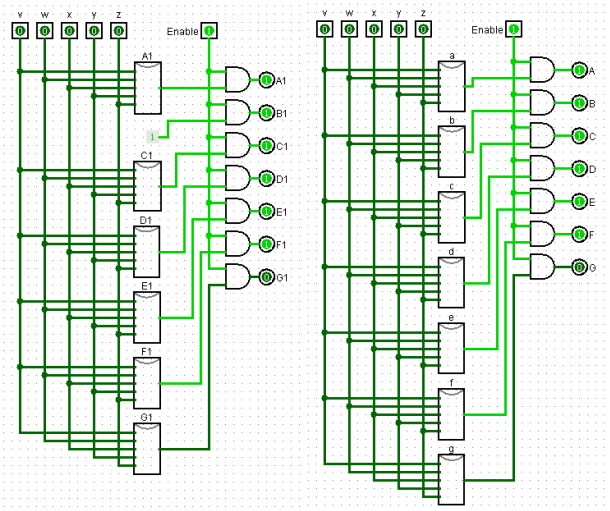
La estructura interna del **InToDisplay** es la siguiente:



(Figura 6: Estructura del InToDisplay)

Como se puede apreciar en la figura, la entrada de datos se divide en diez , cinco correspondientes a las decenas y cinco a las unidades, para hacer internamente en dos subcircuitos (decodificadores 2:7 con habilitador) la conversión para el cableado de los displays.

En el caso de este InToDisplay el pin denominado como **habilitador** no recibe mayor importancia debido a que siempre está encendido, más adelante tendrá su debida explicación y forma de manejo.



(Figura 7: Decenas)

(Figura 8: Unidades)

En ambos circuitos nos encontramos con una cadena formada de subcircuitos cuya arquitectura está compuesta por compuertas lógicas correspondientes a los mapas de Karnaugh formados a partir de la siguiente tabla de verdad:

	v	w	x	y	z	a1	b1	c1	d1	e1	f1	g1	a	b	c	d	e	f	g
0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0
2	0	0	0	1	0	1	1	1	1	1	1	0	1	1	0	1	1	0	1
3	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	1
4	0	0	1	0	0	1	1	1	1	1	1	0	0	1	1	0	0	1	1
5	0	0	1	0	1	1	1	1	1	1	1	0	1	0	1	1	0	1	1
6	0	0	1	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1
7	0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	0	0
8	0	1	0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	0	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1
10	0	1	0	1	0	0	1	1	0	0	0	0	1	1	1	1	1	1	0
11	0	1	0	1	1	0	1	1	0	0	0	0	0	1	1	0	0	0	0
12	0	1	1	0	0	0	1	1	0	0	0	0	1	1	0	1	1	0	1
13	0	1	1	0	1	0	1	1	0	0	0	0	1	1	1	1	0	0	1
14	0	1	1	1	0	0	1	1	0	0	0	0	0	1	1	0	0	1	1
15	0	1	1	1	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1
16	1	0	0	0	0	0	1	1	0	0	0	0	1	0	1	1	1	1	1
17	1	0	0	0	1	0	1	1	0	0	0	0	1	1	1	0	0	0	0
18	1	0	0	1	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1
19	1	0	0	1	1	0	1	1	0	0	0	0	1	1	1	1	0	1	1
20	1	0	1	0	0	1	1	0	1	1	0	1	1	1	1	1	1	1	0
21	1	0	1	0	1	1	1	0	1	1	0	1	0	1	1	0	0	0	0
22	1	0	1	1	0	1	1	0	1	1	0	1	1	1	0	1	1	0	1
23	1	0	1	1	1	1	1	0	1	1	0	1	1	1	1	1	0	0	1
24	1	1	0	0	0	1	1	0	1	1	0	1	0	1	1	0	0	1	1
25	1	1	0	0	1	1	1	0	1	1	0	1	1	0	1	1	0	1	1
26	1	1	0	1	0	1	1	0	1	1	0	1	1	0	1	1	1	1	1
27	1	1	0	1	1	1	1	0	1	1	0	1	1	1	1	0	0	0	0
28	1	1	1	0	0	1	1	0	1	1	0	1	1	1	1	1	1	1	1
29	1	1	1	0	1	1	1	0	1	1	0	1	1	1	1	1	0	1	1
30	1	1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	1	1	0
31	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	0	0	0	0

(Tabla de verdad 1)

Las columnas "**v w x y z"** corresponden a el número de entrada dividido bit a bit, desde la columna "**a1"** a la **"g1"** corresponden a los leds necesarios para formar el número de las décimas en el display y de la columna "**a"** a la "**g"** corresponden a los leds de las unidades, los mapas de Karnaugh encargados de la simplificación de las funciones son los siguientes:

A1:

wx / yz	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	0
10	1	1	0	0
V=0				

wx / yz	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1
V=1				

F(v,w,x,y,z)= v'w'+w'x+vw+v'x'y'

B1:

wx / yz	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1
V=0				

wx / yz	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1
V=1				

F(v,w,x,y,z)= 1

C1:

wx / yz	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1
V=0				
wx / yz	00	01	11	10
00	1	0	0	0
01	1	0	0	0
11	1	0	1	0
10	1	0	1	0
V=1				

F(v,w,x,y,z)= v'+w'x'+wxy

D1:

wx / yz	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	0
10	1	1	0	0
V=0				
wx / yz	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1
V=1				

F(v,w,x,y,z)= v'w'+w'x+vw+v'x'y'

E1:

wx / yz	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	0
10	1	1	0	0
V=0				

wx / yz	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	0	1	0	1
10	0	1	0	1
V=1				

F1:

wx / yz	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	0
10	1	1	0	0
V=0				

wx / yz	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0
V=1				

G1:

wx / yz	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0
V=0				

wx / yz	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1
V=1				

A:

wx / yz	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	0
10	1	1	0	0
V=0				
wx / yz	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1
V=1				

F(v,w,x,y,z)=

B:

wx / yz	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1
V=0				
wx / yz	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1
V=1				

F(v,w,x,y,z)=

C:

wx / yz	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1
V=0				

wx / yz	00	01	11	10
00	1	0	0	0
01	1	0	0	0
11	1	0	1	0
10	1	0	1	0
V=1				

F(v,w,x,y,z)=

D:

wx / yz	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	1	0	1	0
10	1	1	0	1
V=0				

wx / yz	00	01	11	10
00	1	1	1	0
01	0	0	1	1
11	1	1	0	0
10	1	1	1	1
V=1				

F(v,w,x,y,z)=

E:

yz \ wx	00	01	11	10
00	1	0	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

V=0

yz \ wx	00	01	11	10
00	1	1	1	0
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

V=1

F:

yz \ wx	00	01	11	10
00	1	1	0	1
01	0	1	0	1
11	0	0	1	0
10	0	1	1	1

V=0

yz \ wx	00	01	11	10
00	1	1	1	1
01	0	0	1	1
11	1	0	0	0
10	1	0	1	1

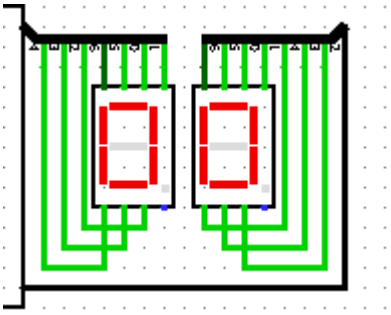
V=1

G:

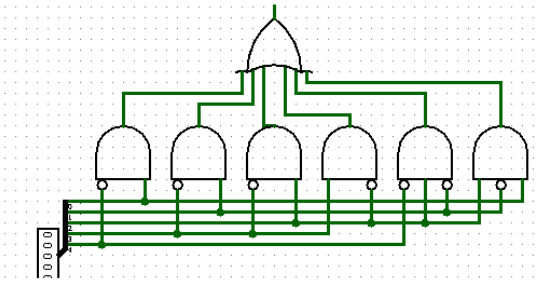
wx / yz	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	0	1	0
10	1	1	1	0
V=0				

wx / yz	00	01	11	10
00	1	0	1	1
01	0	0	1	1
11	1	1	0	0
10	1	1	0	1
V=1				

Luego de que el número pasa por los circuitos **decenas** y **unidades** obtenemos dos salidas de 7 bits que se usan mas adelante para encender los leds pertinentes fuera del **InToDisplay**.



Continuando con la señal del bus que usamos como entrada en el **InToDisplay** podemos observar que también es usado como entrada en un circuito llamado "**enable "** que usamos para detectar cuando es ingresado un número "invalido" o que esté fuera del recorrido de nuestro robot (0, 14, 16, 17, 28, 30, 31):



La tabla de verdad usada para la simplificación y los mapas de Karnaugh son:

Inválidos

wx / yz	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	0	1
V=0				

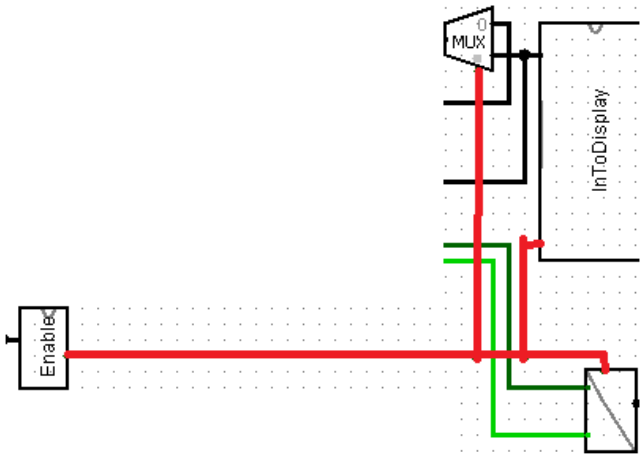
wx / yz	00	01	11	10
00	0	1	0	1
01	0	1	1	1
11	1	1	0	1
10	1	1	0	1
V=1				

F(v,w,x,y,z)= v'z+w'y+w'x+wx'+v'xy'+xy'z

	v	w	x	y	z	X
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	0	0	1	0	1
3	0	0	0	1	1	1
4	0	0	1	0	0	1
5	0	0	1	0	1	1
6	0	0	1	1	0	1
7	0	0	1	1	1	1
8	0	1	0	0	0	1
9	0	1	0	0	1	1
10	0	1	0	1	0	1
11	0	1	0	1	1	1
12	0	1	1	0	0	1
13	0	1	1	0	1	1
14	0	1	1	1	0	0
15	0	1	1	1	1	1
16	1	0	0	0	0	0
17	1	0	0	0	1	0
18	1	0	0	1	0	1
19	1	0	0	1	1	1
20	1	0	1	0	0	1
21	1	0	1	0	1	1
22	1	0	1	1	0	1
23	1	0	1	1	1	1
24	1	1	0	0	0	1
25	1	1	0	0	1	1
26	1	1	0	1	0	1
27	1	1	0	1	1	1
28	1	1	1	0	0	0
29	1	1	1	0	1	1
30	1	1	1	1	0	0
31	1	1	1	1	1	0

(Tabla de verdad 2)

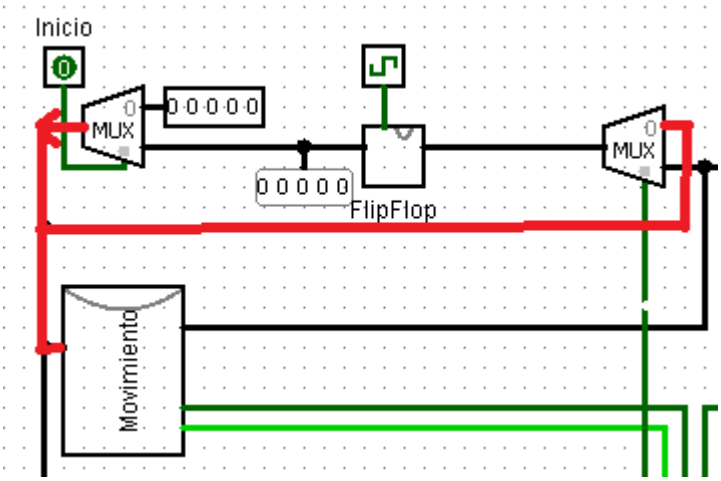
De **enable** obtenemos una señal que es dirigida a otro **multiplexor,** otro **InToDisplay** y a un decodificador 2:4 modificado cuyas utilidades se especifican más adelante.



(Figura 9: Recorrido señal enable)

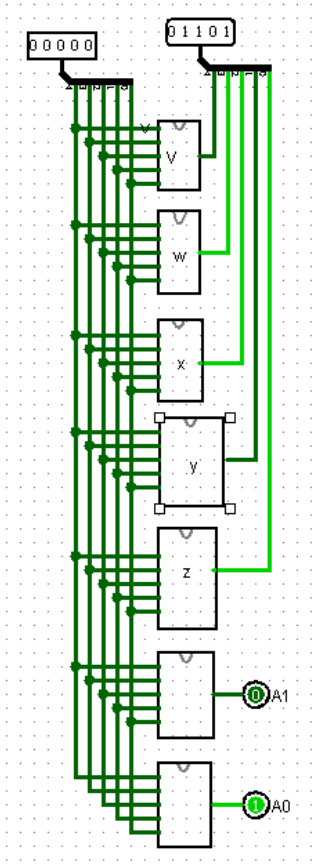
Esta señal únicamente está activa cuando el **número de inicio** es válido, si se ingresa otro número como el 00000 en este caso, la señal se cambiaría y se bloquearía el sistema, como veremos mas adelante.

El recorrido del **número de inicio** se desvía antes de ser la entrada del **enable** y el **InToDisplay** hacia dos circuitos mas, el **multiplexor 2** y el **control de movimiento.**



(Figura 10: Recorrido parcial del número inicial)

El **control de movimiento** solo cuenta con una señal de entrada (el **número inicial**) y tres de salida, una es un bus de 5 bits que contiene la **siguiente posición** y las otras dos son el equivalente en 2 bits del movimiento a realizar (adelante, derecha, izquierda, para)**:**



(Figura 11: Interior del control de movimiento)

En el control de movimiento nos encontramos con una cadena de subcircuitos hechos con las compuertas lógicas correspondientes a los mapas de Karnaugh formados a partir de la siguiente tabla de verdad

Orden	#In	Entrada					Salida					Acción		#Out
		i 4	i 3	i 2	i 1	i 0	s 4	s 3	s 2	s 1	s 0	a 1	a 0	
x	**0**	0	0	0	0	0	x	x	x	x	x	x	x	x
18	**1**	0	0	0	0	1	0	0	1	1	1	0	1	7
16	**2**	0	0	0	1	0	1	0	1	1	0	1	0	22
8	**3**	0	0	0	1	1	1	0	1	0	0	1	0	20
11	**4**	0	0	1	0	0	1	0	0	1	1	1	0	19
1	**5**	0	0	1	0	1	1	1	1	0	1	1	0	29
10	**6**	0	0	1	1	0	0	0	1	0	0	0	1	4
19	**7**	0	0	1	1	1	1	0	0	1	0	1	0	18
14	**8**	0	1	0	0	0	0	1	1	0	0	0	1	12
4	**9**	0	1	0	0	1	1	0	1	1	1	1	0	23
7	**10**	0	1	0	1	0	1	0	1	0	0	1	0	3
3	**11**	0	1	0	1	1	0	1	0	0	1	1	0	9
15	**12**	0	1	1	0	0	0	0	0	1	0	1	0	2
21	**13**	0	1	1	0	1	1	0	1	0	1	1	0	21
X	**14**	0	1	1	1	0	x	x	x	x	x	x	x	x
6	**15**	0	1	1	1	1	0	1	0	1	0	1	0	10
X	**16**	1	0	0	0	0	x	x	x	x	x	x	x	x
X	**17**	1	0	0	0	1	x	x	x	x	x	x	x	x
20	**18**	1	0	0	1	0	0	1	1	0	1	0	0	13
12	**19**	1	0	0	1	1	1	1	0	1	1	1	0	27
9	**20**	1	0	1	0	0	0	0	1	1	0	1	0	6
22	**21**	1	0	1	0	1	1	1	0	0	0	0	0	24
17	**22**	1	0	1	1	0	0	0	0	0	1	0	1	1
5	**23**	1	0	1	1	1	0	1	1	1	1	0	1	15
23	**24**	1	1	0	0	0	1	1	0	1	0	1	0	26
25	**25**	1	1	0	0	1	1	1	0	0	1	1	1	25
24	**26**	1	1	0	1	0	1	1	0	0	1	1	0	25
13	**27**	1	1	0	1	1	0	1	0	0	0	1	0	8
X	**28**	1	1	1	0	0	x	x	x	x	x	x	x	x
2	**29**	1	1	1	0	1	0	1	0	1	1	1	0	11
X	**30**	1	1	1	1	0	x	x	x	x	x	x	x	x
X	**31**	1	1	1	1	1	x	x	x	x	x	x	x	x

(Tabla de verdad 3)

Salida S4

wx / yz	00	01	11	10
00	x	1	0	0
01	0	1	1	1
11	1	1	0	0
10	1	0	x	1

V=0

wx / yz	00	01	11	10
00	x	0	x	1
01	x	1	0	1
11	1	0	x	0
10	0	0	x	1

V=1

Salida S3

wx / yz	00	01	11	10
00	x	0	0	1
01	0	1	0	0
11	0	0	1	1
10	0	0	x	0

V=0

wx / yz	00	01	11	10
00	x	0	x	1
01	x	1	0	1
11	1	1	x	1
10	1	0	x	1

V=1

F(v,w,x,y,z)=xz+wyz'+vw'z+vwz'

Salida S2

wx / yz	00	01	11	10
00	x	0	0	1
01	1	1	1	1
11	0	0	0	0
10	1	1	x	1

V=0

wx / yz	00	01	11	10
00	x	1	x	0
01	x	0	0	0
11	0	1	x	0
10	1	0	x	0

V=1

F(v,w,x,y,z)=w'x'z+w'x'y+vwx+v'w'xy'+v'wx'y'+vx'yz+vw'yz'

Salida S1

wx / yz	00	01	11	10
00	x	1	1	0
01	0	0	0	1
11	1	1	1	0
10	1	0	x	0

V=0

wx / yz	00	01	11	10
00	x	1	x	1
01	x	0	1	0
11	1	1	x	0
10	0	0	x	0

V=1

Salida S0

wx / yz	00	01	11	10
00	x	1	0	0
01	1	1	1	1
11	0	0	0	1
10	0	0	x	0
V=0				

wx / yz	00	01	11	10
00	x	0	x	0
01	x	0	1	1
11	1	1	x	0
10	1	1	x	1
V=1				

A1

wx / yz	00	01	11	10
00	x	1	1	0
01	0	1	1	1
11	1	1	1	1
10	1	0	x	1
V=0				

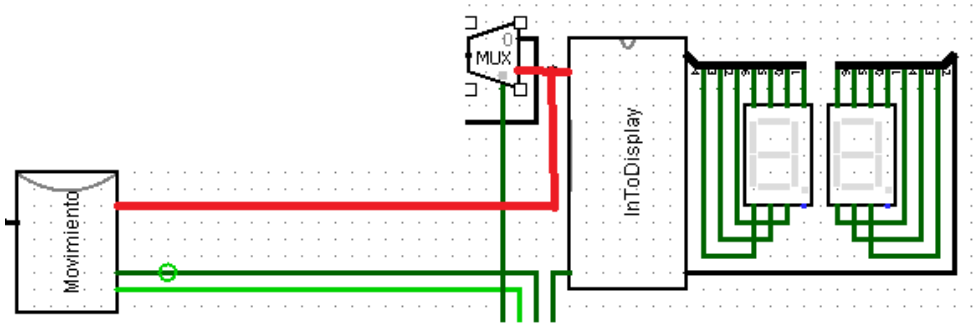
wx / yz	00	01	11	10
00	x	1	x	1
01	x	0	1	1
11	1	0	x	1
10	0	0	x	1
V=1				

A0

wx / yz	00	01	11	10
00	x	0	0	1
01	1	0	0	0
11	0	0	0	0
10	0	1	x	0
V=0				

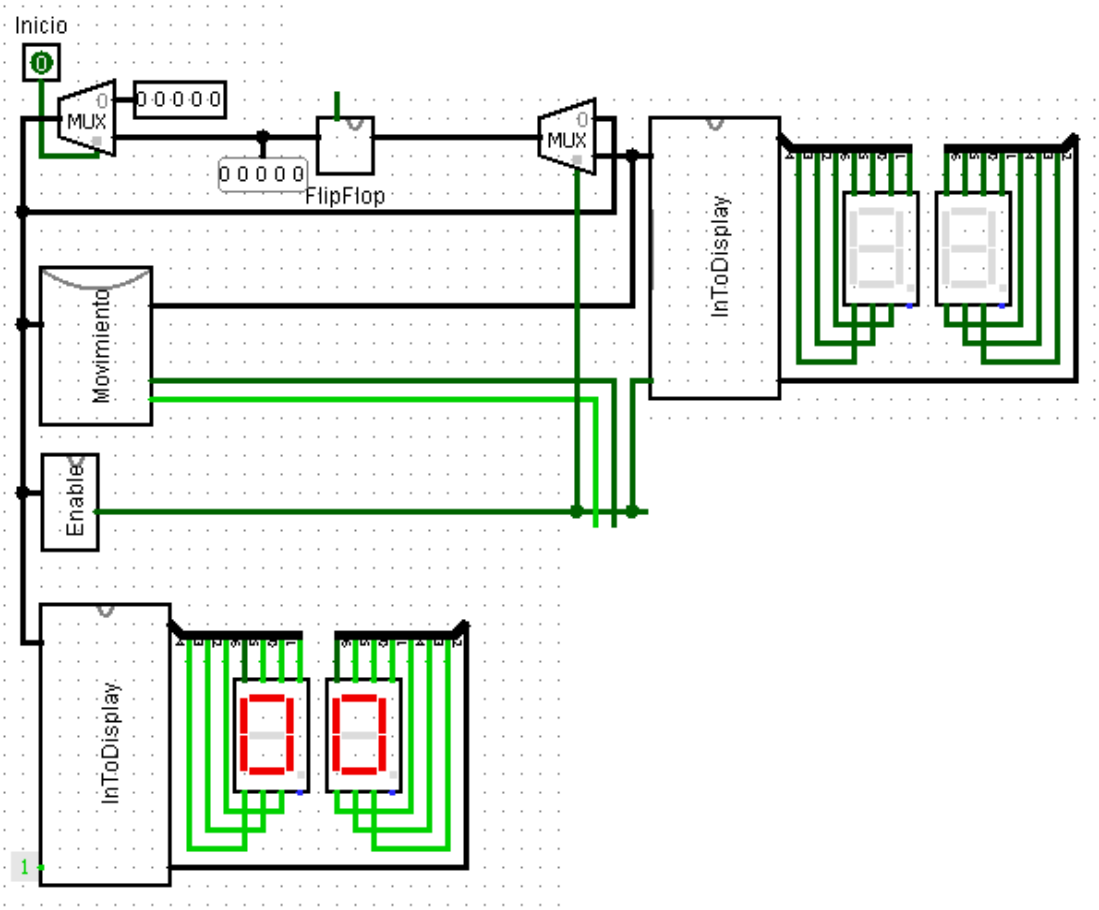
wx / yz	00	01	11	10
00	x	0	x	0
01	x	0	0	1
11	0	1	x	0
10	0	1	x	0
V=1				

La salida del **control de movimiento** referente a la **posición siguiente** se dirige a dos circuitos más, otro **InToDisplay** y al **multiplexor 2**:

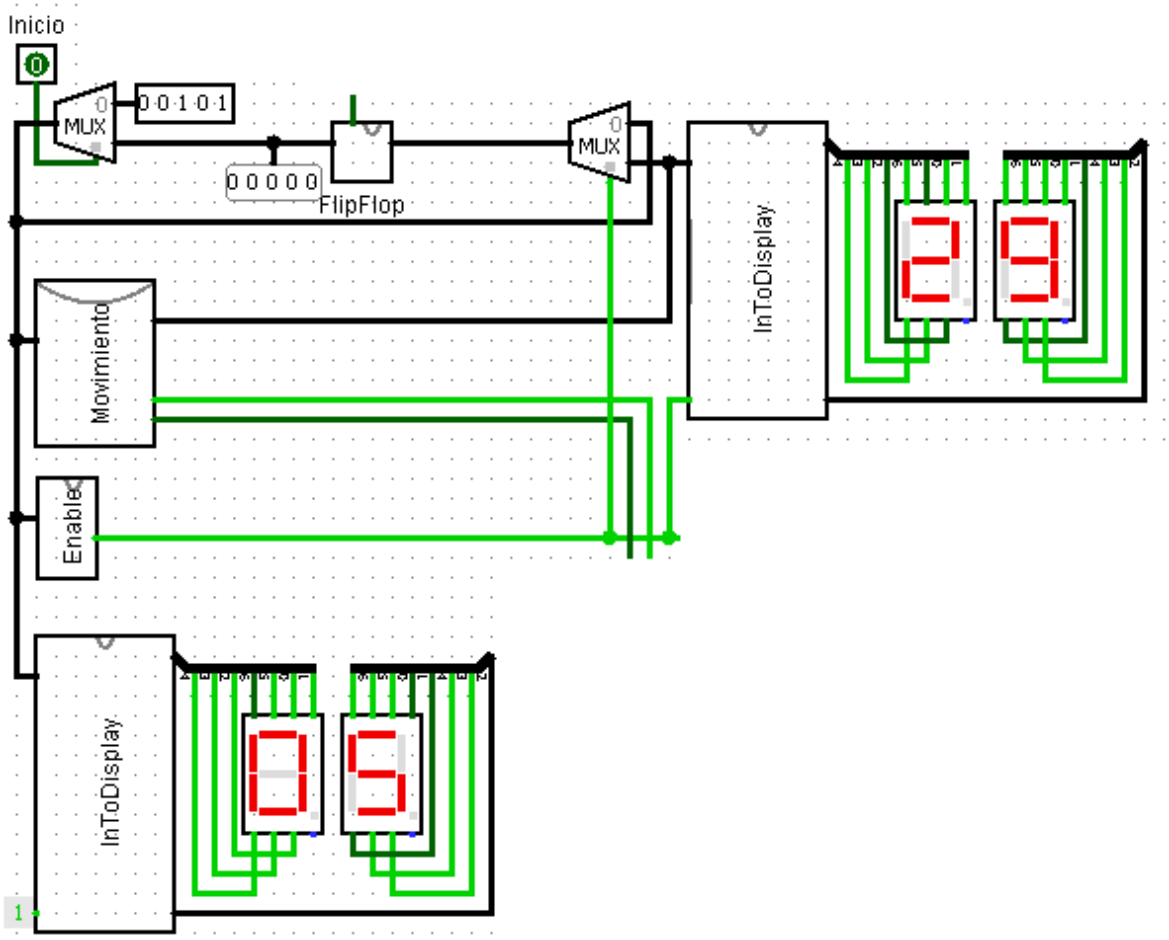


(Figura 12: Recorrido posición siguiente)

El segundo **InToDisplay** tiene la misma estructura y el mismo funcionamiento del enseñado previamente, sin embargo este si hace uso del **enable**, cuando el **número inicial** es un número inválido o fuera del recorrido del robot, el segundo display se apaga y no muestra nada; Aquí es donde entra el **multiplexor 2** siendo este quien regula el funcionamiento del circuito en automático teniendo como entradas dos buses, uno con el **número inicial** y otro con la **posición siguiente**, un seleccionador que es la salida del **enable** y llevando su salida a un **flip flop**, así entonces cuando se ingresa cualquier número que no esté en los parámetros del circuito completo el **enable** se desactiva (envía una señal "0") lo que hace que el **multiplexor 2** envíe el **número inicial** en bucle y deje de avanzar, mostrando en el display uno, la posición actual y manteniendo el segundo diaplay siempre apagado (**Figura 13.1**); Por el contrario cuando se ingresa un número valido como **número inicial** el **enable** se activa, permitiendo el paso de la **posición siguiente** en el **multiplexor 2** y el funcionamiento del segundo display, pudiendo así visualizar esta última(**Figura 13.2**).

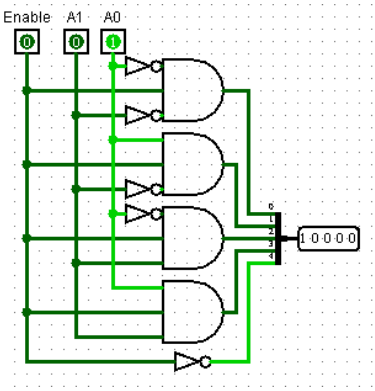

Inicio
0
MUX
0
00000
00000
FlipFlop
MUX
0
InToDisplay
Movimiento
Enable
InToDisplay
1

(Figura 13.1)

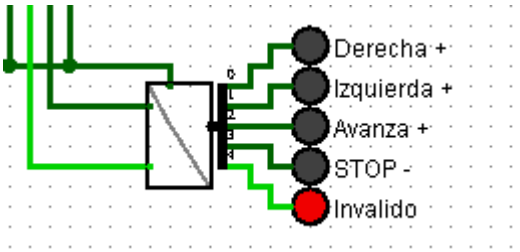


(Figura 13.2)

El **control de movimiento** también va conectado, usando sus dos salidas restantes (las correspondientes al **movimiento actual**) a un **display 3:5 con habilitador modificado** que haciendo uso del circuito **enable** puede encender, o bien cuatro posibles movimientos (avanza, derecha, izquierda, para) o una quinta salida que es la señal del **enable** invertida (inválido):

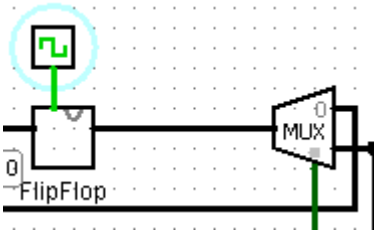


(Figura 14: arquitectura interna del display 3:5 modificado)



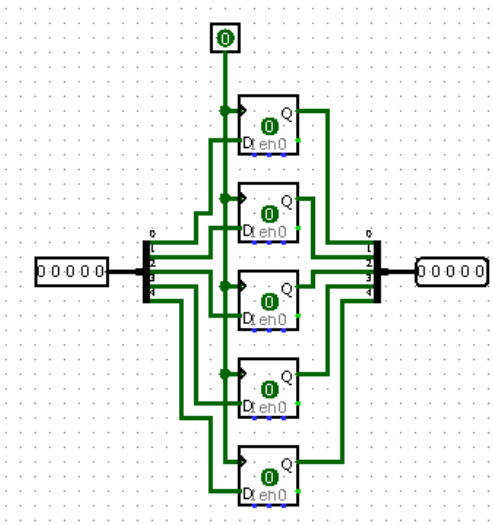
(Figura 15: Display 2:4 modificado (entrada: 00000))

Para finalizar el **flip flop** que recibe como entrada el bus de salida del **multiplexor 2** tiene también conectado un reloj, que se encarga de alternar la señal emitida automatizando casi completamente el proceso del circuito:



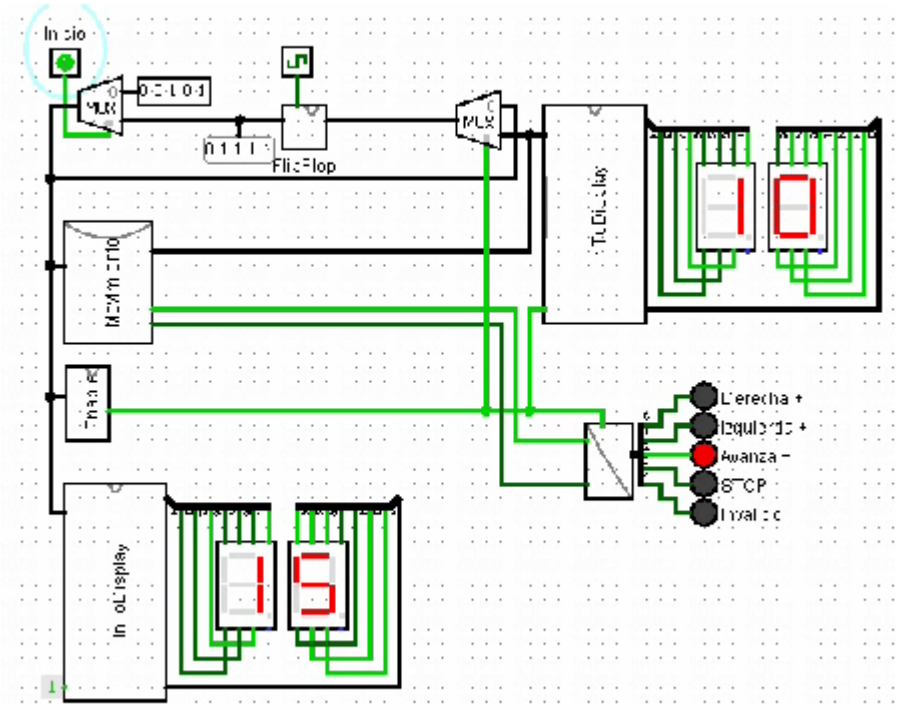
(Figura 16: flip flop)

El **flip flop** cumple la labor de guardar la información que recibe del **MUX 2** aunque esta cambie, hasta que el reloj envíe por segunda vez una señal equivalente a “1” que es cuando se actualiza a la nueva señal:



(Figura 17: Arquitectura interna del flip flop)

Como un detalle estético decidimos conectar a la salida del **flip flop** una herramienta que nos permitiera ver la escritura en binario del número saliente **control de movimiento.**



(Figura 18: Todo el sistema en funcionamiento)

Observaciones y análisis de resultados:

-El circuito completo tiene 3 modalidades de uso: Manual, semiautomática y automática.

Manual: Consiste en ingresar el código en binario de todos los números en el orden establecido por la normativa de la practica.

Semiautomático: Se basa básicamente en ingresar el número de arranque o número inicial, presionar el botón “inicio” y empezar a hacer click en el reloj para que oscile la señal que envía y empiece a recorrer y a mostrar solo todos los números correspondientes al camino que debe recorrer el robot.

Automático: Se inserta el número inicial, se presiona el botón “iniciar” y usando la combinación cntrl+k en el teclado el circuito recorrerá y mostrará totalmente solo el recorrido que debe hacer el robot.

-No hace falta que el número inicial sea el punto de inicio establecido por la practica, puede ponerse cualquier coordenada y el robot CONTINUARÁ su camino establecido, tomando la información ingresada en el panel como **número inicial** o **punto inicial**.

-Al ingresar números fuera de los parámetros el sistema no colapsa, reacciona tal y como fue programado, entra en un bucle (siendo ejecutado de manera semiautomática o automática).

-El circuito puede mostrar en el display cualquier número entero entre 0 y 31.

-El número en binario resultante del **control de movimiento** no siempre es el mismo reflejado en el **display 2**, las posiciones inválidas, por ejemplo.

Conclusiones:

Es importante destacar la importancia de diseñar circuitos modulares para minimizar la complejidad de estos y hacerlos escalables y compatibles con nuevos circuitos. El orden en este tipo de trabajos puede suponer un factor esencial que afectará directamente el tiempo de implementación que gaste el diseñador.

Esta practica fue de gran ayuda para comprender a un nivel “decente” el funcionamiento del simulador: “logisim”, a la vez que aplicamos los conceptos vistos en clase como el uso de mapas de karnaught para lograr crear circuitos basados en compuertas básicas como AND y OR.