# Informe

**Laboratorio 2**

Luis Ángel Vanegas Martinez

Único Problema

Diseño de Circuitos Secuenciales

Arquitectura de Computadores y Laboratorio

Fredy Alexánder Rivera Vélez

Universidad de Antioquia

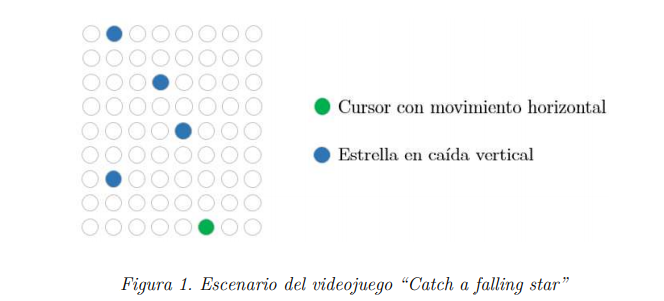
2021-1

# Objetivo

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

# Problema

Cada equipo de trabajo debe diseñar un sistema secuencial que implemente un sencillo videojuego denominado “Catch a falling star” en el que el jugador tiene la posibilidad de mover horizontalmente un cursor ubicado en la parte inferior del escenario, buscando atrapar “estrellas” que caen verticalmente desde la parte superior del mismo, tal como se muestra en la Figura 1. El sistema debe incluir un contador para ilustrar el número de estrellas atrapadas, y otro para mostrar el número de estrellas que se le han pasado por alto al jugador.



En el proceso se debe emplear un estilo de diseño estructural y jerárquico, en el que se diseñan bloques básicos que luego son instanciados para crear otros más complejos y de mayor nivel en la jerarquía de diseño. La herramienta a emplear en este laboratorio será Logisim Evolution v.3.4.1. Su biblioteca de componentes podrá ser empleada con total libertad, mientras el sistema se ajuste a los siguientes requerimientos:

• El control del movimiento del cursor debe ser realizado mediante un joystick.

• Por lo menos uno de los bloques secuenciales que hacen parte del sistema debe ser diseñado a partir de un modelo basado en máquinas de estados finitos. Para su implementación, el equipo de trabajo debe emplear flip flops JK si su identificador es par, o flip flops SR si su identificador es impar.

• El escenario en el que caen las estrellas debe construirse usando una matriz de LEDs, que debe tener unas dimensiones mínimas de 16×16 (filas × columnas). La fila en la que se mueve el cursor no está incluida en este escenario.

• El número de estrellas atrapadas por el jugador y el número de estrellas que se le han pasado por alto deben ser ilustrados mediante displays de siete segmentos.

• Una vez alcanzado un determinado número de estrellas que se le han pasado por alto al jugador, el juego debe detenerse. Para iniciar un nuevo juego será necesario reiniciar el sistema.

• El videojuego debe iniciar con una determinada velocidad de caída de las estrellas. A partir de cierto número de estrellas atrapadas, la velocidad de caída de las estrellas se debe incrementar.

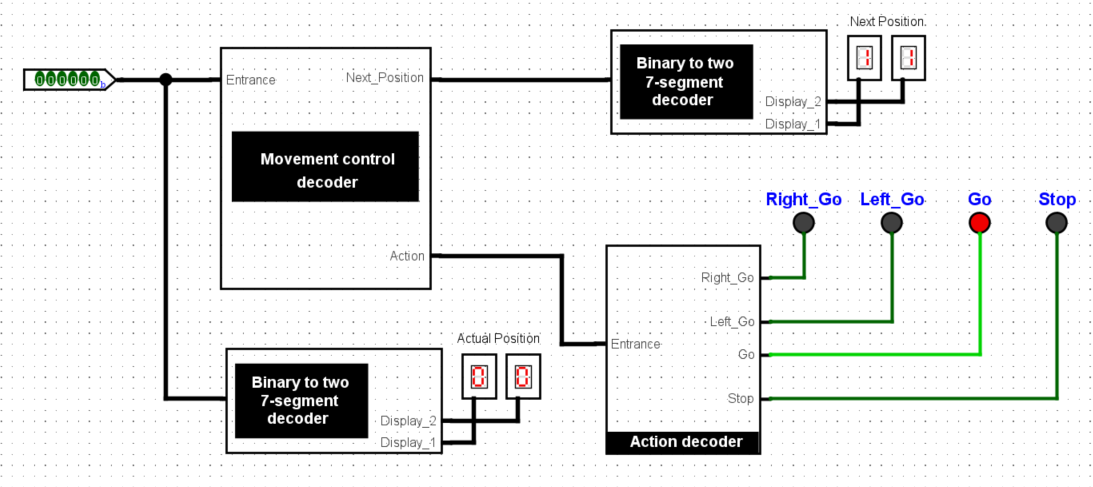
• Las estrellas que caen verticalmente deben ser generadas de manera pseudoaleatoria o, por lo menos, difícil de predecir.

El diseño de los diferentes módulos del sistema debe ser realizado de acuerdo con los procedimientos estudiados en clase. Este es un problema abierto en muchos sentidos, por lo que cada equipo recurrirá a su ingenio y creatividad para tomar decisiones de diseño.

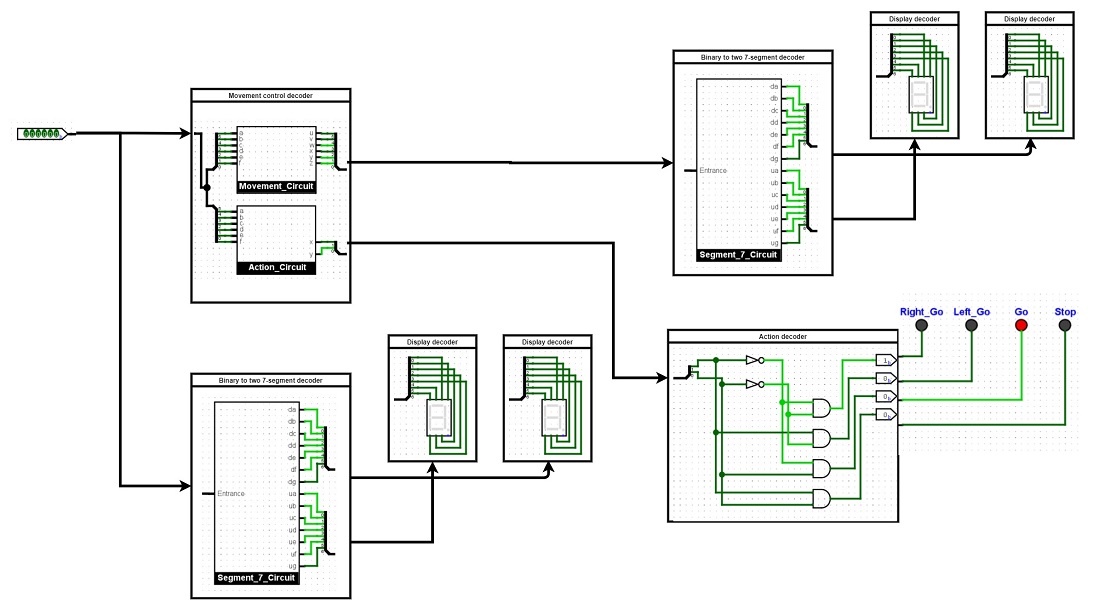
El sistema secuencial tiene que ser simulado exhaustivamente con la misma herramienta para verificar su correcto funcionamiento.

# Arquitecturas

## Arquitectura de la solución



## Arquitectura del Diseño de Circuito



# Decisiones de Diseño

Se ha tomado diferentes decisiones con respecto al diseño del circuito para llegar a una funcionalidad que se asemeje a lo que se expone en el problema.

1. Se decide crear un circuito combinacional convertir valores de 4 bits en valores de 7 bits correspondientes a 7 Segmentos, el cual dicho componente se usará para mostrar los valores del contador. Este componente se llama: **seven\_segment**.
2. Se usa separadores de bits para enlazar por medio de un único cable los diferentes componentes o decoders tal y como la arquitectura propuesta lo exige.
3. Se crea un componente Display\_decoder para instanciar los 4 displays requeridos en la solución y este se crea también teniendo en cuenta el punto 1 anterior.
4. Se crean componentes principales: **Movement control decoder**, **Binary to two 7-segment decoder**, **Action decoder.** Los cuales cada uno instancia sub componentes o circuitos combinacionales previamente diseñados.
5. De acuerdo al problema #2 que me correspondió, se toma el número mayor usado en este problema el cual es el: **63** que en binario corresponde a **111111** igual a **6 bits.** Partiendo de esto, decido que la entrada de datos del robot será de 6 bits.
6. Para tener una consistencia de datos, las entradas de los componentes: **Binary to two 7-segment decoder** y **Movement control decoder** corresponden a 6 bits incluyendo la entrada de los sub circuitos combinacionales que contienen estos.
7. El diseño de los sub circuitos combinacionales se hacen partiendo de tablas de verdad con entradas de 6 bits para todos sus circuitos, excepto para el circuito **Circuit\_Action\_Decoder** (que es el mismo componente **Action Decoder**) debido a que la entrada de este será en 2 bits.

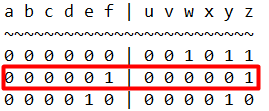
# Proceso de Diseño

Se inicia diseñando las tablas de verdad para cada sub circuito combinacional.

## - Circuit\_Movement

### Análisis

* Se crean entradas de 6 bits para los 64 números posibles a usar por el robot.
* Las entradas de los 64 números se crean en orden de 0 a 63, siendo cada entrada el número correspondiente en binario y la salida la posición siguiente que debería tomar el robot.
* Para los números que no están mapeados en la matriz del problema, como los son: (1,2,5,10,14,15,26,30,35,39,44,45,46,47,48,50,51,53,57,58,59,62,63,64) tendrán como salida el mismo número. Ejemplo del número 1:

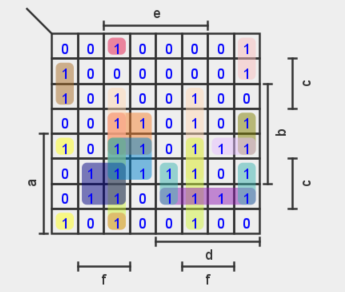


* De este circuito la entrada proviene del usuario y la salida va directa a los display que mostrarán la posición siguiente

### Tabla de Verdad

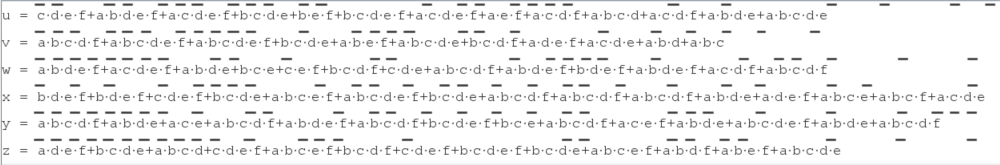
|  |
| --- |
| a b c d e f | u v w x y z  ~~~~~~~~~~~~~~~~~~~~~~~~~  0 0 0 0 0 0 | 0 0 1 0 1 1  0 0 0 0 0 1 | 0 0 0 0 0 1  0 0 0 0 1 0 | 0 0 0 0 1 0  0 0 0 0 1 1 | 1 0 1 0 1 0  0 0 0 1 0 0 | 1 1 0 0 0 1  0 0 0 1 0 1 | 0 0 0 1 0 1  0 0 0 1 1 0 | 0 1 1 1 0 1  0 0 0 1 1 1 | 0 0 1 0 0 1  0 0 1 0 0 0 | 1 1 1 1 0 1  0 0 1 0 0 1 | 0 0 0 0 0 0  0 0 1 0 1 0 | 0 0 1 0 1 0  0 0 1 0 1 1 | 0 0 1 0 1 1  0 0 1 1 0 0 | 1 0 0 0 0 0  0 0 1 1 0 1 | 0 1 1 1 1 1  0 0 1 1 1 0 | 0 0 1 1 1 0  0 0 1 1 1 1 | 0 0 1 1 1 1  0 1 0 0 0 0 | 0 1 1 0 0 1  0 1 0 0 0 1 | 0 1 0 1 1 0  0 1 0 0 1 0 | 1 1 1 1 0 0  0 1 0 0 1 1 | 1 0 1 0 0 1  0 1 0 1 0 0 | 1 0 0 1 1 0  0 1 0 1 0 1 | 0 0 1 1 0 0  0 1 0 1 1 0 | 0 1 1 0 1 1  0 1 0 1 1 1 | 1 1 0 1 0 0  0 1 1 0 0 0 | 1 0 0 1 0 1  0 1 1 0 0 1 | 0 1 0 1 1 1  0 1 1 0 1 0 | 0 1 1 0 1 0  0 1 1 0 1 1 | 1 1 0 1 1 0  0 1 1 1 0 0 | 0 0 0 1 0 0  0 1 1 1 0 1 | 0 0 1 0 0 0  0 1 1 1 1 0 | 0 1 1 1 1 0  0 1 1 1 1 1 | 1 0 1 0 1 1  1 0 0 0 0 0 | 1 0 0 0 0 1  1 0 0 0 0 1 | 0 0 0 0 1 1  1 0 0 0 1 0 | 0 1 1 0 0 0  1 0 0 0 1 1 | 1 0 0 0 1 1  1 0 0 1 0 0 | 0 1 0 0 0 0  1 0 0 1 0 1 | 0 1 0 1 0 0  1 0 0 1 1 0 | 0 0 0 1 1 1  1 0 0 1 1 1 | 1 0 0 1 1 1  1 0 1 0 0 0 | 0 0 0 1 1 0  1 0 1 0 0 1 | 1 0 0 1 0 0  1 0 1 0 1 0 | 0 1 1 1 0 0  1 0 1 0 1 1 | 1 0 1 0 0 0  1 0 1 1 0 0 | 1 0 1 1 0 0  1 0 1 1 0 1 | 1 0 1 1 0 1  1 0 1 1 1 0 | 1 0 1 1 1 0  1 0 1 1 1 1 | 1 0 1 1 1 1  1 1 0 0 0 0 | 1 1 0 0 0 0  1 1 0 0 0 1 | 0 1 0 0 0 1  1 1 0 0 1 0 | 1 1 0 0 1 0  1 1 0 0 1 1 | 1 1 0 0 1 1  1 1 0 1 0 0 | 1 1 0 1 1 1  1 1 0 1 0 1 | 1 1 0 1 0 1  1 1 0 1 1 0 | 0 0 1 1 0 1  1 1 0 1 1 1 | 1 0 0 0 1 0  1 1 1 0 0 0 | 0 1 0 1 0 1  1 1 1 0 0 1 | 1 1 1 0 0 1  1 1 1 0 1 0 | 1 1 1 0 1 0  1 1 1 0 1 1 | 1 1 1 0 1 1  1 1 1 1 0 0 | 1 1 1 0 0 0  1 1 1 1 0 1 | 0 1 0 0 1 1  1 1 1 1 1 0 | 1 1 1 1 1 0  1 1 1 1 1 1 | 1 1 1 1 1 1 |

### Expresión Minimizada

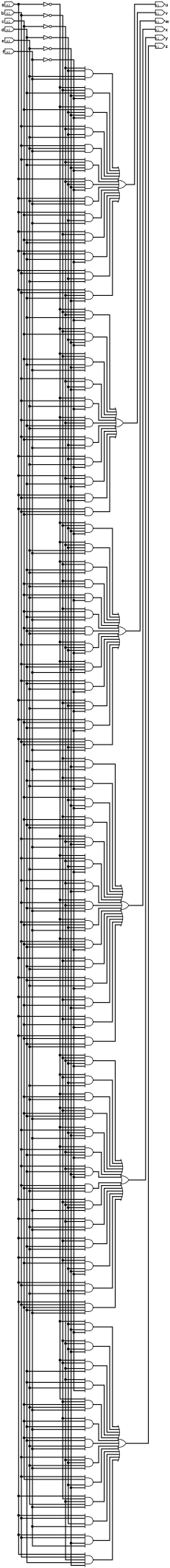




### Expresión de Salida



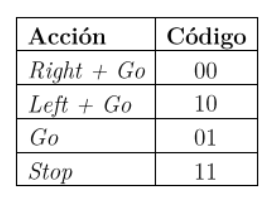
### Circuito



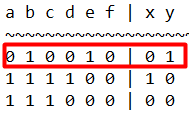
## - Circuit\_Action

### Análisis

* Se crean entradas de 6 bits para los 19 números que usará el robot para hacer el recorrido hasta la posición final.
* Las entradas de los 19 números se crean en orden desde 18 a 11 tal y como indica el camino del problema #2, siendo cada entrada el número correspondiente en binario y la salida el código correspondiente a la acción que ejecutará el robot de acuerdo a la posición donde se encuentra. Código de la acción:



Ejemplo de la primera acción a ejecutar por el robot (Go):

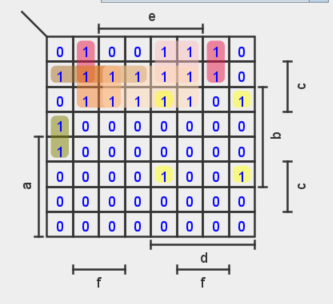


Desde el 18 ejecuta la acción con código 01 para ir al 60.

### Tabla de Verdad

|  |
| --- |
| a b c d e f | x y  ~~~~~~~~~~~~~  0 1 0 0 1 0 | 0 1  1 1 1 1 0 0 | 1 0  1 1 1 0 0 0 | 0 0  0 1 0 1 0 1 | 0 1  0 0 1 1 0 0 | 0 0  1 0 0 0 0 0 | 0 1  1 0 0 0 0 1 | 0 1  0 0 0 0 1 1 | 0 0  1 0 1 0 1 0 | 0 0  0 1 1 1 0 0 | 1 0  0 0 0 1 0 0 | 0 1  1 1 0 0 0 1 | 0 1  0 1 0 0 0 1 | 0 0  0 1 0 1 1 0 | 0 1  0 1 1 0 1 1 | 1 0  1 1 0 1 1 0 | 0 1  0 0 1 1 0 1 | 1 0  0 1 1 1 1 1 | 1 0  1 0 1 0 1 1 | 0 0  1 0 1 0 0 0 | 0 0  0 0 0 1 1 0 | 1 0  0 1 1 1 0 1 | 0 1  0 0 1 0 0 0 | 1 0  1 1 1 1 0 1 | 0 1  0 1 0 0 1 1 | 0 0  1 0 1 0 0 1 | 0 0  1 0 0 1 0 0 | 0 1  0 1 0 0 0 0 | 1 0  0 1 1 0 0 1 | 1 0  0 1 0 1 1 1 | 0 1  1 1 0 1 0 0 | 0 1  1 1 0 1 1 1 | 0 0  1 0 0 0 1 0 | 0 1  0 1 1 0 0 0 | 0 0  1 0 0 1 0 1 | 0 1  0 1 0 1 0 0 | 0 0  1 0 0 1 1 0 | 0 0  0 0 0 1 1 1 | 1 0  0 0 1 0 0 1 | 1 0  0 0 0 0 0 0 | 0 1  0 0 1 0 1 1 | 1 1 |

### Expresión Minimizada

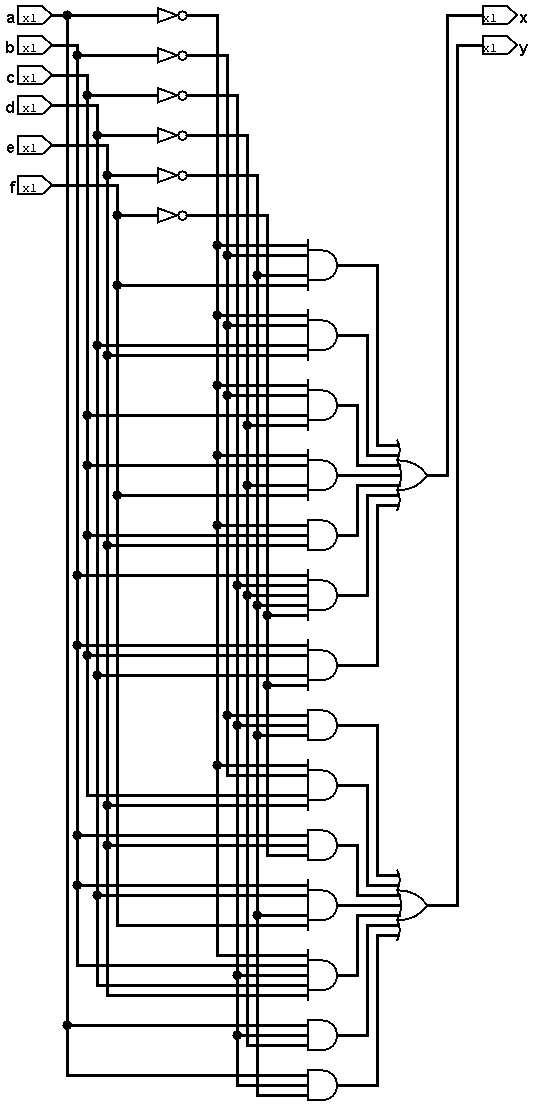




### Expresión de Salida



### Circuito



## - Action\_Decoder

### Análisis

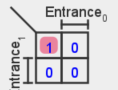
* Se crean entradas de 2 bits para los 4 códigos que usará el robot al ejecutar una acción.
* Las entradas son el código de la acción que ejecutará el robot y la salida es de 1 bit correspondiente al cable que llevará la energía al LED de acuerdo al código de la acción

### Tabla de Verdad

Entrada r: right\_go, Entrada l: left\_go, Entrada g: go, Entrada s: stop

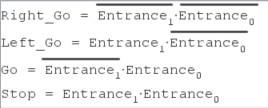
|  |
| --- |
| a b | r l g s  ~~~~~~~~~~  0 0 | 1 0 0 0  1 0 | 0 1 0 0  0 1 | 0 0 1 0  1 1 | 0 0 0 1 |

### Expresión Minimizada

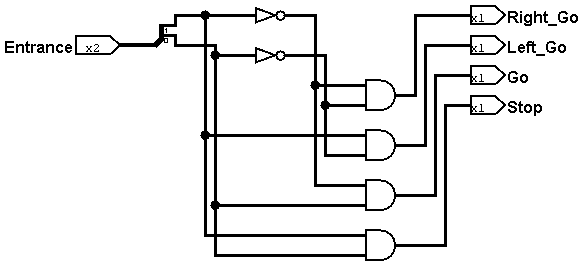




### Expresión de Salida



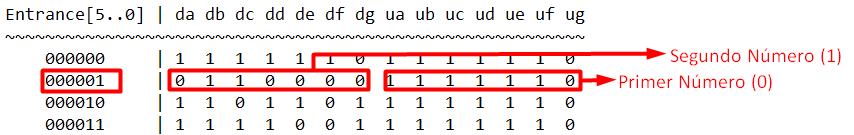
### Circuito



## - Circuit\_two\_7\_Segment

### Análisis

* Se crean entradas de 6 bits para los 64 números posibles a usar por el robot.
* Las entradas de los 64 números se crean en orden de 0 a 63, siendo cada entrada el número correspondiente en binario y las salidas son los bits correspondiente a cada segmento de los dos display.
* Un ejemplo para el número 1, habrán 2 display y los 7 primeros bits corresponden al segundo número y los 7 bits restantes al primer número, para mostrar en ambos display el 01



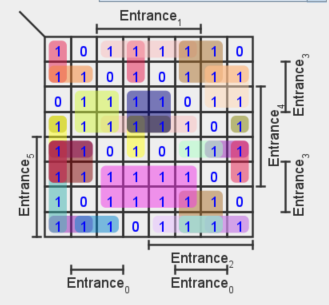
### Tabla de Verdad

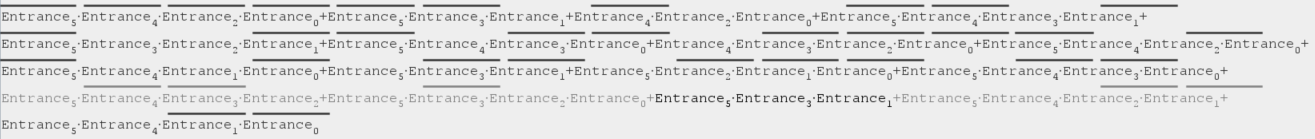
Las salidas (da,db,dc,dd,de,df,dg) corresponden a los bits del segundo 7 segmentos

Las salidas (ua,ub,uc,ud,ue,uf,ug) corresponden a los bits del primer 7 segmentos

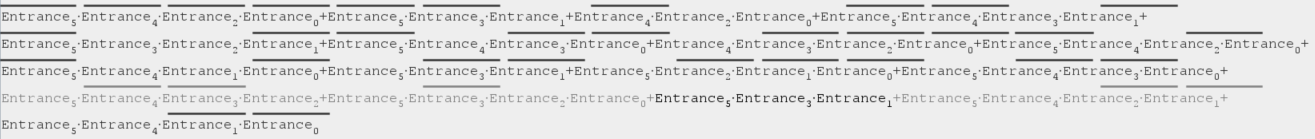
|  |
| --- |
| Entrance[5..0] | da db dc dd de df dg ua ub uc ud ue uf ug  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  000000 | 1 1 1 1 1 1 0 1 1 1 1 1 1 0  000001 | 0 1 1 0 0 0 0 1 1 1 1 1 1 0  000010 | 1 1 0 1 1 0 1 1 1 1 1 1 1 0  000011 | 1 1 1 1 0 0 1 1 1 1 1 1 1 0  000100 | 0 1 1 0 0 1 1 1 1 1 1 1 1 0  000101 | 1 0 1 1 0 1 1 1 1 1 1 1 1 0  000110 | 1 0 1 1 1 1 1 1 1 1 1 1 1 0  000111 | 1 1 1 0 0 0 0 1 1 1 1 1 1 0  001000 | 1 1 1 1 1 1 1 1 1 1 1 1 1 0  001001 | 1 1 1 0 0 1 1 1 1 1 1 1 1 0  001010 | 1 1 1 1 1 1 0 0 1 1 0 0 0 0  001011 | 0 1 1 0 0 0 0 0 1 1 0 0 0 0  001100 | 1 1 0 1 1 0 1 0 1 1 0 0 0 0  001101 | 1 1 1 1 0 0 1 0 1 1 0 0 0 0  001110 | 0 1 1 0 0 1 1 0 1 1 0 0 0 0  001111 | 1 0 1 1 0 1 1 0 1 1 0 0 0 0  010000 | 1 0 1 1 1 1 1 0 1 1 0 0 0 0  010001 | 1 1 1 0 0 0 0 0 1 1 0 0 0 0  010010 | 1 1 1 1 1 1 1 0 1 1 0 0 0 0  010011 | 1 1 1 0 0 1 1 0 1 1 0 0 0 0  010100 | 1 1 1 1 1 1 0 1 1 0 1 1 0 1  010101 | 0 1 1 0 0 0 0 1 1 0 1 1 0 1  010110 | 1 1 0 1 1 0 1 1 1 0 1 1 0 1  010111 | 1 1 1 1 0 0 1 1 1 0 1 1 0 1  011000 | 0 1 1 0 0 1 1 1 1 0 1 1 0 1  011001 | 1 0 1 1 0 1 1 1 1 0 1 1 0 1  011010 | 1 0 1 1 1 1 1 1 1 0 1 1 0 1  011011 | 1 1 1 0 0 0 0 1 1 0 1 1 0 1  011100 | 1 1 1 1 1 1 1 1 1 0 1 1 0 1  011101 | 1 1 1 0 0 1 1 1 1 0 1 1 0 1  011110 | 1 1 1 1 1 1 0 1 1 1 1 0 0 1  011111 | 0 1 1 0 0 0 0 1 1 1 1 0 0 1  100000 | 1 1 0 1 1 0 1 1 1 1 1 0 0 1  100001 | 1 1 1 1 0 0 1 1 1 1 1 0 0 1  100010 | 0 1 1 0 0 1 1 1 1 1 1 0 0 1  100011 | 1 0 1 1 0 1 1 1 1 1 1 0 0 1  100100 | 1 0 1 1 1 1 1 1 1 1 1 0 0 1  100101 | 1 1 1 0 0 0 0 1 1 1 1 0 0 1  100110 | 1 1 1 1 1 1 1 1 1 1 1 0 0 1  100111 | 1 1 1 0 0 1 1 1 1 1 1 0 0 1  101000 | 1 1 1 1 1 1 0 0 1 1 0 0 1 1  101001 | 0 1 1 0 0 0 0 0 1 1 0 0 1 1  101010 | 1 1 1 0 1 1 1 0 1 1 0 0 1 1  101011 | 1 1 1 1 0 0 1 0 1 1 0 0 1 1  101100 | 0 1 1 0 0 1 1 0 1 1 0 0 1 1  101101 | 1 0 1 1 0 1 1 0 1 1 0 0 1 1  101110 | 1 0 1 1 1 1 1 0 1 1 0 0 1 1  101111 | 1 1 1 0 0 0 0 0 1 1 0 0 1 1  110000 | 1 1 1 1 1 1 1 0 1 1 0 0 1 1  110001 | 1 1 1 0 0 1 1 0 1 1 0 0 1 1  110010 | 1 1 1 1 1 1 0 1 0 1 1 0 1 1  110011 | 0 1 1 0 0 0 0 1 0 1 1 0 1 1  110100 | 1 1 0 1 1 0 1 1 0 1 1 0 1 1  110101 | 1 1 1 1 0 0 1 1 0 1 1 0 1 1  110110 | 0 1 1 0 0 1 1 1 0 1 1 0 1 1  110111 | 1 0 1 1 0 1 1 1 0 1 1 0 1 1  111000 | 1 0 1 1 1 1 1 1 0 1 1 0 1 1  111001 | 1 1 1 0 0 0 0 1 0 1 1 0 1 1  111010 | 1 1 1 1 1 1 1 1 0 1 1 0 1 1  111011 | 1 1 1 0 0 1 1 1 0 1 1 0 1 1  111100 | 1 1 1 1 1 1 0 1 0 1 1 1 1 1  111101 | 0 1 1 0 0 0 0 1 0 1 1 1 1 1  111110 | 1 1 0 1 1 0 1 1 0 1 1 1 1 1  111111 | 1 1 1 1 0 0 1 1 0 1 1 1 1 1 |

### Expresión Minimizada

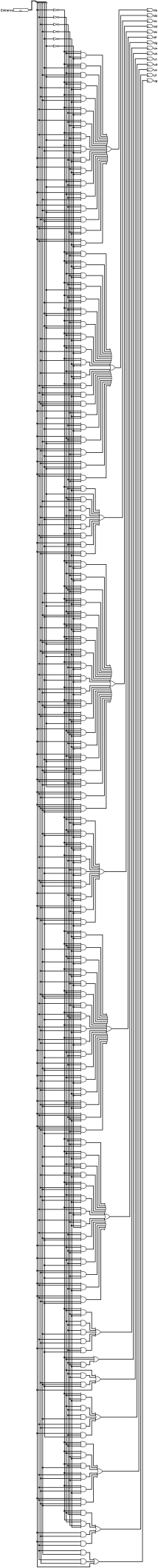




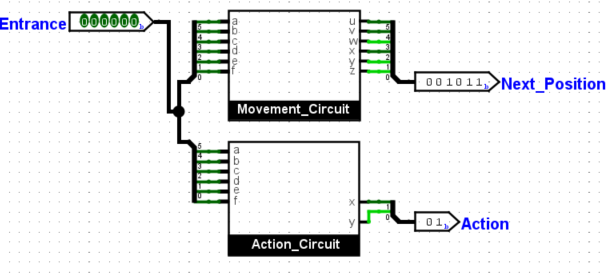
### Expresión de Salida



### Circuito



## Componente Movement\_Control\_Decoder

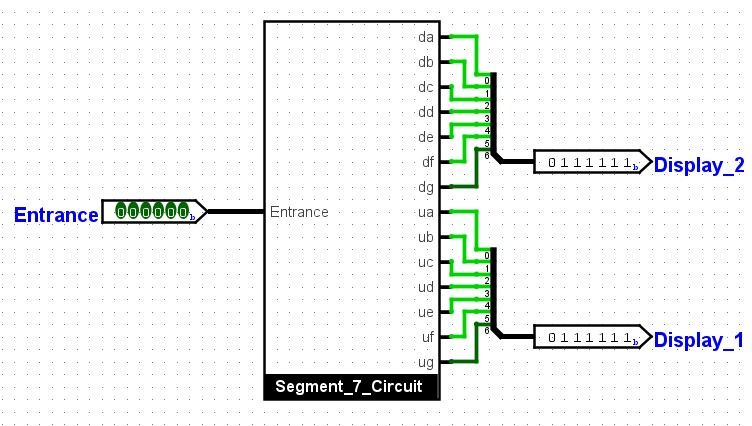


Este componente es creado partiendo de los circuitos **Action\_Circuit** y **Movement\_Circuit** Los cuales ambos tienen como entradas un número en base binaria de 6 bits.

**Action\_Circuit** El tiene una salida de 2 bits (como describo anteriormente) y este conecta directamente al componente Action\_Decoder.

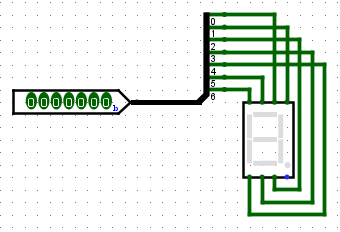
**Movement\_Circuit** El tiene una salida de 6 bits (como describo anteriormente) y este conecta directamente al componente Binary\_to\_two\_7\_Segment\_Decoder.

## Componente Binary\_to\_two\_7\_Segment\_Decoder



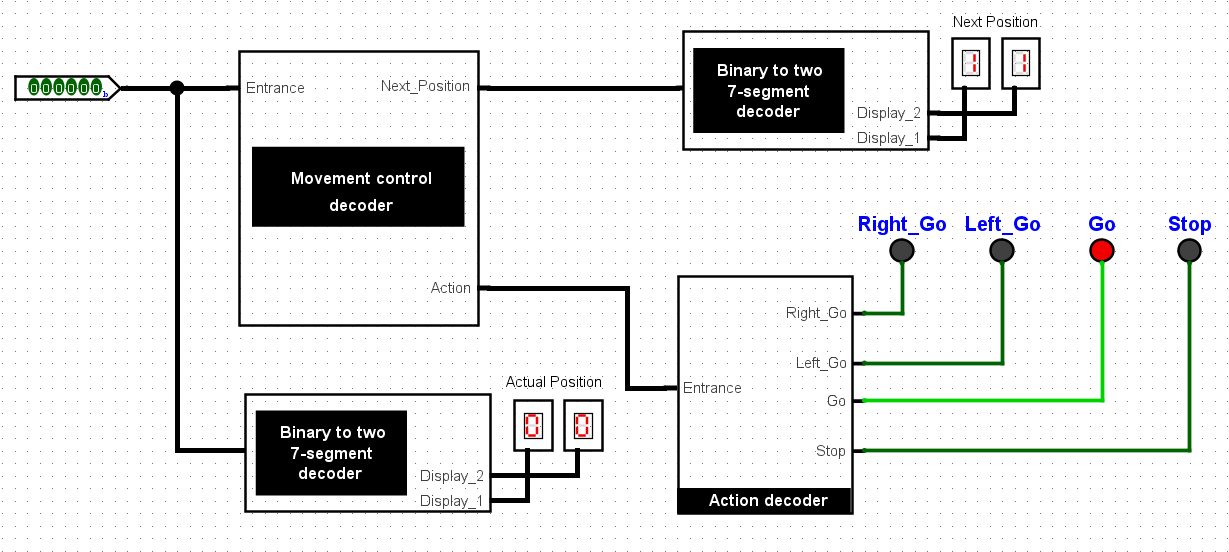
Este componente es creado partiendo del circuito **Circuit\_two\_7\_Segment** El cual tiene como entrada un número en base binaria de 6 bits y como salida 14 bits, de los cuales los 7 primeros corresponden al número binario expresado en decimal del número que va en el display 2 y los 7 bits restantes corresponden al número binario expresado en decimal del número que va en el display 1.

## Componente Display\_Decoder



Este componente es creado con el fin de incrustar el 7 segmentos de forma ordenada con un separador de bits de 7 y una entrada de 7bits para que posteriormente se conecte al decoder del número a mostrar.

## Componente Main



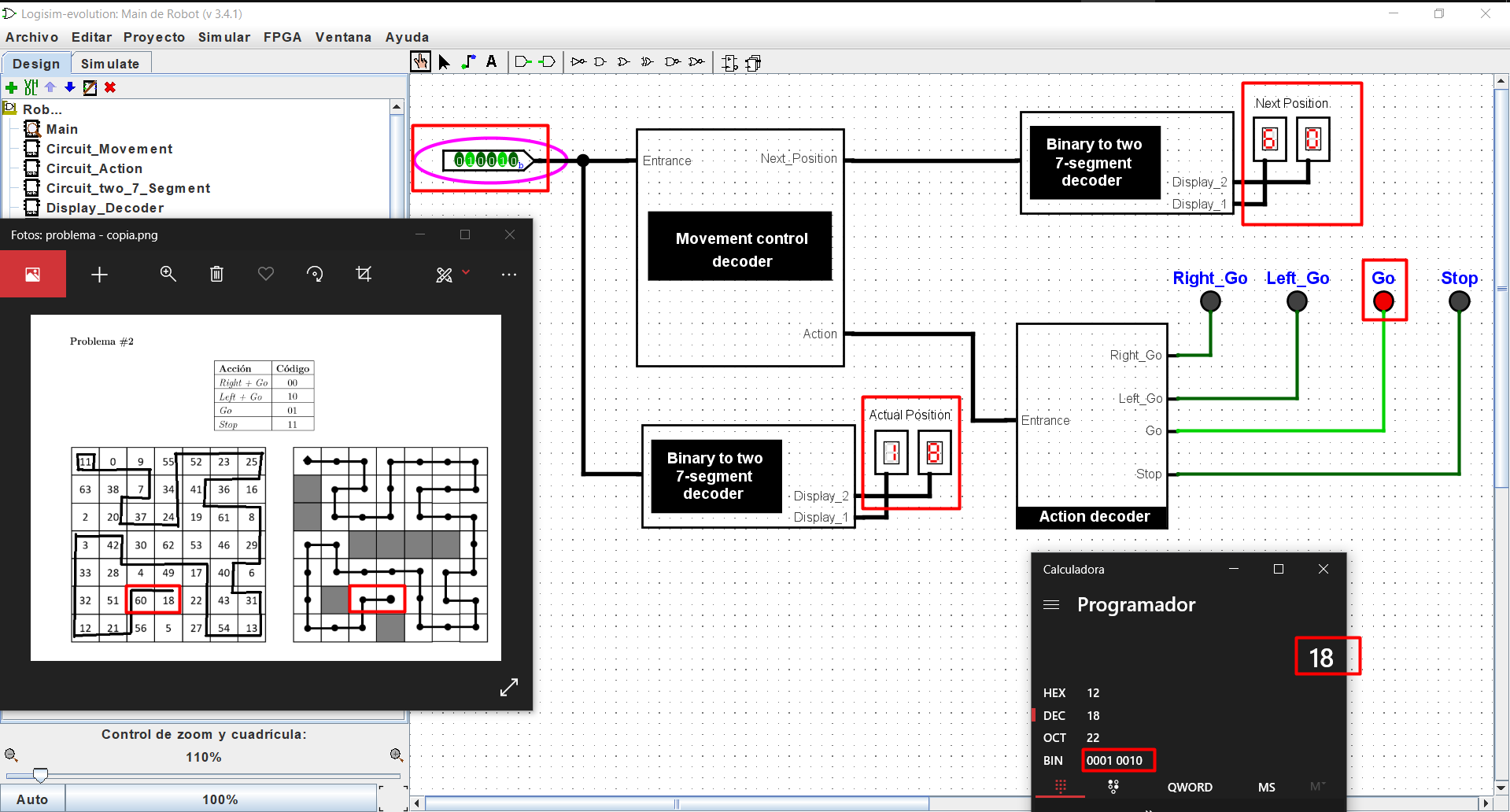
En este componente integro instanciando los demás componentes con el fin de que todos trabajen mancomunadamente para satisfacer las necesidades y problematicas propuestas, cumpliendo con la arquitectura exigida.

Cada vez que se digite un número entre 0 y 63 en base binaria, ese número viaja a binary to tow 7-segment decoder para mostrar él número ingresado en base decimal en el display de 7 segmentos, ese mismo número digitado viaja a Movement control decoder, dónde se obtendrá la acción a realizar y el valor de la siguiente posición. Luego en el Action decoder se toma el código de la acción a realizar y se decodifica en un bit que se encarga de encender el LED correspondiente a dicha acción.

# Simulación

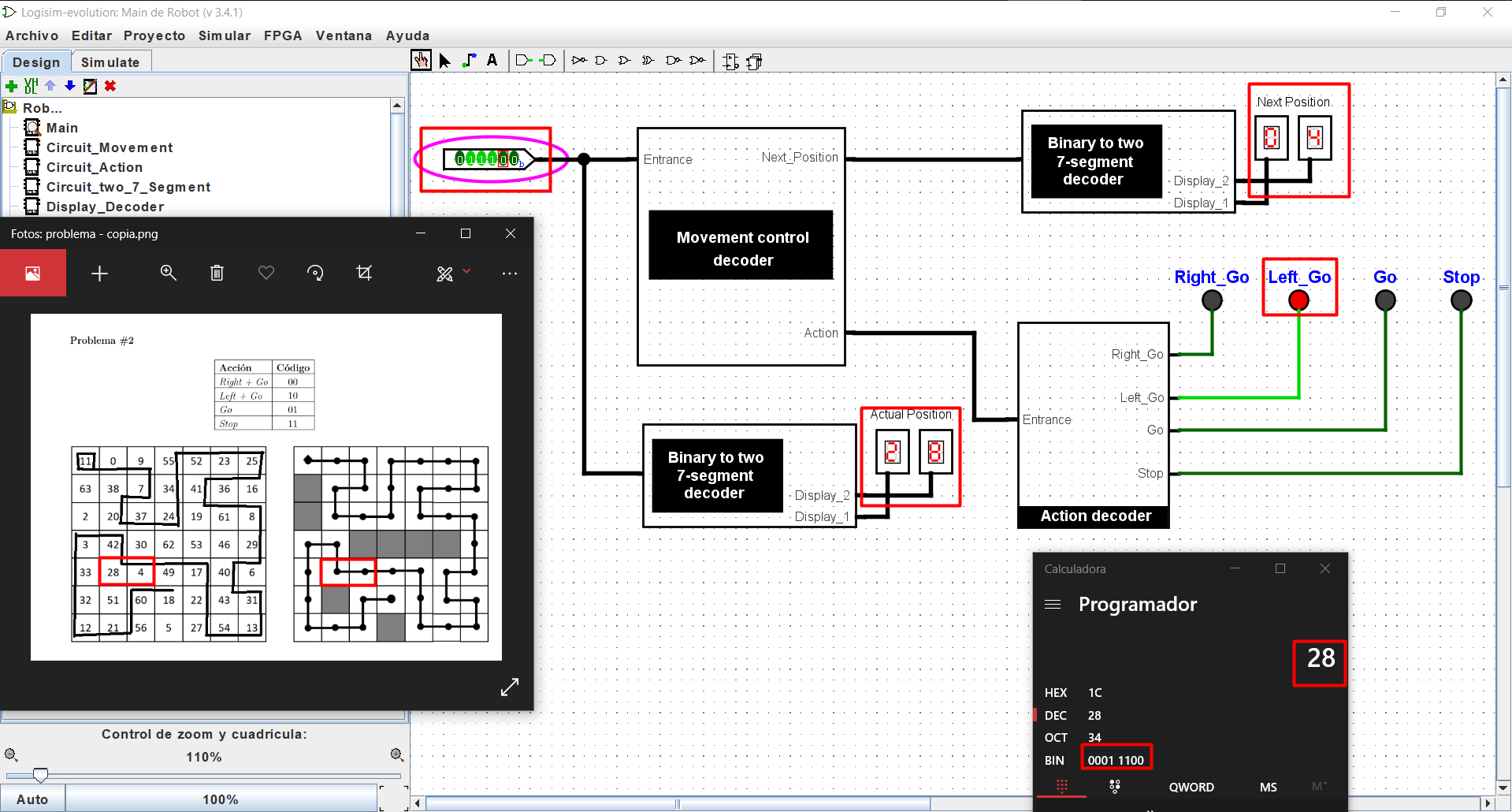
La simulación la haremos con base a distintos casos. Caso real, caso probable, caso extremo.

### Caso real 1:



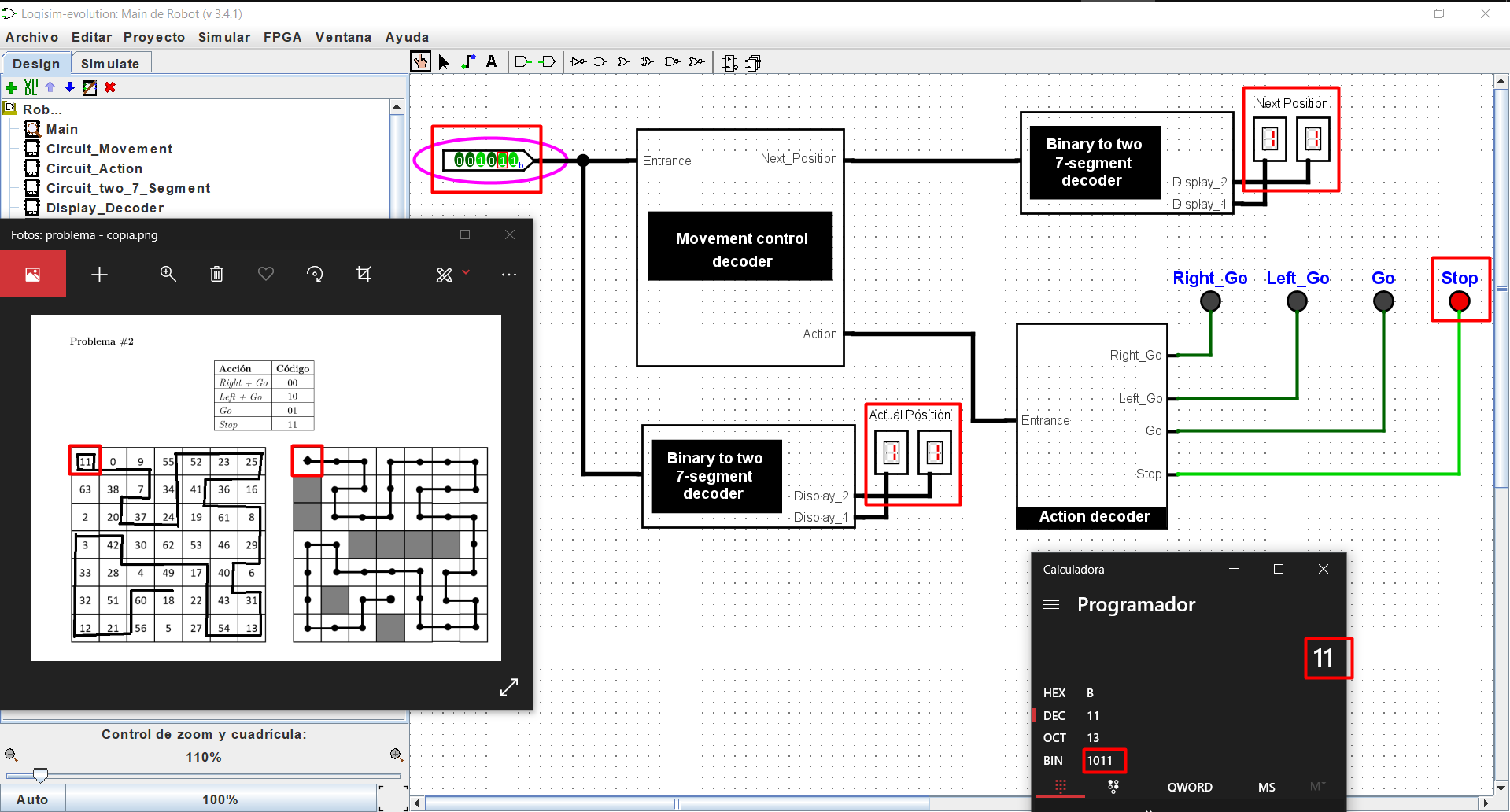
Visualizamos el número 18 en binario como: 010010 en la entrada y como posición actual, el LED indica que la acción es Go y que la siguiente posición es 60; de acuerdo al diagrama, es correcto.

### Caso Real 2:



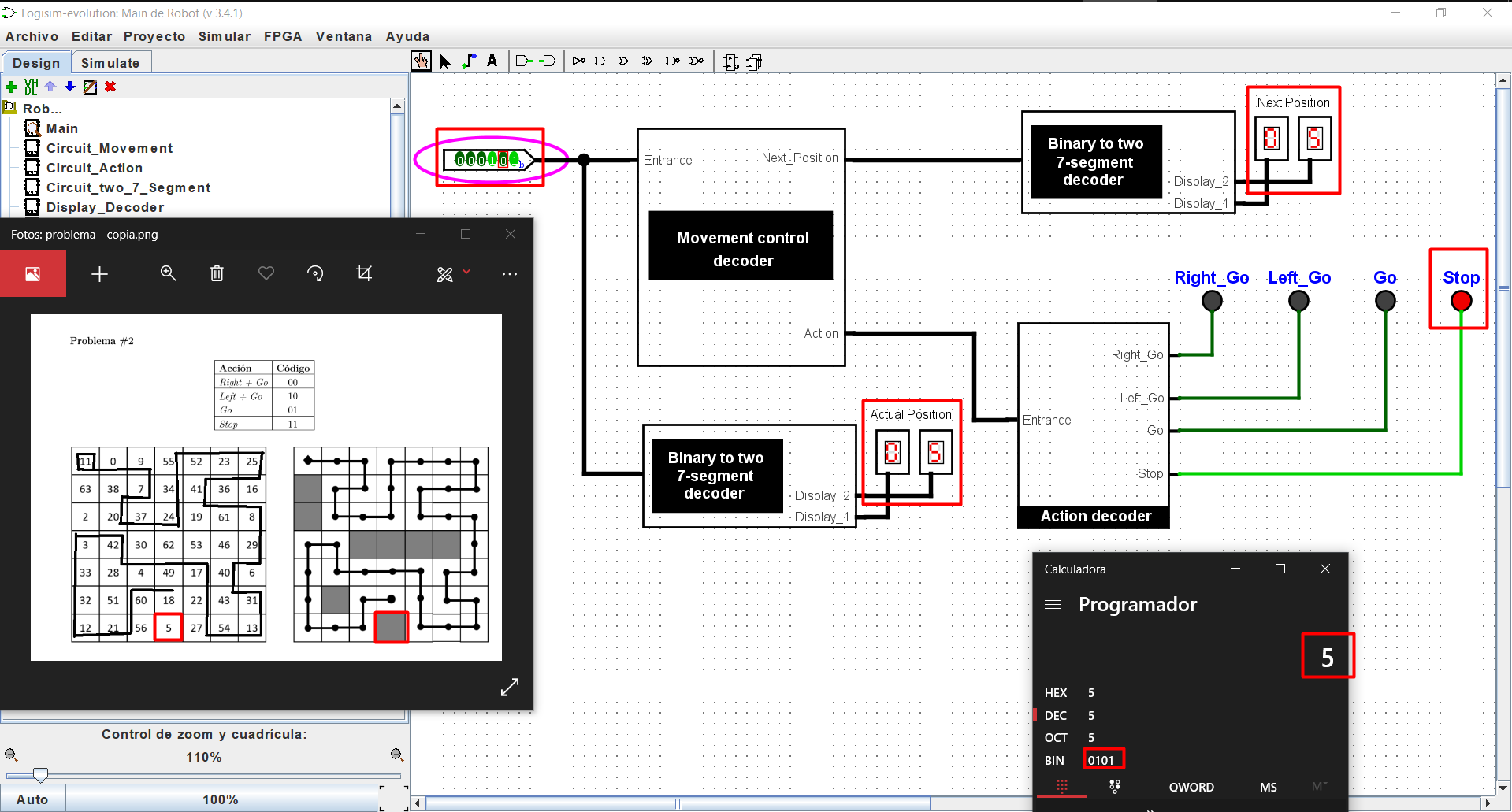
En este caso vemos que la posición actual es 28 denotada en binario como: 011100 de entrada, la acción a ejercer sería ir Left\_Go para ir a la posición siguiente que es 04, la cual se compara con el diagrama y muestra que es correcto.

### 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 el mismo número 11 porque no puede avanzar.

### Caso Extremo 1:

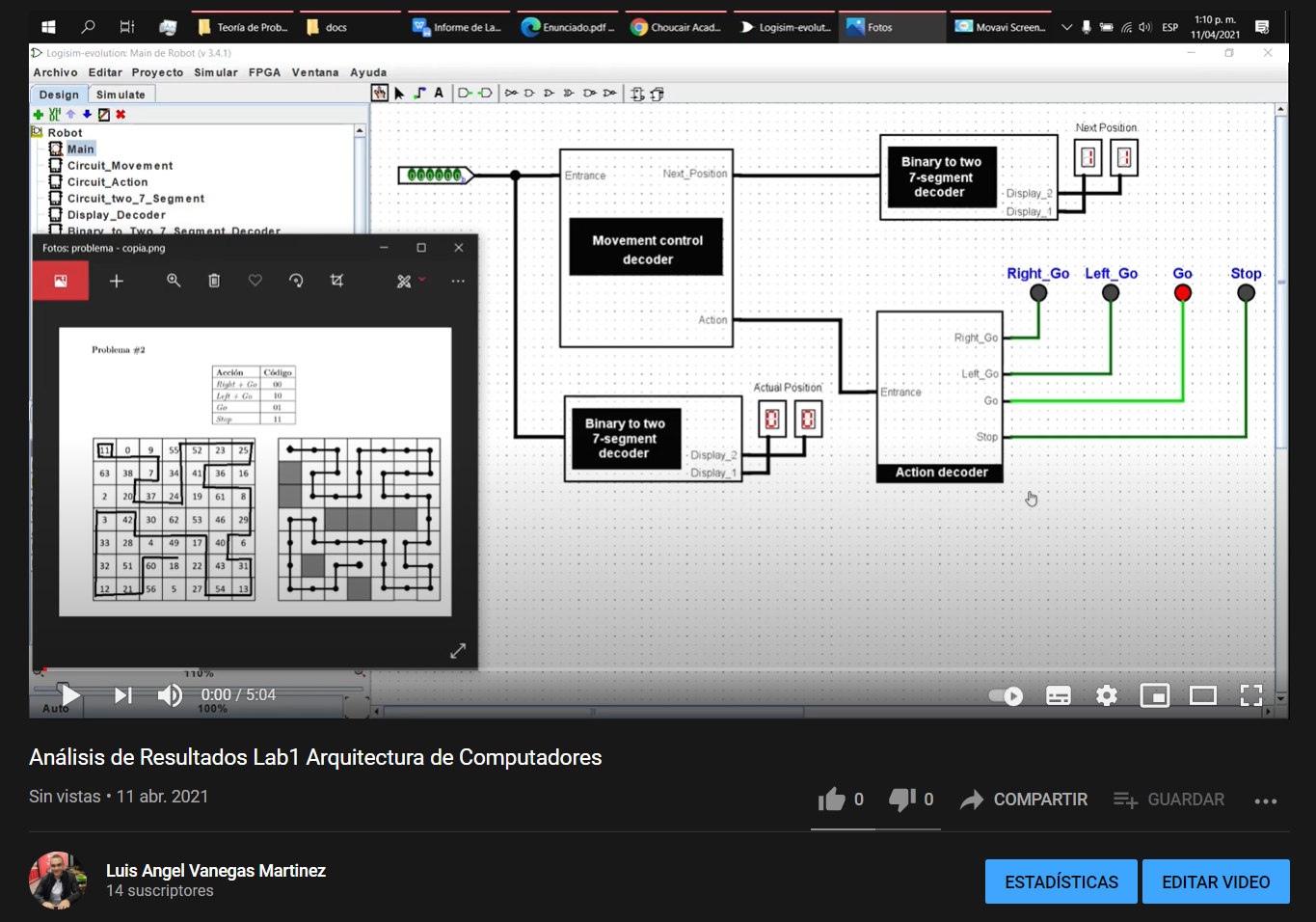


Este caso sucede cuando se pone una posición actual que no exista en la tabla o que este sombreada, mostrará el valor de entrada que se dio y la acción será Stop, adicional la siguiente posición es el mismo valor de entrada porque no puede avanzar.

# Análisis de Resultados

Para un mejor entendimiento plantee el análisis de resultados en forma de vídeo 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/CfGTDqGxXZg>



# Observaciones

Todos los componentes han sido diseñados con el fin de que sean reutilizables y escalables en términos de arquitectura. En caso de querer implementar una nueva funcionalidad, basta con instancia un componente requerido y asignarle los valores de entrada requeridos para que de sus valores de salidas.

El robot se encuentra en la nube documentado y con licencia MIT para un futuro uso de acuerdo a las políticas que yo como creador asigno. El repositorio en mención es: <https://github.com/xlavm/Logisim-Circuits/tree/main/Lab-1>

# Conclusión

Para concluir,Se diseñan circuitos combinacionales que luego son utilizados por componentes de un nivel medio y que se interconectan entre si de acuerdo a la problemática y arquitectura planteada. Estos circuitos son utilizados de forma repetida por los componentes que los requieran; Así mismo estos componentes de nivel medio son instanciados por un componente de alto nivel llamado “Main” este es el componente que se encargará de ejecutar las funcionalidades de los otros componentes de nivel medio y de los circuitos.

Para soluciones más optimas que contribuyan a una mayor escalabilidad podrían emplearse otros componentes como de “Memoria” por ejemplo ROM que registren los valores del problema o poblemas propuestos con el fin de que en caso de un cambio de problema, la arquitectura del circuito no cambie. Esta solución la he planteado y se encuentra en el siguiente repositorio: <https://github.com/xlavm/Logisim-Circuits/tree/main/Lab-1-simplified>