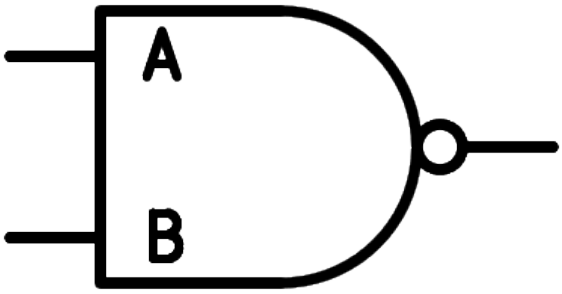

Trabajo práctico 3

Montacargas

- **Autores:**
  - Nahuel Valentin Pereyra - Leg. 402333
  - Valentino Rao - Leg. 402308
- **Curso:** 3R1
- **Docentes:**
  - Ing. Olmedo, Sergio Daniel
  - Ing Molla, Florencia Belén
- **Asignatura:** Técnicas Digitales I
- **Institución:** Universidad Tecnológica Nacional - Facultad Regional de Córdoba.

UTN FRC

Índice

1. Implementación del Diseño

1.1. Descripción del Sistema

El objetivo del diseño es la automatización de un montacargas de tres niveles. El sistema debe ser capaz de recibir comandos de llamada desde tres pulsadores ($P1, P2, P3$), detectar la posición de la cabina mediante finales de carrera ($fc1, fc2, fc3$) y controlar tanto el motor de tracción como un indicador visual de posición.

El núcleo del control se basa en una **Máquina de Estados Finitos (FSM)** que gestiona la lógica secuencial del movimiento. Debido a la necesidad de una respuesta inmediata de las salidas (indicadores de motor y display) ante cambios en las entradas, se optó por una arquitectura de **Máquina de Mealy**.

1.2. Máquina de Estados Finita (FSM)

La FSM diseñada consta de 7 estados lógicos, codificados en un registro de 3 bits. La selección del modelo de Mealy permite que las salidas (S para el motor y D para el display) dependan tanto del estado actual como de las entradas presentes, lo que reduce la latencia en la indicación de las acciones.

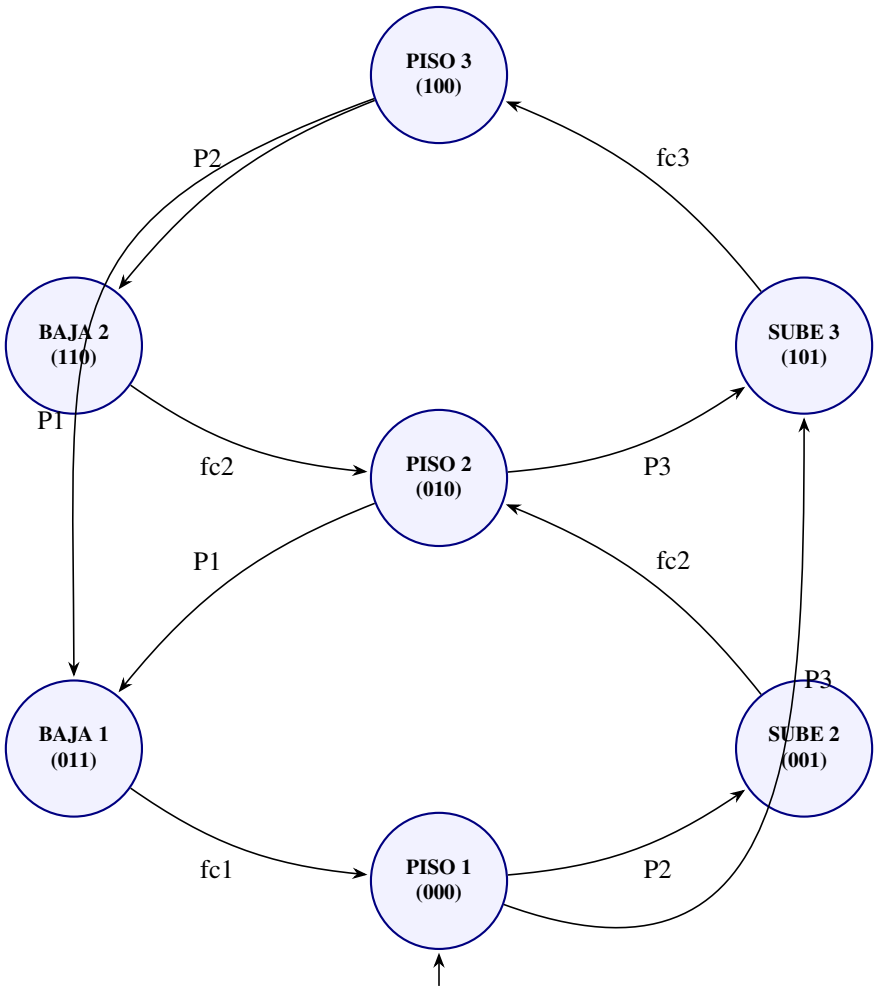

Figura 1: Diagrama de Estados (FSM) del Montacargas.

1.2.1. Definición de Estados

Los estados se clasifican en dos categorías funcionales:

- **Estados de Reposo (Estacionarios):** Representan la permanencia del montacargas en un piso específico.
  - **PISO 1:** Estado inicial y de reposo en el primer nivel. Salidas: $S = 00$ (Quieto), $D = 1$ (Piso 1).
  - **PISO 2:** Estado de reposo en el segundo nivel. Salidas: $S = 00$, $D = 2$.
  - **PISO 3:** Estado de reposo en el tercer nivel. Salidas: $S = 00$, $D = 3$.

- **Estados de Transición (Movimiento):** Representan el desplazamiento de la cabina entre niveles.
  - **SUBE 2:** Movimiento ascendente hacia el piso 2. Salidas: $S = 01$ (Sube), $D =$ Apagado.
  - **SUBE 3:** Movimiento ascendente hacia el piso 3. Salidas: $S = 01$, $D =$ Apagado.
  - **BAJA 1:** Movimiento descendente hacia el piso 1. Salidas: $S = 10$ (Baja), $D =$ Apagado.
  - **BAJA 2:** Movimiento descendente hacia el piso 2. Salidas: $S = 10$, $D =$ Apagado.

1.2.2. Lógica de Transiciones

El diagrama de estados se comporta de la siguiente manera:

I. Estando en **PISO 1**: Si se activa $P2$, transita a *SUBE 2*. Si se activa $P3$, transita a *SUBE 3*.

II. Estando en **SUBE 2**: Permanece en este estado hasta que $fc2$ se activa, momento en el que transita a *PISO 2*.

III. Estando en **PISO 2**: Si se activa $P3$, transita a *SUBE 3*. Si se activa $P1$, transita a *BAJA 1*.

IV. Estando en **SUBE 3**: Permanece en este estado hasta que $fc3$ se activa, provocando la transición a *PISO 3*.

V. Estando en **PISO 3**: Si se activa $P2$, transita a *BAJA 2*. Si se activa $P1$, transita a *BAJA 1*.

VI. Estando en **BAJA 2** o **BAJA 1**: El sistema mantiene el movimiento descendente hasta que se activa el final de carrera correspondiente ($fc2$ o $fc1$), retornando al estado de reposo respectivo.

1.3. Salidas del Sistema

El comportamiento de las salidas en función del estado y las entradas se resume a continuación:

- **Motor (S):** 2 bits. '00' detiene el motor, '01' activa la subida y '10' activa la bajada.
- **Display (D):** 7 bits. Muestra el número del piso actual solo cuando el montacargas está detenido (en estados PISO X). Durante los estados de movimiento, el display permanece apagado para indicar tránsito.

2. Descripción en HDL y Hardware

2.1. Código Verilog

La descripción del hardware se realizó utilizando el lenguaje Verilog. El módulo principal, denominado `monta`, implementa la máquina de estados descrita anteriormente.

2.1.1. Código

```
`timescale 1ns / 1ps
module monta(
 input clk, //pulso de reloj
 output en, //habilitador del display
 input reset, //bot n de reset
 input P1, //piso 1
 input P2, //piso 2
 input P3, //piso 3
 input fc1, //fin de carrera 1
 input fc2, //fin de carrera 2
 input fc3, //fin de carrera 3
 output reg [1:0] S, //leds de movimiento
 output reg [6:0] D //display
);
assign en = 0;

reg [2:0] state; //estado
reg [2:0] nextstate; //estado siguiente

parameter piso1 = 3'b000;
parameter piso2 = 3'b010;
parameter piso3 = 3'b100;

parameter sube2 = 3'b001;
parameter baja1 = 3'b011;
parameter sube3 = 3'b101;
parameter baja2 = 3'b110;
```

```
28
29 always @ (posedge clk, posedge reset) //bucle del estado siguiente
30  if (reset) state <= piso1; //reseteo
31  else state <= nextstate; //estado siguiente
32
33 always @(P1, P2, P3, fc1, fc2, fc3, state, nextstate)
34  case(state) //estado actual
35 ////////////////////////////////////////////////////////////////
36   piso1:
37    if(P2) //piso seleccionado
38     begin
39      nextstate <= sube2;
40      S <= 2'b01;
41      D <= 7'b0000000;
42     end
43    else if(P3)
44     begin
45      nextstate <= sube3;
46      S <= 2'b01;
47      D <= 7'b0000000;
48     end
49    else
50     begin
51      nextstate <= state;
52      S <= 2'b00;
53      D <= 7'b0110000;
54     end
55 ////////////////////////////////////////////////////////////////
56   sube2:
57    if(fc2)
58     begin
59      nextstate <= piso2;
60      S <= 2'b00;
61      D <= 7'b1101101;
62     end
63    else
64     begin
65      nextstate <= state;
66      S <= 2'b01;
67      D <= 7'b0000000;
68     end
69 ////////////////////////////////////////////////////////////////
70
71   sube3:
72    if(fc3)
73     begin
74      nextstate <= piso3;
75      S <= 2'b00;
76      D <= 7'b1111001;
77     end
78    else
79     begin
80      nextstate <= state;
81      S <= 2'b01;
82      D <= 7'b0000000;
83     end
84 ////////////////////////////////////////////////////////////////
85
86   piso2:
87    if(P1)
88     begin
89      nextstate <= baja1;
90      S <= 2'b10;
91      D <= 7'b0000000;
92     end
93    else if(P3)
94     begin
95      nextstate <= sube3;
96      S <= 2'b01;
97      D <= 7'b0000000;
98     end
99    else
100    begin
101     nextstate <= state;
102     S <= 2'b00;
```

```
103       D <= 7'b1101101;
104      end
105 ////////////////////////////////////////////////////////////////////
106   baja1:
107    if(fc1)
108     begin
109      nextstate <= piso1;
110      S <= 2'b00;
111      D <= 7'b0110000;
112     end
113   else
114    begin
115     nextstate <= state;
116     S <= 2'b10;
117     D <= 7'b0000000;
118    end
119 ////////////////////////////////////////////////////////////////////
120   piso3:
121    if(P1)
122     begin
123      nextstate <= baja1;
124      S<= 2'b10;
125      D <= 7'b0000000;
126     end
127    else if(P2)
128     begin
129      nextstate <= baja2;
130      S<= 2'b10;
131      D <= 7'b0000000;
132     end
133    else
134     begin
135      nextstate <= state;
136      S <= 2'b00;
137      D <= 7'b1111001;
138     end
139 ////////////////////////////////////////////////////////////////////
140   baja2:
141    if(fc2)
142     begin
143      nextstate <= piso2;
144      S <= 2'b00;
145      D <= 7'b1101101;
146     end
147    else
148     begin
149      nextstate <= state;
150      S <= 2'b10;
151      D <= 7'b0000000;
152     end
153 ////////////////////////////////////////////////////////////////////
154
155   default: nextstate <= piso1; //en caso de piso inv lido, regresar al piso 1
156  endcase
157 endmodule
```

2.1.2. *Estructura del Módulo*

El diseño se divide en dos bloques principales dentro del módulo:

I. **Lógica Secuencial (Memoria de Estado):** Se utiliza un bloque `always @ (posedge clk, posedge reset)` para actualizar el estado del sistema.

- Si la señal de `reset` es alta, el sistema se fuerza asincrónicamente al estado `piso1` (estado inicial).
- En cada flanco positivo del reloj (`clk`), el registro `state` toma el valor de `nextstate`.

II. **Lógica Combinacional (Estado Siguiente y Salida):** Se utiliza un bloque `always @(...)` sensible a todas las entradas y al estado actual. Mediante una sentencia `case`, se evalúa el estado presente para determinar:

- El valor de `nextstate` según las entradas de los pulsadores (P) y finales de carrera (fc).
- Los valores de las salidas S (motor) y D (display).

Se definieron parámetros (ej. `parameter piso1 = 3'b000`) para hacer el código más legible y facilitar el mantenimiento.

2.2. Implementación en FPGA/CPLD

El diseño fue sintetizado e implementado en el kit de desarrollo de la cátedra, basado en un CPLD de Xilinx. La asignación de pines (User Constraints) se realizó de acuerdo a la disposición física de los periféricos en la placa.

2.2.1. Asignación de Puertos

A continuación se detalla el mapeo físico entre las señales del diseño Verilog y los pines del dispositivo CPLD:

Tabla 1: Asignación de Pines en el CPLD

Señal Verilog	Pin CPLD	Descripción del Periférico
`clk`	P6	Señal de reloj (Frecuencia Variable)
`reset`	P38	Switch para reinicio del sistema
`P1, P2, P3`	P34, P35, P36	Switches para selección de piso
`fc1, fc2`	P27, P33	Pulsadores (Lógica inversa: 0 presionado)
`fc3`	P37	Switch (Simulación de final de carrera superior)
`S`	P14	LED indicador de estado del motor
`en`	P28	Habilitador del Display (Transistor)
`D[6:0]`	P18 - P26	Segmentos del Display 7-Segmentos (a-g)

2.2.2. Consideraciones de Hardware

- **Display de 7 Segmentos:** Los segmentos están conectados a los pines P18 a P26. El código envía los vectores de bits correspondientes para formar los números '1', '2' y '3'. El pin P28 controla el habilitador común del display.
- **Entradas de Usuario:** Se utilizaron switches (P34-P36) para simular las llamadas de los pisos, permitiendo mantener la señal activa.
- **Sensores de Posición:** Para los finales de carrera 1 y 2 se utilizaron los pulsadores de la placa (P27 y P33), los cuales operan con lógica negativa (envían un '0' al ser presionados). Para el final de carrera 3, se utilizó un switch (P37) por disponibilidad de hardware.