Tarea 6

Sebastian Astorga y Leonardo Millar

[a] *Pontificia Universidad Católica de Chile, Av. Vicuña Mackenna 4686, Santiago de Chile,*

1. Maquina dinero en juego

Hacemos el diagrama de flujo

$$Entradas : A = Inicio, D_1 D_0 = Dado$$
$$Salidas : Dinero$$

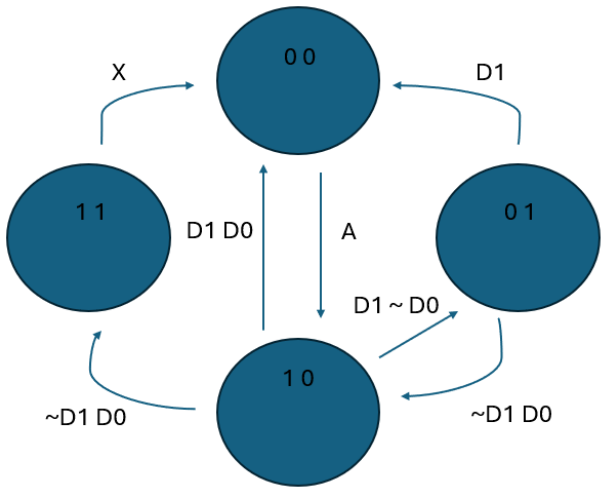

Figura 1: Diagrama de flujo dinero en juego

Construimos la tabla de verdad:

A	D_1	D_0	Q_1^t	Q_0^t	Q_1^{t+1}	Q_0^{t+1}	N
0	0	1	0	0	0	0	0
0	0	1	0	1	1	0	2
0	0	1	1	0	1	1	3
0	0	1	1	1	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0
0	1	0	1	0	0	1	1
0	1	0	1	1	0	0	0
0	1	1	0	0	0	0	0
0	1	1	0	1	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	1	0	0	0
1	0	1	0	0	1	0	2
1	0	1	0	1	1	0	2
1	0	1	1	0	1	1	3
1	0	1	1	1	0	0	0
1	1	0	0	0	1	0	2
1	1	0	0	1	0	0	0
1	1	0	1	0	0	1	1
1	1	0	1	1	0	0	0
1	1	1	0	0	1	0	2
1	1	1	0	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0

Hacemos los mapas de karnaugh para Q0:

A = 0

D_1D_0 \ Q_1Q_0	00	01	11	10
00	x	x	x	x
01	0	1	0	1
11	0	0	0	0
10	0	0	0	0

Quedando en la siguiente ecuacion:

$$Q_1^{t+1} = A\tilde{Q}_1^t\tilde{Q}_0^t + \tilde{D}_1\tilde{Q}_1^tQ_0^t + \tilde{D}_1\tilde{Q}_0^tQ_1^t$$

A = 1

$D_1D_0 \diagdown Q_1Q_0$	00	01	11	10
00	x	x	x	x
01	1	1	0	1
11	1	0	0	0
10	1	0	0	0

Haciendo los mapas para Q1:

A = 0

$D_1D_0 \diagdown Q_1Q_0$	00	01	11	10
00	x	x	x	x
01	0	0	0	1
11	0	0	0	0
10	0	0	0	1

A = 1

$D_1D_0 \diagdown Q_1Q_0$	00	01	11	10
00	x	x	x	x
01	0	0	0	1
11	0	0	0	0
10	0	0	0	1

Quedando en la siguiente ecuacion:

$$Q_0^{t+1} = \tilde{D}_1 Q_1^t \tilde{Q}_0^t + \tilde{D}_0 Q_1^t \tilde{Q}_0^t$$

2. Maquina cambio de jugador

Realizamos el diagrma de una maquina para el cambio de los jugadores.

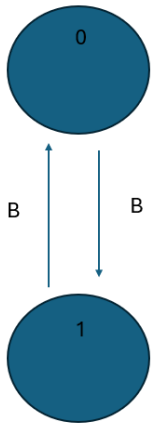

Figura 2: Diagrama de flujo jugadores

Hacemos su tabla de verdad:

B	Q_0^t	Q_0^{t+1}
1	0	1
1	1	0

Hacemos los mapas de karnaugh:

$B \diagdown Q_0^t$	0	1
0	x	x
1	1	0

3. Redactamos el código en Verilog.

Una vez tenemos nuestro circuito planificado procedemos a escribirlo en Verilog usando como base la tarea anterior.

3.1. Main.v

El primer módulo actúa como el módulo principal que integra todos los submódulos necesarios. Toma un reloj de entrada, una señal de 2 bits de los interruptores y señales de entrada de botones. Controla las salidas para un display de 7 segmentos y sus ánodos. Instancia un divisor de reloj para generar una frecuencia más baja y un módulo para visualizar números en el display. Además, maneja los estados del juego y del jugador.

```
`timescale 1ns / 1ps

module Main(
    input logic clk,
    input logic [1:0] sw,
    input logic btnR, btnL, btnU, // btnR -> Confirmación, btnL -> Empezar, btnU -> Reset
    output logic [6:0] seg,
    output logic [3:0] an
);

    typedef enum logic [1:0] {M0, M1, M2} p_states;
    typedef enum logic [1:0] {S0, S1, S2, S3} g_states;
    p_states p_state, p_nextstate;
    g_states g_state, g_nextstate;

    parameter DIV = 100_000;
    wire slw_clk;
    logic fin;
    logic [1:0] j_exe, n_jug, d_poz, g_jug;

    Clock_Divider #(DIV) Clock_Divider_inst (clk, slw_clk);

    // Sincronizacion de btnR
    logic btnR_sync, btnR_sync2;

    always_ff @(posedge clk) begin
        btnR_sync <= btnR;
        btnR_sync2 <= btnR_sync;
    end
```

Figura 3: Código Main1

```
    // Estados de jugador
    always_ff @(posedge btnR_sync2 or posedge btnU or posedge btnL)
        if (btnU) p_state <= M0;
        else if (btnL) p_state <= M1;
        else p_state <= p_nextstate;

    always_comb begin
        case (p_state)
            M0: p_nextstate = M0;
            M1: if (fin) p_nextstate = M0;
                else if (btnR_sync2) p_nextstate = M2;
            M2: if (fin) p_nextstate = M0;
                else if (btnR_sync2) p_nextstate = M1;
            default: p_nextstate = M0;
        endcase
    end
```

Figura 4: Código Main2

```
    // Estados de partida
    always_ff @(posedge btnR_sync2 or posedge btnU or posedge btnL)
        if (btnU) g_state <= S0;
        else if (btnL) g_state <= S2;
        else if (sw[1] & sw[0] & btnR_sync2) g_state <= S0;
        else g_state <= g_nextstate;

    always_comb begin
        case (g_state)
            S0: if (btnL) g_nextstate = S2;
                else g_nextstate = S0;
            S1: if (sw[1] & btnR) g_nextstate = S0;
                else if (~sw[1] & sw[0] & btnR) g_nextstate = S2;
                else g_nextstate = S1;
            S2: if (sw[1] & ~sw[0] & btnR) g_nextstate = S1;
                else if (~sw[1] & sw[0] & btnR) g_nextstate = S3;
                else g_nextstate = S2;
            S3: if (sw[1] & ~sw[0] & btnR) g_nextstate = S2;
                else if (~sw[1] & sw[0] & btnR) g_nextstate = S0;
                else g_nextstate = S3;
            default: g_nextstate = S0;
        endcase
    end

    assign fin = (g_state == S0);
    VisualizarNum VisualizarNum_inst (slw_clk, p_state, g_state, seg, an);

endmodule
```

Figura 5: Código Main3

3.2. *Clock_divider.v*

Este módulo divide la frecuencia del reloj de entrada para generar una señal de reloj más lenta. Esto es útil para reducir la velocidad de las operaciones que no requieren alta frecuencia. Tiene como entrada una señal de reloj y como salida la señal de reloj dividida.

```
`timescale 1ns / 1ps

module Clock_Divider #(integer DIV = 1)(
    input wire clk,
    output reg clk_out
);

    logic [31:0] counter;

    always_ff @(posedge clk) begin
        if (counter == (DIV-1)) begin
            counter <= 0;
            clk_out <= ~clk_out;
        end else begin
            counter <= counter + 1;
        end
    end
endmodule
```

Figura 6: Código Clock_divider.v

3.3. *VisualizarNum.v*

Este módulo maneja la visualización de números en el display de 7 segmentos. Coordina la salida de varios submódulos (JuegoExe, NumJugador, DineroPozo, Ganador) y los combina en el controlador de 7 segmentos. Toma como entrada un reloj y los estados del jugador y del juego, y controla los segmentos del display y sus ánodos.

```
`timescale 1ns / 1ps

module VisualizarNum(
    input logic clk,
    input logic [1:0] p_state,
    input logic [1:0] g_state,
    output logic [6:0] seg,
    output logic [3:0] an
);

    logic [6:0] seg_aux_1, seg_aux_2, seg_aux_3, seg_aux_4;

    JuegoExe numero_1 (p_state, seg_aux_1);
    NumJugador numero_2 (p_state, g_state, seg_aux_2);
    DineroPozo numero_3 (p_state, g_state, seg_aux_3);
    Ganador numero_4 (p_state, g_state, seg_aux_4);

    SevenSegController salida (clk,
                               seg_aux_1,
                               seg_aux_2,
                               seg_aux_3,
                               seg_aux_4,
                               an,
                               seg
                               );

endmodule
```

Figura 7: Código VisualizarNum.v

3.4. *JuegoExe.v*

Este submódulo genera el valor que se muestra en el display de 7 segmentos basado en el estado del jugador. Toma el estado del jugador como entrada y proporciona una señal de salida para los segmentos del display.

3.5. *NumJugador.v*

Este submódulo muestra el número del jugador en el display de 7 segmentos basado en los estados del jugador y del juego. Toma ambos estados como entradas y proporciona una señal de salida para los segmentos del display.

3.6. *DineroPozo.v*

Este submódulo muestra el dinero en el pozo en el display de 7 segmentos basado en los estados del jugador y del juego. Toma ambos estados como entradas y proporciona una señal de salida para los segmentos del display.

```
`timescale 1ns / 1ps

module DineroPozo(
    input logic [1:0] p_state, g_state,
    output logic [6:0] seg
);

    always_comb begin
        if (p_state != 2'b00) begin
            case (g_state)
                2'b00: seg = 7'b1000000;
                2'b01: seg = 7'b1111001;
                2'b10: seg = 7'b0100100;
                2'b11: seg = 7'b0110000;
                default: seg = 7'b1111111;
            endcase
        end else begin
            seg = 7'b1111111;
        end
    end

endmodule
```

Figura 8: Código Dinero en el Pozo.v

3.7. *Ganador.v*

Este submódulo indica si hay un ganador en el display de 7 segmentos basado en los estados del jugador y del juego. Toma ambos estados como entradas y proporciona una señal de salida para los segmentos del display.

3.8. *SevenSegController.v*

Este módulo controla la multiplexación de los dígitos en el display de 7 segmentos para mostrar diferentes valores en cada dígito de manera secuencial. Toma como entrada un reloj y los valores para cada dígito del display, y controla los ánodos y los segmentos del display.

```
`timescale 1ns / 1ps

module Ganador(
    input logic [1:0] p_state, g_state,
    output logic [6:0] seg
);

    always_comb begin
        if (g_state == 2'b00) begin
            case (p_state)
                2'b00: seg = 7'b1111111;
                2'b01: seg = 7'b0100100;
                2'b10: seg = 7'b1111001;
                2'b11: seg = 7'b1111111;
                default: seg = 7'b1111111;
            endcase
        end else begin
            seg = 7'b1111111;
        end
    end

endmodule
```

Figura 9: Código Ganador.v

```
1  `timescale 1ns / 1ps
2
3  module SevenSegController(
4      input wire clk,
5      input wire [6:0] Y0,Y1,Y2,Y3,
6      output reg [3:0] an,
7      output reg [6:0] seg
8  );
9
10     reg [2:0] contador;
11
12     always @(posedge clk) begin
13         contador <= contador + 1;
14
15         if (contador == 3'b0) begin
16             an <= 4'b1110;
17             seg <= Y3;
18         end else if (contador == 3'b1) begin
19             an <= 4'b1101;
20             seg <= Y2;
21         end else if (contador == 3'b10) begin
22             an <= 4'b1011;
23             seg <= Y1;
24         end else if (contador == 3'b11) begin
25             contador <= 3'b0;
26             an <= 4'b0111;
27             seg <= Y0;
28         end
29     end
30 endmodule
```

Figura 10: Código SevenSegController.v

4. Ejemplos con la tarjeta

Ahora mostraremos un ejemplo de juego con la tarjeta. Con el orden de información de izquierda a derecha del display como: juego en curso (1 o 0), turno del jugador (1 o 2), Dinero en el pozo (1, 2 o 3) y ganador de la partida (1 o 2). Si algún led no se enciende es porque el estado del juego no es congruente con la información dada. Para avanzar el progreso del juego se debe accionar siempre el botón de acción.

1. La tarjeta inicia de base, apretamos el botón izquierdo que comienza un nuevo juego.
2. El jugador uno recibe un 2 así que toma $1 del pozo.
3. El jugador dos recibe un 1 así que pone $1 en el pozo.
4. El jugador uno recibe otro 1 así que pone $1 en el pozo.
5. El jugador dos obtiene un 3 reclamando todo el pozo.
6. El jugador dos es el ganador. Terminando el juego.

(Las imágenes se leen de arriba a abajo)

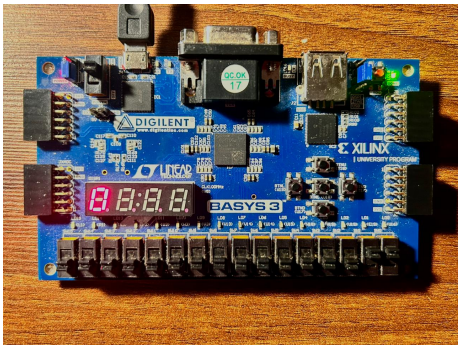

Figura 11: La tarjeta inicia de base, apretamos el botón izquierdo que comienza un nuevo juego

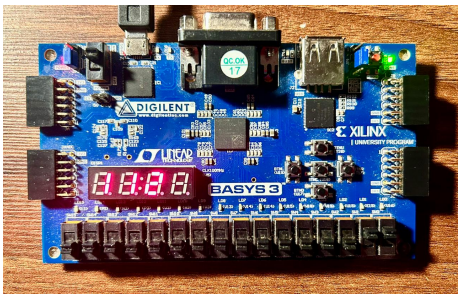

Figura 12: El jugador uno recibe un 2 así que toma $1 del pozo.

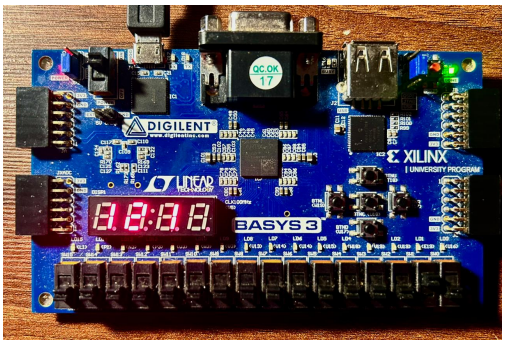

Figura 13: El jugador dos recibe un 1 así que pone $1 en el pozo.

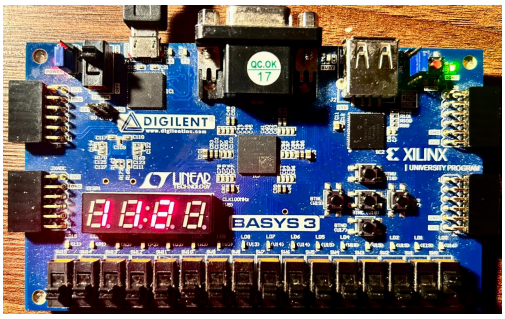

Figura 14: El jugador uno recibe otro 1 así que pone $1 en el pozo.

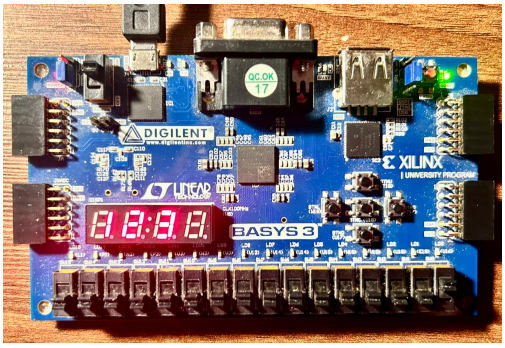

Figura 15: El jugador dos obtiene un 3 reclamando todo el pozo.

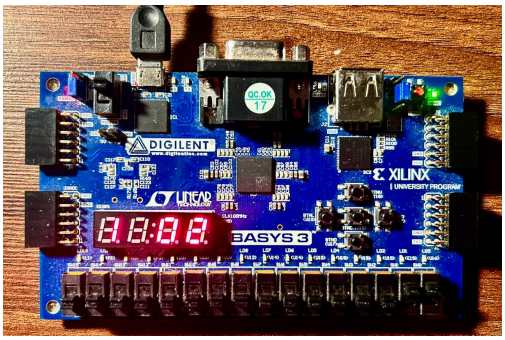

Figura 16: El jugador dos es el ganador. Terminando el juego.