INF245 – Laboratorio 3

Don Bit y Bitópolis

Sebastián Richiardi Pérez – 202030555-2 – 201
Gabriel Alejandro Toro Varela – 202204557-4 – 201

02 de junio de 2025

Índice

1. Introducción

En este laboratorio se implementa un sistema digital en Logisim para simular los recorridos dentro de la ciudad digital Bitópolis. Se emplea lógica secuencial sin memorias ROM/RAM, únicamente con compuertas lógicas y flip-flops tipo D, mostrando paso a paso el trayecto en un display de 7 segmentos.

2. Descripción del problema

Don Bit construyó una ciudad digital donde los vehículos se guían por un código de 4 bits que define un nodo inicial en un grafo. Cada uno de los 16 códigos posibles activa una ruta única y secuencial hacia el nodo final común: F.

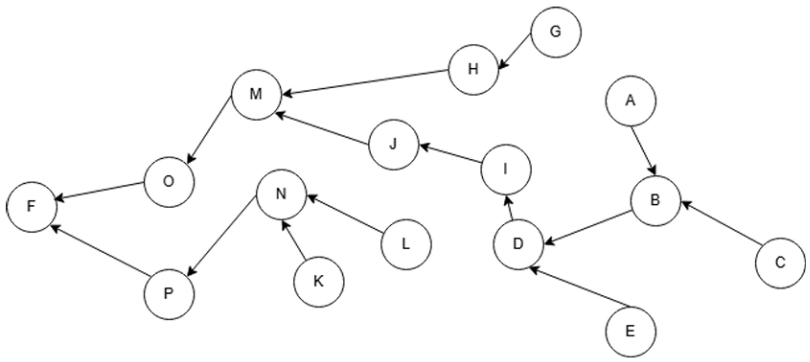

Figura 1: Mapa de rutas del grafo de Bitópolis: cada nodo (A–P) tiene una única transición hacia el nodo final F.

A partir del mapa de rutas presentado en la Figura 1, el objetivo es diseñar un circuito secuencial que, dado un nodo inicial en el rango $\texttt{0000}_2$ a $\texttt{1111}_2$ (correspondiente a las letras A–P), recorra automáticamente la única ruta predefinida hasta el nodo final F. En cada estado intermedio del recorrido, el sistema debe mostrar el identificador del nodo actual en un display de 7 segmentos, usando un decodificador de 4 bits a 7 salidas (subcircuito `D7`).

3. Diseño del sistema

La máquina de estados finitos (FSM) se implementa en Logisim usando únicamente compuertas lógicas, flip-flops tipo D y multiplexores, sin recurrir a memorias ROM/RAM. A continuación se describen los bloques principales y cómo se relacionan:

1. **StateRegister (Registro de estado)**: Consta de cuatro flip-flops tipo D en paralelo que almacenan el "estado actual" $State[3.,0]$. Este registro recibe en cada flanco de reloj el valor de entrada proveniente de `MuxDeCarga`. Además cuenta con una señal de `Reset` asíncrona que fuerza el valor "0000" (nodo `A`) cuando se activa, y con una entrada `Enable` vinculada a `DetectF` para detener la captura una vez que se alcanza el nodo final `F`.

2. **MuxDeCarga (Multiplexor de carga)**: Consta de cuatro multiplexores 2:1 (uno por cada bit) cuyo selector es la señal `LOAD`.

   - Si `LOAD` $= 1$, cada multiplexor deja pasar el bit correspondiente de `UserIn`$[i]$ (entrada de 4 bits del usuario, nodo inicial).
   - Si `LOAD` $= 0$, deja pasar el bit correspondiente de `NextState`$[i]$.

   La salida de cada multiplexor (`D`$[i]$) se conecta a la entrada D de cada flip-flop en `StateRegister`.

3. **NextState (Lógica combinacional 4 $\rightarrow$ 4)**: Toma como entrada los 4 bits $State[3.,0]$ (estado actual) y genera 4 bits $NextState[3.,0]$, calculando el nodo sucesor según la ruta única predefinida. En particular, cuando $State = F$, se asegura que $NextState = F$ para detener el avance. Esta lógica se puede implementar bien mediante mapas de Karnaugh y puertas lógicas, o bien con cuatro multiplexores 16:1 que, seleccionados por ($State[3.,0]$), indiquen cada bit de $NextState$.

4. **DetectF (Detección de nodo final)**: Bloque combinacional que compara $State[3.,0]$ con el valor binario correspondiente a `F` (por ejemplo, $\texttt{0101}_2$ si `F` $= 5$). Cuando coinciden, la salida `Finished` $= 1$. Esta señal alimenta el pin `Enable` de `StateRegister` (o bien se utiliza para forzar $NextState(F) = F$), de modo que, al llegar a `F`, la FSM se "congela" y no avanza más.

5. **D7 (Decodificador 4 $\rightarrow$ 7 segmentos)**: Convierte los 4 bits $State[3.,0]$ en las 7 señales (a, b, c, d, e, f, g) que activan los segmentos del display para formar la letra del nodo actual (A–P). De esta forma, en cada ciclo de reloj, el registro `State` actualiza la salida de `D7`, y el display muestra la letra correspondiente.

6. **Reloj (`CLK`) y control de carga (`LOAD`)**:

   - La señal `CLK` es el pulso principal que sincroniza los flip-flops de `StateRegister`.
   - La señal `LOAD` proviene de un botón o interruptor del usuario:
     - Cuando `LOAD` $= 1$, en el siguiente flanco ascendente de `CLK` se cargan los 4 bits de `UserIn`$[3.,0]$ en `StateRegister`.
     - Inmediatamente después de ese pulso, el usuario deja `LOAD` $= 0$, y entonces `MuxDeCarga` conmuta a usar `NextState`.

3.1. Algoritmo general

El comportamiento de la FSM se puede resumir en los siguientes pasos:

1. **Entrada inicial:** El usuario coloca, mediante interruptores, un valor binario de 4 bits `UserIn`[3.,0] (rango 0000–1111), que representa el nodo inicial S_0.

2. **Carga del estado inicial:**

   - El usuario activa `LOAD` = 1.
   - En el flanco ascendente de `CLK` siguiente, `MuxDeCarga` toma `UserIn` y lo proporciona a las entradas `D`[3.,0] de `StateRegister`.
   - `StateRegister` almacena $State \leftarrow$ `UserIn`.

3. **Inicio del recorrido:** `LOAD` vuelve a 0 (botón suelto). Ahora, `MuxDeCarga` pasa a seleccionar `NextState`[3.,0] en lugar de `UserIn`.

4. **Ciclo de transición:** Repetir en cada flanco ascendente de `CLK` mientras `Finished` = 0:

   a) `DetectF` comprueba si $State = F$.
   - Si $State \neq F$, `Finished` = 0 y `Enable` = 1.
   - Si $State = F$, `Finished` = 1 y `Enable` = 0.

   b) Como `Enable` = 1, en ese flanco de `CLK`, `StateRegister` actualiza

   $$State \leftarrow NextState(State)\,,$$

   donde $NextState(State)$ se obtiene de la lógica combinacional del bloque `NextState`.

   c) Automáticamente, los 4 bits de $State$ pasan al bloque `D7`, el cual enciende los segmentos correspondientes para mostrar la letra $\{\texttt{A}, \texttt{B}, \ldots, \texttt{P}\}$.

5. **Detención al llegar a F:** Cuando $State = F$, el bloque `DetectF` pone `Finished` = 1. Entonces `Enable` = 0 detiene la captura en `StateRegister` (o, alternativamente, $NextState(F) = F$ impide cualquier cambio). El valor F permanece en pantalla, y el recorrido finaliza.

6. **Visualización continua:** Mientras `Finished` = 0, cada nuevo valor de $State$ (el nodo intermedio en la ruta) se muestra automáticamente en el display de 7 segmentos vía `D7`. Una vez que `Finished` = 1, la letra `F` permanece fija hasta que se active nuevamente `Reset`.

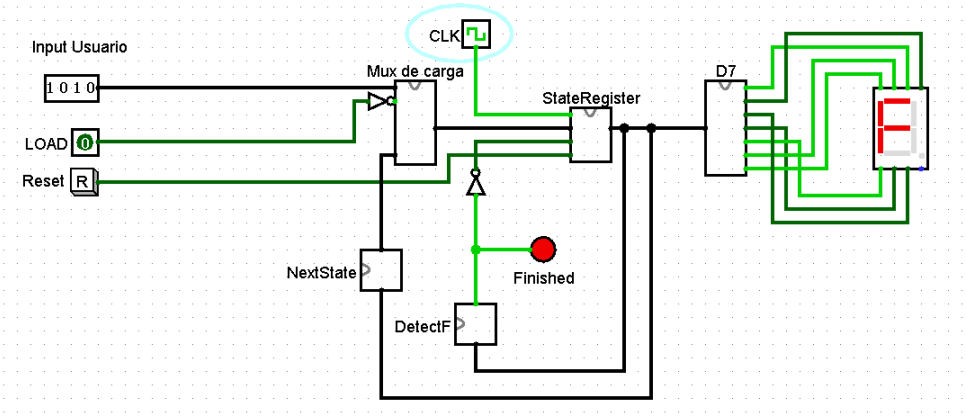

Figura 2: Diagrama general del sistema secuencial implementado en Logisim.

Descripción de los bloques en la Figura 2:

- **StateRegister (4 D–FF)**: almacena el "estado actual" (`State[3..0]`).
- **MuxDeCarga**: multiplexores 2:1 que eligen entre `UserIn` y `NextState` según `LOAD`.
- **NextState ($4 \to 4$)**: lógica combinacional o MUXes 16:1 que calculan $NextState = f(State)$.
- **DetectF ($4 \to 1$)**: comparador que genera `Finished` cuando $State = F$.
- **D7 ($4 \to 7$)**: decodificador que convierte $State$ en las señales (a, b, c, d, e, f, g) del display.
- `CLK`: señal de reloj que sincroniza `StateRegister` y `DetectF`.
- `LOAD`: botón que activa la carga inicial de `UserIn` en `StateRegister`.
- `Reset`: señal asíncrona que fuerza $State \leftarrow 0000$ (nodo A).
- `Finished`: indicador (LED) que se enciende cuando se llega a F.

3.2. Subcircuito `Decodificador 7 segmentos`

El subcircuito `D7` recibe como entrada el código en binario de 4 bits (s_3, s_2, s_1, s_0) que representa el nodo actual (A... P) y produce como salida el código en binario de 7 bits (a, b, c, d, f, g) correspondiente al segmento del display LED.

3.2.1. Tabla de verdad del subcircuito

Dec	Binario	Nodo	a	b	c	d	e	f	g
0	0000	A	1	1	1	0	1	1	1
1	0001	B	0	0	1	1	1	1	1
2	0010	C	1	0	0	1	1	1	0
3	0011	D	0	1	1	1	1	0	1
4	0100	E	1	0	0	1	1	1	1
5	0101	F	1	0	0	0	1	1	1
6	0110	G	1	0	1	1	1	1	0
7	0111	H	0	1	1	0	1	1	1
8	1000	I	0	1	1	0	0	0	0
9	1001	J	0	1	1	1	1	0	0
10	1010	K	1	0	1	0	1	1	1
11	1011	L	0	0	0	1	1	1	0
12	1100	M	1	0	1	0	1	0	1
13	1101	N	1	0	1	0	1	0	0
14	1110	O	1	1	1	1	1	1	0
15	1111	P	1	1	0	0	1	1	1

Cuadro 1: Tabla de verdad del decodificador de 7 segmentos para nodos A–P.

3.2.2. Mapas de Karnaugh para cada bit de salida

Mapa K-map de segmento a

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	1	0	0	1
01	1	1	0	1
11	1	1	1	1
10	0	0	0	1

Función minimizada para a:

$$x_3x_2 + x_2\overline{x_1} + \overline{x_3}\,\overline{x_1}\,\overline{x_0} + x_1\,\overline{x_0}$$

Mapa K-map de segmento b

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	1	0	1	0
01	0	0	1	0
11	1	0	1	1
10	1	1	0	0

Función minimizada para b:

$$\overline{x_2}\,\overline{x_1}\,\overline{x_0} + x_3\,\overline{x_2}\,\overline{x_1} + \overline{x_3}\,x_1\,x_0 + x_2\,x_1\,x_0 + x_3\,x_2\,x_1$$

Mapa K-map de segmento c

$x_3x_2\backslash x_1x_0$	00	01	11	10
00	1	1	1	0
01	0	0	1	1
11	1	1	0	1
10	1	1	0	1

Función minimizada para c:

$$\overline{x_2}\,\overline{x_1} + \overline{x_3}\,\overline{x_2}\,x_0 + \overline{x_3}\,x_2\,x_1 + x_3\,x_1\,\overline{x_0} + x_3\,x_2\,\overline{x_1}$$

Mapa K-map de segmento d

$x_3x_2\backslash x_1x_0$	00	01	11	10
00	0	1	1	1
01	1	0	0	1
11	0	0	0	1
10	0	1	1	0

Función minimizada para d:

$$\overline{x_3}\,x_1\,\overline{x_0} + \overline{x_3}\,\overline{x_2}\,x_1 + x_2\,x_1\,\overline{x_0} + \overline{x_3}\,\overline{x_2}\,x_0 + x_3\,\overline{x_2}\,x_0 + \overline{x_3}\,x_2\,\overline{x_1}\,\overline{x_0}$$

Mapa K-map de segmento e

$x_3x_2\backslash x_1x_0$	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	0	1	1	1

Función minimizada para e:

$$\overline{x_3} + x_2 + x_1 + x_0$$

Mapa K-map de segmento f

$x_3x_2\backslash x_1x_0$	00	01	11	10
00	1	1	0	1
01	1	1	1	1
11	0	0	1	1
10	0	0	1	1

Función minimizada para f:

$$x_3\,x_1 + \overline{x_3}\,x_2 + \overline{x_3}\,\overline{x_2}\,\overline{x_1} + x_1\,\overline{x_0}$$

Mapa K-map de segmento g

$x_3x_2 \backslash x_1x_0$	00	01	11	10
00	1	1	1	0
01	1	1	1	0
11	1	0	1	0
10	0	0	0	1

Función minimizada para g:

$$\overline{x_3}\,\overline{x_1} + \overline{x_3}\,x_1\,x_0 + x_2\,\overline{x_1}\,\overline{x_0} + x_2\,x_1\,x_0 + x_3\,\overline{x_2}\,x_1\,\overline{x_0}$$

3.3. Subcircuito `NextState`

El subcircuito `NextState` recibe como entrada el código en binario de 4 bits (s_3, s_2, s_1, s_0) que representa el nodo actual (A. . . P) y produce como salida el código en binario de 4 bits (n_3, n_2, n_1, n_0) correspondiente al siguiente nodo en la ruta.

3.3.1. Tabla de verdad del subcircuito

Estado presente $(s_3\ s_2\ s_1\ s_0)$				**NextState** $(n_3\ n_2\ n_1\ n_0)$			
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	1
0	0	1	0	0	0	0	1
0	0	1	1	1	0	0	0
0	1	0	0	0	0	1	1
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	1
0	1	1	1	1	1	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	0	1	1	1	0
1	1	0	1	1	1	1	1
1	1	1	0	0	1	0	1
1	1	1	1	0	1	0	1

Cuadro 2: Tabla de verdad del subcircuito `NextState`. Cada fila indica la transición desde el estado presente $(s_3\, s_2\, s_1\, s_0)$ al siguiente $(n_3\, n_2\, n_1\, n_0)$.

3.3.2. Mapas de Karnaugh para cada bit de salida

Se usarán mapas de Karnaugh 4×4 con la convención Gray en filas $(s_3\, s_2)$ y columnas $(s_1\, s_0)$.

(a) Mapa de Karnaugh para $n_3(s_3, s_2, s_1, s_0)$.

	$(s_1\, s_0)$			
$(s_3\, s_2)$	00	01	11	10
00	0	0	**1**	0
01	0	0	**1**	0
11	**1**	**1**	0	0
10	**1**	**1**	**1**	**1**

Agrupaciones:

- $\{8, 9, 11, 10\}$ (fila $s_3 s_2 = 10$ entera) $\longrightarrow$ $s_3\, \overline{s_2}$.
- $\{8, 9, 12, 13\}$ ($(s_3 s_2) \in \{10, 11\}$, $(s_1 s_0) \in \{00, 01\}$) $\longrightarrow$ $s_3\, \overline{s_1}$.

- $\{3, 7\}$ $((s_3 s_2) \in \{00, 01\},\ (s_1 s_0) = 11) \longrightarrow \overline{s_3}\, s_1\, s_0$.

Función minimizada para n_3:

$$n_3 = \underbrace{s_3\,\overline{s_2}}_{\text{(fila 10)}} + \underbrace{s_3\,\overline{s_1}}_{\substack{\text{(filas 10–11 en} \\ (s_1 s_0)\in\{00,01\})}} + \underbrace{\overline{s_3}\,s_1\,s_0}_{\text{(columnas 11)}}.$$

(b) Mapa de Karnaugh para $n_2(s_3, s_2, s_1, s_0)$.

	$(s_1\ s_0)$			
$(s_3\ s_2)$	00	01	11	10
00	0	0	0	0
01	0	**1**	**1**	**1**
11	**1**	**1**	**1**	**1**
10	0	**1**	**1**	**1**

Agrupaciones:

- $\{12, 13, 15, 14\}$ (fila $s_3 s_2 = 11$ entera) $\longrightarrow s_3\, s_2$.
- $\{6, 7, 15, 14\}$ $((s_3 s_2) \in \{01, 11\},\ (s_1 s_0) \in \{11, 10\}) \longrightarrow s_2\, s_1$.
- $\{6, 7, 11, 10\}$ $((s_3 s_2) \in \{01, 10\},\ (s_1 s_0) \in \{11, 10\}) \longrightarrow s_1$.
- $\{9, 13\}$ $((s_3 s_2) \in \{10, 11\},\ (s_1 s_0) = 01) \longrightarrow s_3\, s_0$.
- $\{10, 14\}$ $((s_3 s_2) \in \{10, 11\},\ (s_1 s_0) = 10) \longrightarrow s_2\, s_0$.

Función minimizada para n_2:

$$n_2 = \underbrace{s_3\,s_2}_{\{12,13,15,14\}} + \underbrace{s_2\,s_1}_{\{6,7,15,14\}} + \underbrace{s_1}_{\{6,7,11,10\}} + \underbrace{s_3\,s_0}_{\{9,13\}} + \underbrace{s_2\,s_0}_{\{10,14\}}.$$

(c) Mapa de Karnaugh para $n_1(s_3, s_2, s_1, s_0)$.

	$(s_1\ s_0)$			
$(s_3\ s_2)$	00	01	11	10
00	0	**1**	0	0
01	**1**	0	0	**1**
11	**1**	**1**	0	0
10	0	0	0	0

Agrupaciones:

- $\{4, 12\}$ (fila $s_3 s_2 = 01, 11$, columna "00"):

$$s_1 = 0,\ s_0 = 0,\ s_2 = 1,\ (s_3 \text{ varía}) \longrightarrow s_2\,\overline{s_1}\,\overline{s_0}.$$

- $\{12, 13\}$ (fila $s_3 s_2 = 11$, columnas $\{00, 01\}$):

$$s_3 = 1,\ s_2 = 1,\ s_1 = 0,\ (s_0 \text{ varía}) \longrightarrow s_3\, s_2\,\overline{s_1}.$$

- $\{6\}$ (aislado en fila "01", columna "10"):

$$(s_3, s_2, s_1, s_0) = (0, 1, 1, 0) \longrightarrow \overline{s_3}\, s_2\, s_1\, \overline{s_0}.$$

- $\{1\}$ (aislado en fila "00", columna "01"):

$$(s_3, s_2, s_1, s_0) = (0, 0, 0, 1) \longrightarrow \overline{s_3}\, \overline{s_2}\, \overline{s_1}\, s_0.$$

Función minimizada para n_1:

$$n_1 = \underbrace{\left(\overline{s_3}\, s_2\, \overline{s_0}\right)}_{\{4,6\}} + \underbrace{\left(s_3\, s_2\, \overline{s_1}\right)}_{\{12,13\}} + \underbrace{\left(\overline{s_3}\, \overline{s_2}\, \overline{s_1}\, s_0\right)}_{\{1\}}.$$

(d) Mapa de Karnaugh para $n_0(s_3, s_2, s_1, s_0)$.

	$(s_1\, s_0)$			
$(s_3\, s_2)$	00	01	11	10
00	**1**	**1**	0	**1**
01	**1**	**1**	0	**1**
11	0	**1**	**1**	**1**
10	**1**	0	**1**	**1**

Agrupaciones:

- $\{0, 1, 4, 5\}$ (filas "00–01", columnas "00–01"): $s_3 = 0,\ s_1 = 0 \longrightarrow \overline{s_3}\, \overline{s_1}$.
- $\{0, 2, 4, 6\}$ (imbuída: filas "00–01", columnas "00–10" (envolvente)): $s_3 = 0,\ s_0 = 0 \longrightarrow \overline{s_3}\, \overline{s_0}$.
- $\{10, 11, 14, 15\}$ (filas "10–11", columnas "10–11"): $s_3 = 1,\ s_1 = 1 \longrightarrow s_3\, s_1$.
- $\{8, 10\}$ (fila "10", columnas "00–10"): $s_3 = 1,\ s_0 = 0 \longrightarrow s_3\, \overline{s_0}$.
- $\{13\}$ (aislado en fila "11", columna "01"): $(s_3, s_2, s_1, s_0) = (1, 1, 0, 1) \longrightarrow s_3\, s_2\, \overline{s_1}\, s_0$.

Función minimizada para n_0:

$$n_0 = \underbrace{\left(\overline{s_3}\, \overline{s_1}\right)}_{p_1} + \underbrace{\left(\overline{s_3}\, \overline{s_0}\right)}_{p_2} + \underbrace{\left(s_3\, s_1\right)}_{p_3} + \underbrace{\left(s_3\, \overline{s_0}\right)}_{p_5} + \underbrace{\left(s_3\, s_2\, \overline{s_1}\, s_0\right)}_{p_6}.$$

3.3.3. Funciones lógicas minimizadas finales

Finalmente, las cuatro salidas (n_3, n_2, n_1, n_0) se describen mediante:

$$n_3(s_3, s_2, s_1, s_0) = s_3\, \overline{s_2} + s_3\, \overline{s_1} + \overline{s_3}\, s_1\, s_0,$$

$$n_2(s_3, s_2, s_1, s_0) = s_3\, s_2 + s_2\, s_1 + s_1 + s_3\, s_0 + s_2\, s_0,$$

$$n_1(s_3, s_2, s_1, s_0) = \overline{s_3}\, s_2\, \overline{s_0} + s_3\, s_2\, \overline{s_1} + \overline{s_3}\, \overline{s_2}\, \overline{s_1}\, s_0,$$

$$n_0(s_3, s_2, s_1, s_0) = \overline{s_3}\, \overline{s_1} + \overline{s_3}\, \overline{s_0} + s_3\, s_1 + s_3\, \overline{s_0} + s_3\, s_2\, \overline{s_1}\, s_0.$$

3.4. Subcircuito `MuxDeCarga`

El subcircuito `MuxDeCarga` es responsable de seleccionar, en cada ciclo de reloj, si el registro principal debe cargar la entrada del usuario (nodo inicial) o bien el valor proporcionado por la lógica `NextState`. Para lograr esto, se implementa un multiplexor de 2 a 1 para cada uno de los cuatro bits. Cuando la señal `LOAD` está activa, se elige la entrada del usuario; en caso contrario, se deja pasar la salida de `NextState`. La figura 3 muestra el diagrama de este bloque.

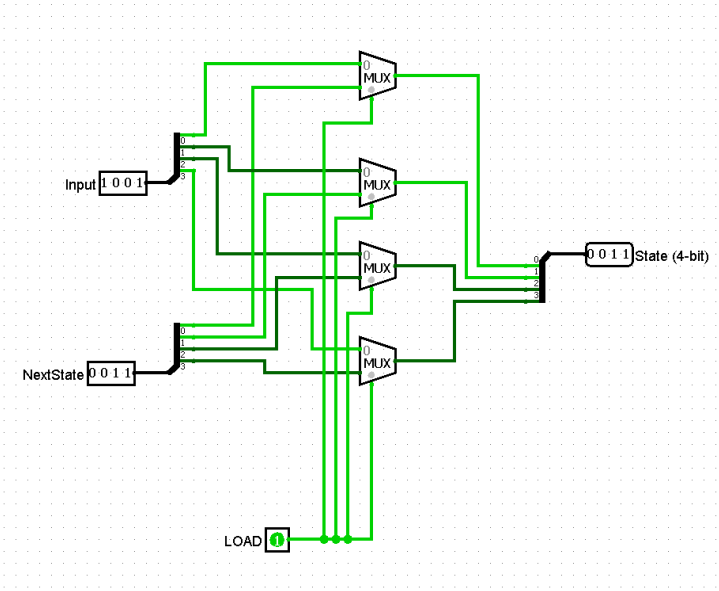

Figura 3: Subcircuito `MuxDeCarga`: selección entre entrada de usuario y NextState.

En pseudocódigo o en forma de tabla de verdad, su funcionamiento es:

- **Entradas:**
  - `LOAD` (bit de control).
  - `UserIn[3..0]` (interruptores del usuario, nodo inicial).
  - `NextState[3..0]` (salida combinacional que indica el siguiente estado).
- **Salidas:**
  - `D[3..0]` (bits que alimentan el registro de estado).
- **Tabla de selección (por bit):**

$$D[i] \;=\; \begin{cases} \texttt{UserIn}[i], & \text{si } \texttt{LOAD} = 1, \\ \texttt{NextState}[i], & \text{si } \texttt{LOAD} = 0. \end{cases} \qquad (i = 0, 1, 2, 3)$$

Internamente, se usan cuatro puertas MUX 2:1 (una por cada bit) tal como se ilustra en la figura.

3.5. Subcircuito `DetectF`

El subcircuito `DetectF` detecta cuándo el registro de estado ha alcanzado el nodo final F. En nuestro diseño, consideramos que F está codificado en 4 bits —por ejemplo, $F = 0101_2$ si el nodo final es F = 5. Cuando el estado actual coincide con ese valor, la salida de `DetectF` se pone a 1 ("`Finished` = 1"), lo cual se usa para, opcionalmente, detener el avance del registro (pasando `Enable` = 0 a los flip-flops). La figura 4 muestra el diagrama de este comparador.

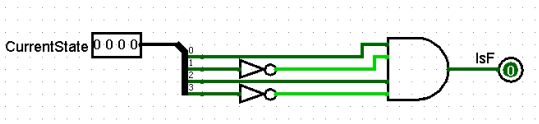

Figura 4: Subcircuito `DetectF`: compara el estado con el valor de F.

Descripción del funcionamiento:

- **Entradas:** $State[3.,0]$ (codificación binaria del nodo actual).
- **Salida:** `Finished` = 1 si y solo si $State = F$.
- **Implementación típica:**

$$\texttt{Finished} \;=\; \left(\overline{State_3}\right) \wedge State_2 \;\wedge\; \overline{State_1} \;\wedge\; State_0 \quad (\text{si } F = 0101_2),$$

  donde cada literal (p. ej. $\overline{State_3}$) se obtiene con un inversor.

Adicionalmente, la señal `Finished` se conecta al pin de `Enable` de los flip-flops de `StateRegister` para "congelar" el estado final y evitar más transiciones.

3.6. Subcircuito `StateRegister`

El subcircuito `StateRegister` consta de cuatro flip-flops tipo D sincronizados con el reloj `CLK`. Cada flip-flop almacena uno de los bits del "estado actual" $State[3.,0]$. El registro recibe su entrada (`D`[3.,0]) directamente desde `MuxDeCarga`. La señal de `Reset` asíncrona se utiliza para inicializar el registro en "0000" (nodo A) en el momento deseado, y la señal `Enable` (opcional) se activa con $\overline{\texttt{Finished}}$ para detener el registro al llegar a F. En la figura 5 se muestra su circuito en logisim.

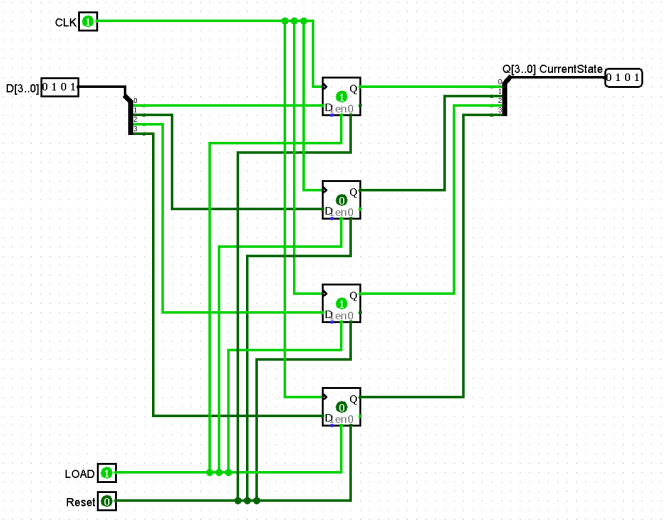

Figura 5: Subcircuito `StateRegister`: cuatro D-Flip-Flops con Reset y Enable.

Pines de control:

- **CLK:** Reloj principal (flanco ascendente) que sincroniza cada flip-flop.
- **Reset:** Señal asíncrona para forzar $State \leftarrow 0000$.
- **Enable:** (opcional) $= \overline{\texttt{Finished}}$. Cuando `Finished` $= 1$, el registro deja de capturar flancos de reloj y mantiene el estado final F.
- **D[3..0]:** Bits de entrada, provenientes de `MuxDeCarga`.
- **Q[3..0]:** Salida actual (`State`), que retroalimenta a `NextState` y `DetectF`, y también se envía a `D7` para el display de 7 segmentos.

Cada flip-flop almacena un bit $State_i$.

4. Supuestos

- No se requiere retroceso ni manejo de errores en tiempo real.
- El sistema parte desde reposo y no repite el ciclo una vez finalizado.

5. Conclusión

En este informe se ha presentado el diseño completo de un circuito secuencial capaz de recibir un código de 4 bits (valor de 0 a 15) que identifica un nodo de inicio en la ciudad digital de Bitópolis, recorrer automáticamente la ruta única y predefinida hacia el nodo final `F`, y mostrar en un display de 7 segmentos cada nodo visitado en cada pulso de reloj.

Para ello, se desglosó la solución en los siguientes bloques:

1. **MuxDeCarga**, que permite seleccionar entre el valor ingresado por el usuario o el valor calculado por *NextState*.
2. **StateRegister**, formado por cuatro flip-flops tipo D, que almacenan el estado actual (nodo en curso) y avanzan de acuerdo al reloj.
3. **DetectF**, que detecta cuándo el estado coincide con el nodo final `F` para encender el indicador `Finished` y detener el recorrido.
4. **NextState**, la lógica combinacional que, dado el estado actual, calcula el siguiente nodo.
5. **Decodificador D7**, que traduce el código de 4 bits en las siete líneas necesarias para encender el display y mostrar la letra correspondiente al nodo actual.

El uso de flip-flops tipo D, multiplexores y puertas lógicas cumple con la restricción de no emplear memorias ROM/RAM ni otros tipos de flip-flops. Además, la señal `LOAD` (activada o negada según la implementación elegida) permite tanto la carga inicial del nodo como el pase a la ejecución automática de la secuencia. El LED `Finished` indica la finalización del recorrido sin entrar en bucle, y el botón `Reset` ofrece un mecanismo sencillo para reiniciar el sistema a `0000` en cualquier momento.

Finalmente, la modularización en los subcircuitos `MuxDeCarga`, `StateRegister` y `DetectF` facilita la depuración y el mantenimiento, y otorga claridad al diseño global.

Con esto, se demuestra un ejemplo completo de un sistema secuencial controlado por reloj, capaz de mostrar de forma visual y paso a paso la trayectoria de cualquier nodo de inicio hasta el nodo final en Bitópolis.