

Informe

Laboratorio 1

Luis Ángel Vanegas Martínez

Problema #2

Diseño de Circuitos Combinacionales
Arquitectura de Computadores y Laboratorio

Fredy Alexander Rivera Vélez

Universidad de Antioquia

2021-1

Objetivo

- Emplear los conocimientos teóricos adquiridos en el curso en el proceso de diseño de sistemas digitales combinacionales.
- Emplear herramientas de software para el diseño y la simulación de sistemas digitales.

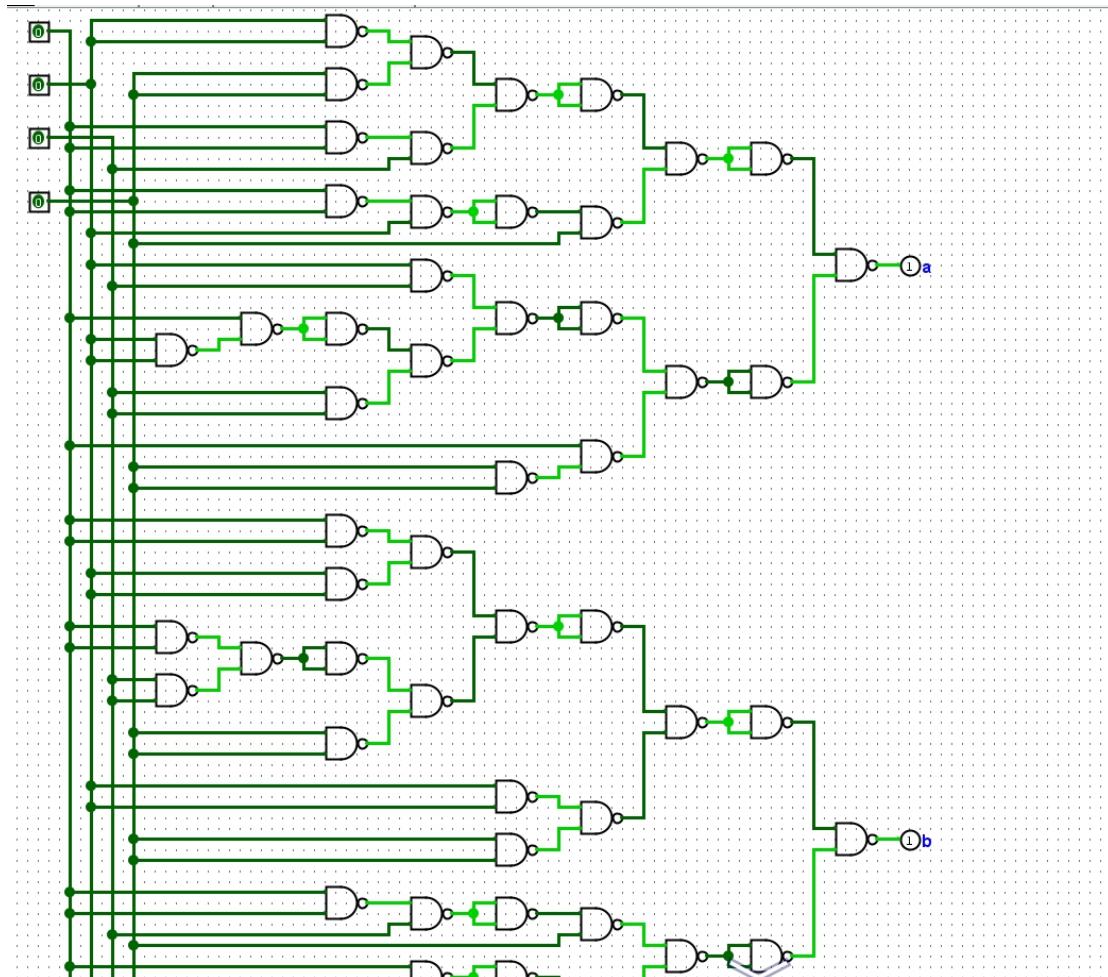
Proceso de Diseño

Sub Componente: Sub_Seven_Segment_Decoder

Se diseña el decodificador de valores de 4 bits a 7 segmentos por medio de un circuito combinacional por medio de compuertas NAND. A este circuito se le da el nombre de **Sub_Seven_Segment_Decoder** y hace parte del sub circuito diseñado para el laboratorio y que va inmerso en un componente mayor llamado **Binary_to_Two_Seven_Segment**.

Análisis

Se tiene como entrada los valores de 4 bits que luego son gestionados por compuertas lógicas para posteriormente dar una salida de 7 bits correspondientes a cada uno de los bits de los 7 segmentos. Para este se lleva a cabo un análisis combinacional que más adelante se describe.



Analisis Combinacional

(D,C,B,A): 4 bits de entrada

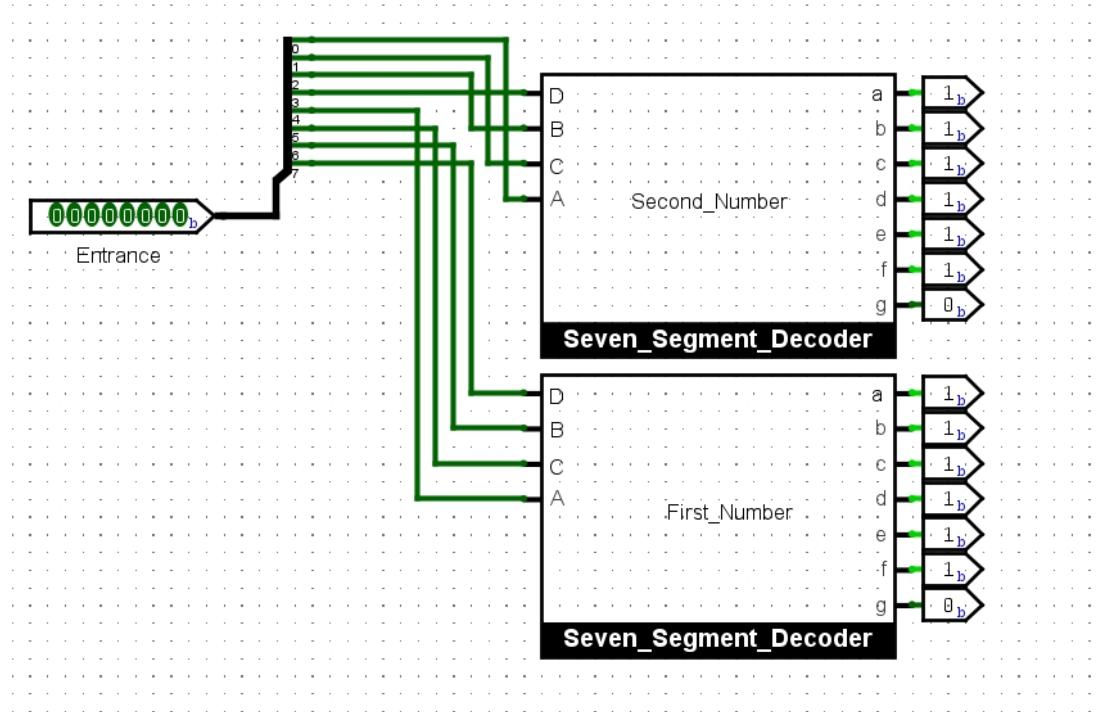
(a,b,c,d,e,f,g): 7 bits de salida que van directo al display de 7 segmentos

16 de 16 filas mostradas										
D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

Componente: Binary_to_Two_Seven_Segment

Se diseña el circuito que permitirá convertir una entrada binaria a un valor numérico decimal que se pueda visualizar en el display de 7 segmentos. Para este circuito se hace uso del sub circuito diseñado anteriormente de forma inclusiva e instanciada para que complemente la funcionalidad del presente circuito. Se usa el Seven_Segment_Decoder 2 veces para los 2 displays de 7 segmentos correspondientes a los 2 números que se mostrarán.

Análisis



Se tiene una entrada en base binaria de 8 bits, la cual se separa por medio de un separador de 8 bits de entrada y 8 bits de salida, de los cuales los 4 primeros bits hacen referencia al segundo número que luego será decodificado y mostrado de forma decimal en el display de 7 segmentos. Los 4 bits restantes de salida del separador, hacen referencia al primer número que luego será decodificado y mostrado de forma decimal en el display de 7 segmentos.

En el Seven Segment Decoder se evidencian los 4 bits de entradas y los 7 bits de salidas que conectarán directamente con el display de 7 segmentos. El Seven Segment Decoder es el mismo circuito diseñado anteriormente y que dimos por nombre:

Sub_Seven_Segment_Decoder

Análisis Combinacional

(entrance[7..0]) : equivale a los 8 bits de entrada

(aa,bb,cc,dd,ee,ff,gg): equivale a los 7 bits de salida del display 7 segmentos del número 2

(a,b,c,d,e,f,g): equivale a los 7 bits de salida del display 7 segmentos del número 1

256 de 256 filas mostradas														
entrance[7..0]	aa	bb	cc	dd	ee	ff	gg	a	b	c	d	e	f	g
0 0 0 0 0 0 0 0	1	1	1	1	1	1	0	1	1	1	1	1	1	0
0 0 0 0 0 0 0 1	0	1	1	0	0	0	0	1	1	1	1	1	1	0
0 0 0 0 0 0 1 0	1	1	0	1	1	0	1	1	1	1	1	1	1	0
0 0 0 0 0 0 1 1	1	1	1	1	0	0	1	1	1	1	1	1	1	0
0 0 0 0 0 1 0 0	0	1	1	0	0	1	1	1	1	1	1	1	1	0
0 0 0 0 0 1 0 1	1	0	1	1	0	1	1	1	1	1	1	1	1	0
0 0 0 0 0 1 1 0	1	0	1	1	1	1	1	1	1	1	1	1	1	0
0 0 0 0 0 1 1 1	1	1	1	0	0	0	0	1	1	1	1	1	1	0
0 0 0 0 1 0 0 0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0 0 0 0 1 0 0 1	1	1	1	0	0	1	1	1	1	1	1	1	1	0
0 0 0 0 1 0 1 0	1	1	1	0	1	1	1	1	1	1	1	1	1	0
0 0 0 0 1 0 1 1	0	0	1	1	1	1	1	1	1	1	1	1	1	0
0 0 0 0 1 1 0 0	1	0	0	1	1	1	0	1	1	1	1	1	1	0
0 0 0 0 1 1 0 1	0	1	1	1	1	0	1	1	1	1	1	1	1	0
0 0 0 0 1 1 1 0	1	0	0	1	1	1	1	1	1	1	1	1	1	0
0 0 0 0 1 1 1 1	1	0	0	0	1	1	1	1	1	1	1	1	1	0
0 0 0 1 0 0 0 0	1	1	1	1	1	1	0	0	1	1	0	0	0	0
0 0 0 1 0 0 0 1	0	1	1	0	0	0	0	0	1	1	0	0	0	0
0 0 0 1 0 0 1 0	1	1	0	1	1	0	1	0	1	1	0	0	0	0
0 0 0 1 0 0 1 1	1	1	1	1	0	0	1	0	1	1	0	0	0	0
0 0 0 1 0 1 0 0	0	1	1	0	0	1	1	0	1	1	0	0	0	0
0 0 0 1 0 1 0 1	1	0	1	1	0	1	1	0	1	1	0	0	0	0
0 0 0 1 0 1 1 0	1	0	1	1	1	1	1	0	1	1	0	0	0	0
0 0 0 1 0 1 1 1	1	1	1	0	0	0	0	0	1	1	0	0	0	0
0 0 0 1 1 0 0 0	1	1	1	1	1	1	1	0	1	1	0	0	0	0
0 0 0 1 1 0 0 1	1	1	1	0	0	1	1	0	1	1	0	0	0	0
0 0 0 1 1 0 1 0	1	1	1	0	1	1	1	0	1	1	0	0	0	0
0 0 0 1 1 0 1 1	0	0	1	1	1	1	1	0	1	1	0	0	0	0
0 0 0 1 1 1 0 0	1	0	0	1	1	1	0	0	1	1	0	0	0	0
0 0 0 1 1 1 0 1	0	1	1	1	1	0	1	0	1	1	0	0	0	0
0 0 0 1 1 1 1 0	1	0	0	1	1	1	1	0	1	1	0	0	0	0
0 0 0 1 1 1 1 1	1	0	0	0	1	1	1	0	1	1	0	0	0	0
0 0 1 0 0 0 0 0	1	1	1	1	1	1	0	1	1	0	1	0	1	1

Doble clic para ver el análisis completo:

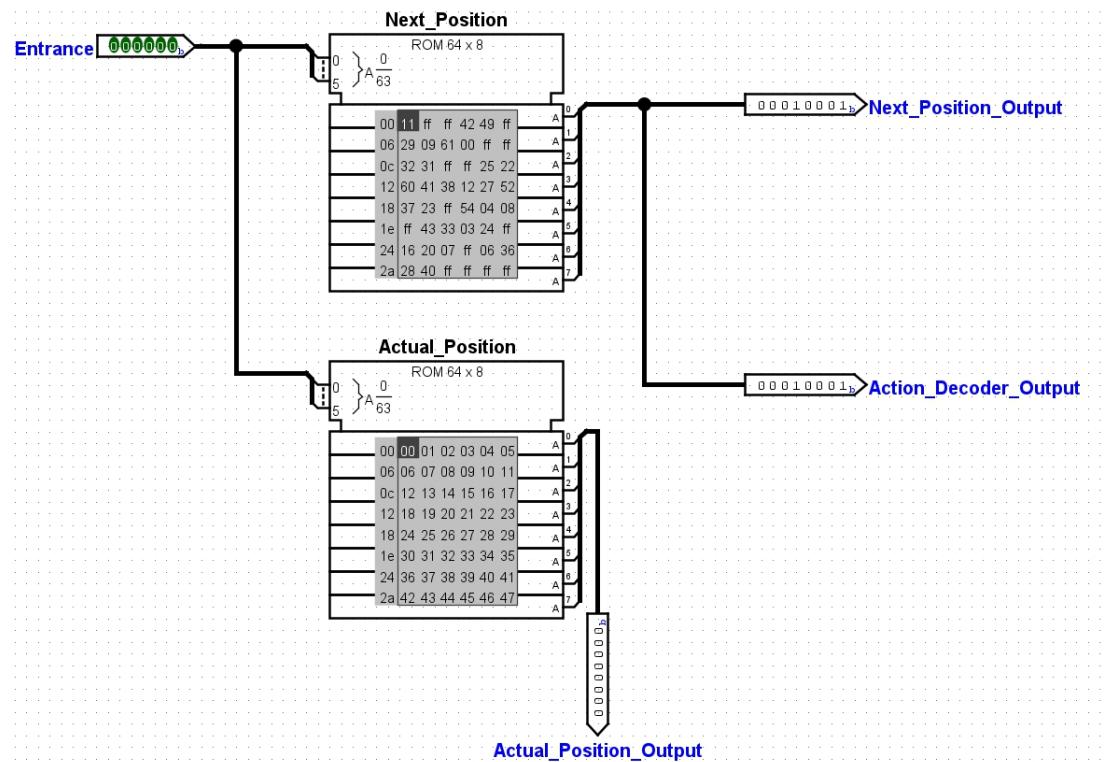


Binary_to_Two_Seven_Segment[analisis].txt

Component: Movement_Control

Componente encargado de generar la posición actual, la posición siguiente y la acción a realizar del robot de acuerdo a un número que entre en base binaria de 6 bits.

Análisis



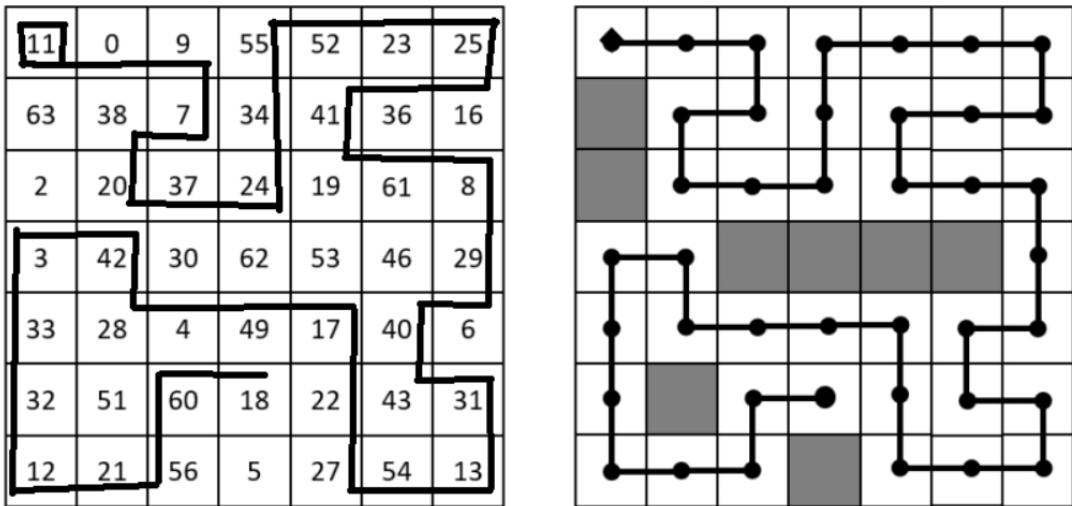
Se tiene una entrada binaria de 6 bits que conecta a 2 memorias ROM con entradas de 6 bits y salidas de 8 bits binarias. En la memorias ROM se mapean los respectivos valores de acuerdo a su posición. Para la ROM de posición actual se tienen los valores correctamente ordenados así:

00	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
10	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
20	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
30	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

Para la ROM de siguiente posición, se ubican los valores correspondientes a la siguiente posición de acuerdo a la posición actual.

00	11	ff	ff	42	49	ff	29	09	61	00	ff	ff	32	31	ff	ff
10	25	22	60	41	38	12	27	52	37	23	ff	54	04	08	ff	43
20	33	03	24	ff	16	20	07	ff	06	36	28	40	ff	ff	ff	ff
30	ff	17	ff	ff	55	ff	13	34	21	ff	ff	ff	56	19	ff	ff

Por ejemplo si la posición actual es 0, la siguiente sería 11 de acuerdo a la tabla anterior. La tabla se crea partiendo del siguiente problema:



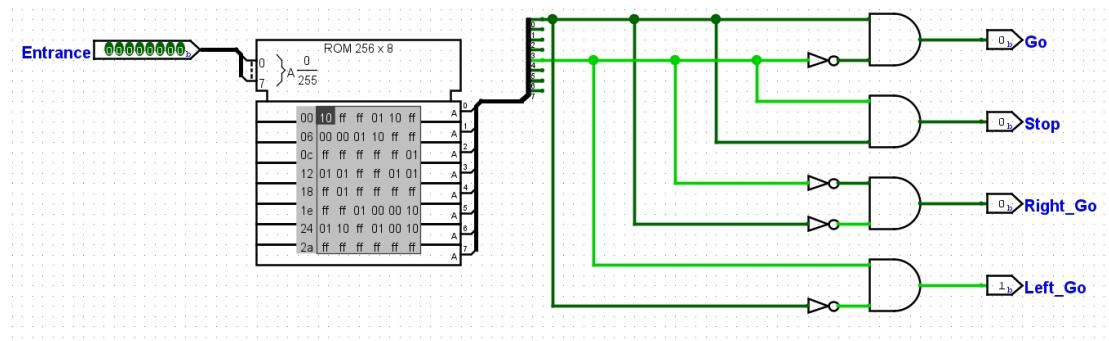
Los valores de FF son los que no se encuentran en la tabla o que están sombreados. Para el componente de Action_Decoder se usa la salida de 8 bits de la ROM de Next_Position.

Componente: Action_Decoder

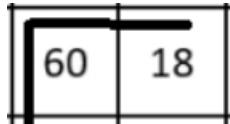
Este es el componente encargado de decodificar la acción siguiente que tiene que realizar el robot para ir a la siguiente posición, esta acción se lleva a cabo de acuerdo al siguiente código del problema #2:

Acción	Código
<i>Right + Go</i>	00
<i>Left + Go</i>	10
<i>Go</i>	01
<i>Stop</i>	11

Análisis



Se implementa una memoria ROM con valores correspondientes de 8x8, 8 bits de entrada y 8 Bits de salida. Los bits de entrada son provenientes del componente **Movement_Control**. Si el next_position es por ejemplo 60:

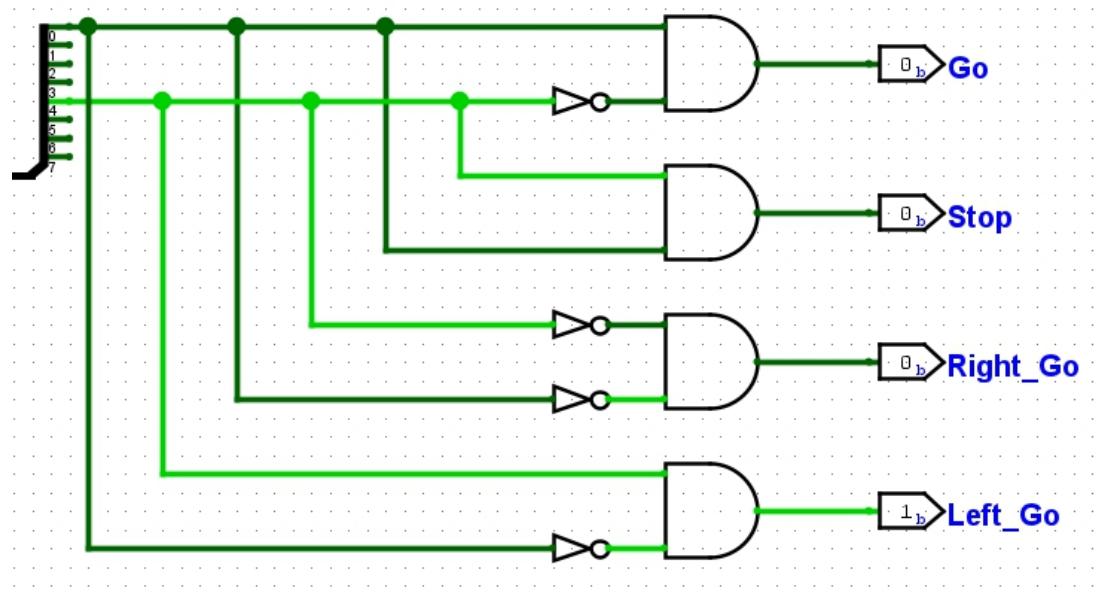


para ir de 18 a 60 debo de realizar un Go. Que según el código es el 01. entonces como dato de entrada llegará el 60 en binario, y el 60 en la memoria ROM tendrá valor de 01 así:

Logisim-evolution: Editor hexadecimal									
Archivo	Editar	Proyecto	Simular	FPGA	Ventana	Ayuda	-	□	×
00	10	ff	ff	01	10	ff	00	00	01
10	ff	01	01	01	ff	ff	01	01	ff
20	01	00	00	10	01	10	ff	01	00
30	ff	10	00	01	10	ff	00	00	ff
40	00	00	00	10	ff	ff	ff	01	ff
50	ff	ff	01	ff	10	01	10	ff	ff
60	01	10	ff	ff	ff	ff	ff	ff	ff
70	ff	ff	ff	ff	ff	ff	ff	ff	ff

Se usa separador de bits para separar los 8 bits de salidas de la ROM, adicional se usan los bits 0 y 3 del separador como datos de entradas del circuito combinacional.

Se usan puertas AND y NOT para que cumpla con la lógica de acción de acuerdo a los códigos propuestos.



Análisis Combinacional

(entrance[7..0]): equivale a los 8 bits de entrada

(Go, Stop, Right_Go, Left_Go): son cada una de las salidas del circuito y equivalen a 1 bit cada uno para que posteriormente sean conectadas a sus respectivos LED.

256 de 256 filas mostradas				
Entrance[7..0]	Go	Stop	Right_Go	Left_Go
0 0 0 0 0 0 0 0	0	0	0	1
0 0 0 0 0 0 0 1	0	1	0	0
0 0 0 0 0 0 1 0	0	1	0	0
0 0 0 0 0 0 1 1	1	0	0	0
0 0 0 0 0 1 0 0	0	0	0	1
0 0 0 0 0 1 0 1	0	1	0	0
0 0 0 0 0 1 1 0	0	0	1	0
0 0 0 0 0 1 1 1	0	0	1	0
0 0 0 0 1 0 0 0	1	0	0	0
0 0 0 0 1 0 0 1	0	0	0	1
0 0 0 0 1 0 1 0	0	1	0	0
0 0 0 0 1 0 1 1	0	1	0	0
0 0 0 0 1 1 0 0	0	1	0	0
0 0 0 0 1 1 0 1	0	1	0	0
0 0 0 0 1 1 1 0	0	1	0	0
0 0 0 0 1 1 1 1	0	1	0	0
0 0 0 1 0 0 0 0	0	1	0	0
0 0 0 1 0 0 0 1	1	0	0	0
0 0 0 1 0 0 1 0	1	0	0	0
0 0 0 1 0 0 1 1	1	0	0	0
0 0 0 1 0 1 0 0	0	1	0	0
0 0 0 1 0 1 0 1	0	1	0	0
0 0 0 1 0 1 1 0	1	0	0	0
0 0 0 1 0 1 1 1	1	0	0	0
0 0 0 1 1 0 0 0	0	1	0	0
0 0 0 1 1 0 0 1	1	0	0	0
0 0 0 1 1 0 1 0	0	1	0	0
0 0 0 1 1 0 1 1	0	1	0	0
0 0 0 1 1 1 0 0	0	1	0	0
0 0 0 1 1 1 0 1	0	1	0	0
0 0 0 1 1 1 1 0	0	1	0	0
0 0 0 1 1 1 1 1	0	1	0	0
0 0 1 0 0 0 0 0	1	0	0	0

Doble clic para ver el análisis completo:



Action_Decoder[Analysis].txt

Componente: Main

Este es el componente principal y dónde se integrarán los demás componentes mencionados anteriormente. Para el diseño de este componente se ha tenido en cuenta la arquitectura propuesta del circuito:

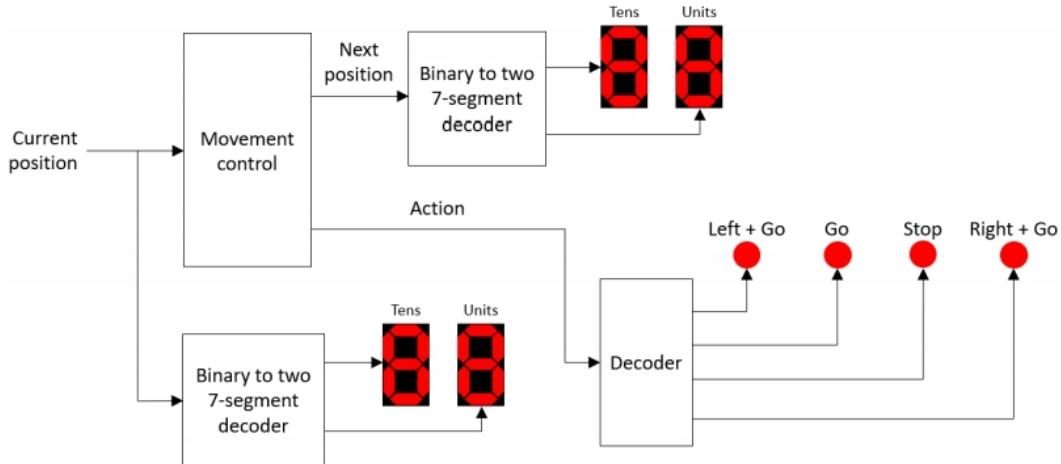
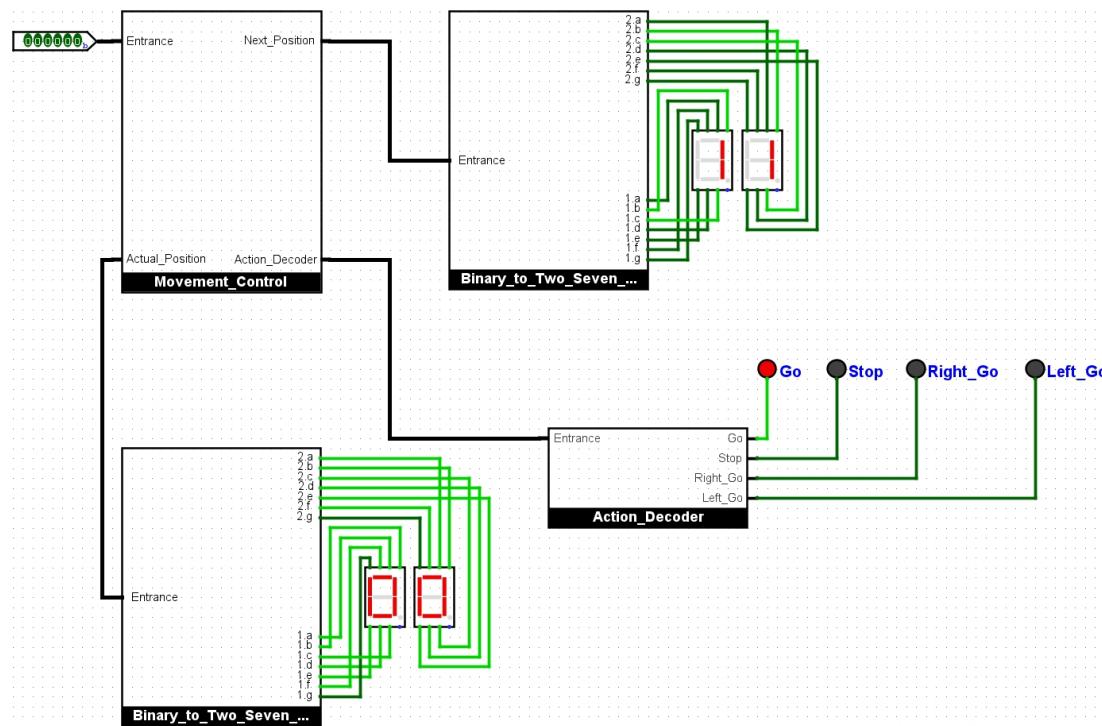


Figura 1. Arquitectura del sistema combinacional

Componente Main diseñado:



Análisis

Dicho circuito tiene una entrada binaria de 6 bits que conecta directamente al componente de **Movement_Control** y este en si conecta los 2 componentes de **Binary_to_Two_Seven_Segment** y **Action_Decoder**. Los cuales están conectados a displays

de 7 segmentos y Leds. La comunicación entre componentes se realiza en 8 bits y sus salidas son de a 1 bits.

Decisiones de Diseño

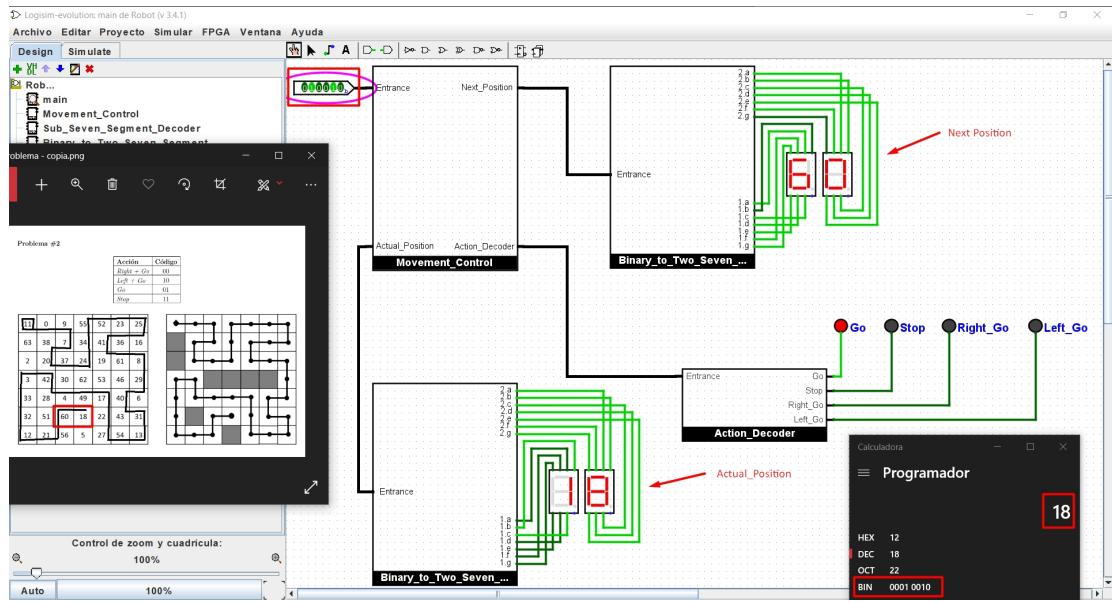
La posición de entrada es definida desde el componente de Movement_Control y no directamente desde la entrada, ya que por una fácil implementación los componentes de Binary_to_Two_Seven_Segment reciben entradas de 8 bits y no de 6, adicional se toma esta decisión debido a que la arquitectura interna del componente Movement_Control está diseñada para generar valores de 8 bits en decimal y no hexadecimal.

Se toma la iniciativa de usar memorias ROM para dar una solución ágil y limpia al problema planteado.

Simulación

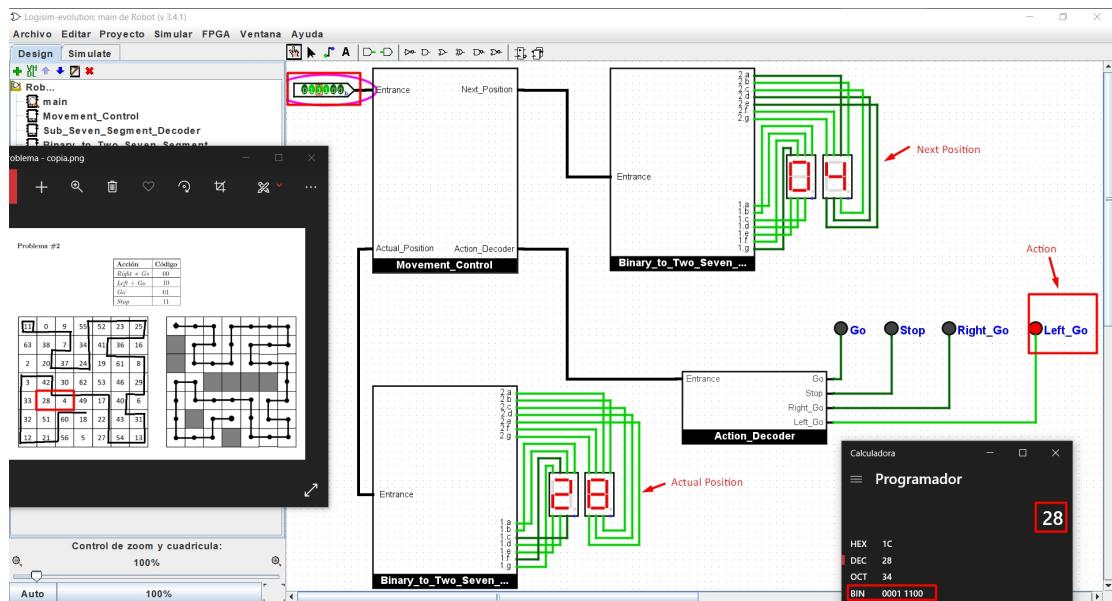
La simulación la haremos con base a distintos casos. Caso real, caso probable, caso extremos.

Caso real 1:



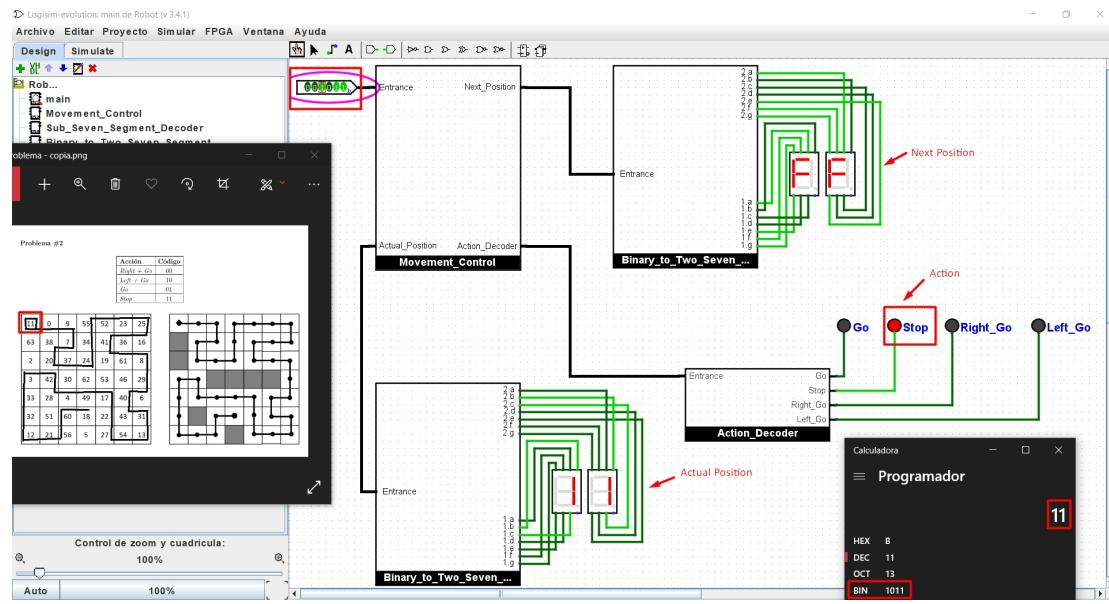
Visualizamos el número 18 en binario como: 010010 en la entrada y como posición actual, el LED indica que la acción es Go y que la siguiente posición es 60; de acuerdo al diagrama, es correcto.

Caso Real 2:



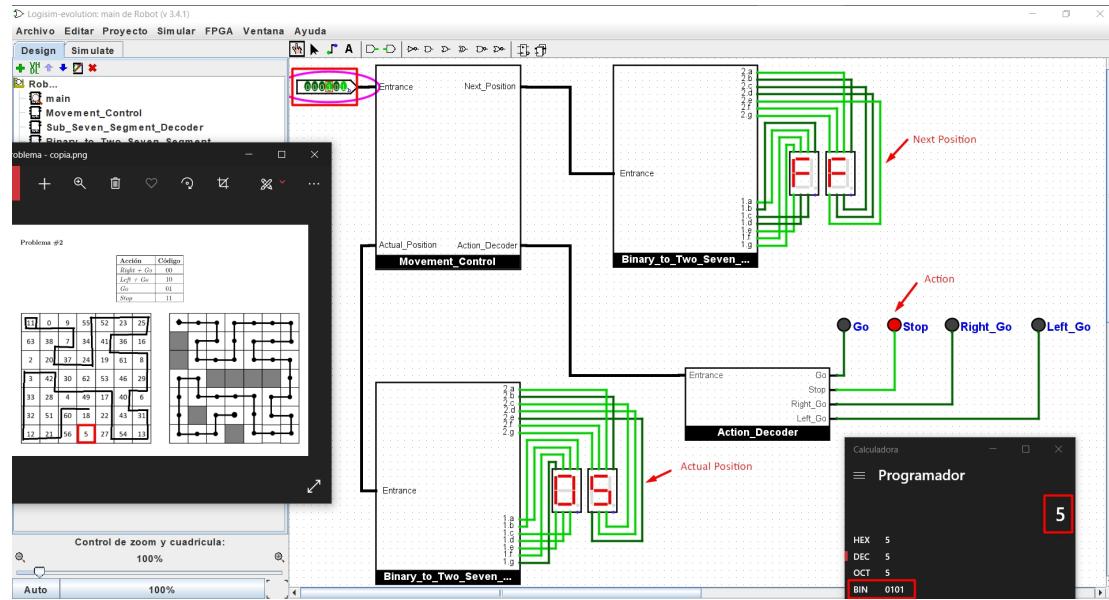
En este caso vemos que la posición actual es 28 denotada en binario como: 011100 de entrada, la acción a ejercer sería ir Left_Go para ir a la posición siguiente que es 04, la cual se compara con el diagrama y muestra que es correcto.

Caso Probable 1:



Este caso probable corresponde a que digiten como posición actual el 11. si vemos es el valor donde se quiere llegar y por consiguiente la acción a tomar es Stop y la siguiente posición es FF que corresponde a Fail.

Caso Extremo 1:

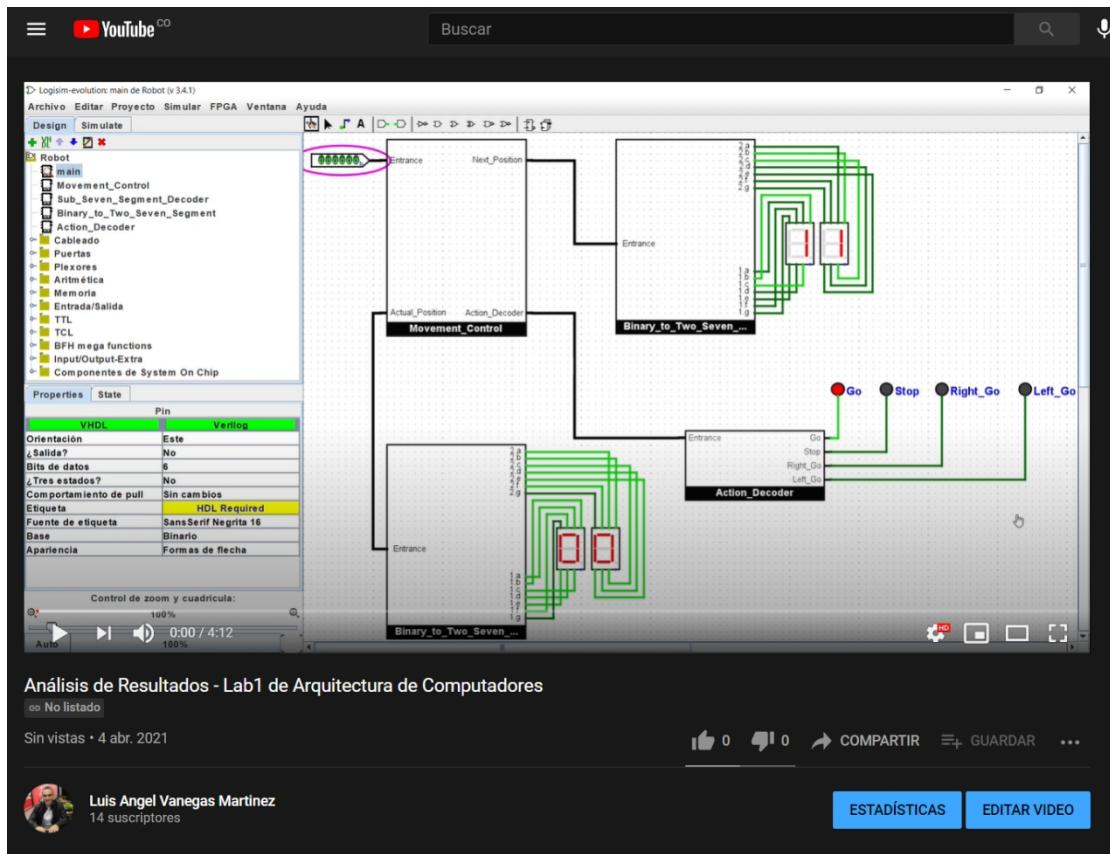


Este caso sucede cuando se pone una posición actual que no exista en la tabla o que este sombreada, mostrará el valor de entrada que se dio y la acción será Stop, adicional la siguiente posición es FF que corresponde a Fail.

Análisis de resultados

Para un mejor entendimiento planteamos el análisis de resultados en forma de video para que vean la interacción de una forma dinámica y orientada a un caso de uso real.

Clic aquí para ver el vídeo: <https://youtu.be/9FAYKp8MSqY>



Observaciones

El laboratorio está orientado a diferentes casos problemas, los cuales podrían aumentar en un futuro; la solución que he planteado tiene como objetivo suplir las necesidades de no sólo el problema #2 sino de cualquier otro problema propuesto, esto se haría sólo modificando los valores almacenados en la ROM del sistema y de resto, el resultado podría ser lo esperado.

Ejemplo: Si se requiere implementar la solución del problema #3 en el presente circuito digital, la arquitectura de este no se tocará; solo se modificará el archivo de memoria ROM con la nueva problemática y la respuesta será la esperada por el usuario.

Conclusión

Para concluir, se hace uso de puertas lógicas y diseño de circuitos combinacionales para los componentes de bajo nivel como lo son: el Sub_Seven_Segment_Decoder que se encarga de decodificar el valor binario de 4 bits que entra, en valores de 7 bits que pueden conectarse a un display de siete segmentos. También como el componente Action_Decoder el cual se encarga de traducir los valores numéricos de entrada en 8 bits a salidas de 1 bits que corresponde a acciones como Go, Left_Go, Stop, Right_Go.

Para los componentes de alto nivel usamos memorias ROM donde se mapean los valores de la tabla del problema propuesto, con entradas y salidas correspondientes a 8 bits y 6 bits (para las ROM de next_position y actual_position). Adicional hacemos usos de separadores de bits para que las salidas de los datos se puedan interpretar en bits unitarios y no en grupos de bits.

La solución esta pensada con el fin de favorecer una alta escalabilidad, con el fin de no amarrar el diseño a una solución sino crear un diseño que pueda satisfacer diferentes soluciones con tan solo un cambio que no sea de tan alto impacto y que interfiera con la arquitectura del circuito.

Repositorio del Trabajo: <https://github.com/xlavm/Logisim-Circuits>