



SISTEMAS ELECTRONICOS DIGITALES

VHDL

BARRERA CONTROL PEAJE VEHICULOS

TRABAJO REALIZADO POR:

JUAN IGNACIO MARTIN MORENO 55975

IÑIGO CASTELLS CASTRO 55790

ALVARO RIBALDA HERNANDEZ 56555

ÍNDICE

INTRODUCCION DEL PROYECTO.....	
PROPUESTA Y FUNCIONAMIENTO.....	
ESTRUCTURA PRINCIPAL DEL CODIGO.....	
1. 1EDGE DETECTOR.....	
1.2 SHYNCRONIZER.....	
1.3 FSM.....	
1.4 COUNTER.....	
1.5 TOP	
GITHUB Y VIDEO DE FUNCIONAMIENTO.....	

INTRODUCCIÓN

Propuesta

El presente proyecto consiste en el diseño y desarrollo de un sistema de peaje automático implementado en una FPGA Nexys A7 utilizando el lenguaje de descripción de hardware VHDL y el software Vivado. El objetivo principal es simular el funcionamiento real de un peaje automatizado, desde la detección del tipo de vehículo hasta el procesamiento de los pagos y la gestión de barreras, asegurando una experiencia eficiente y funcional.

Funcionamiento

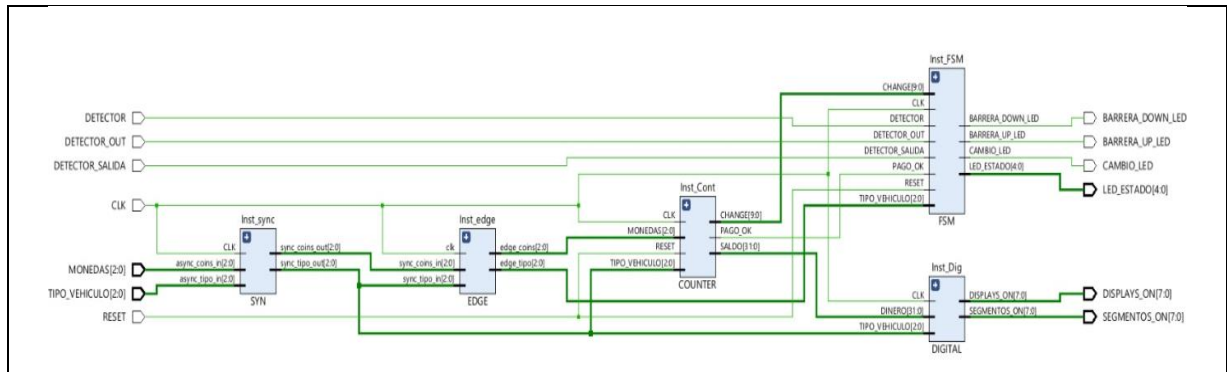
El sistema permite al usuario seleccionar el tipo de vehículo entre Coche, SUV (Jeep) o Bus, siendo estos, tipos de vehículo visualizadas a través de los displays de siete segmentos. En caso de error durante la selección, el usuario puede reiniciar el proceso mediante el botón RESET. Las tarifas están definidas según el tipo de vehículo: un euro para Coche, un euro con cincuenta céntimos para SUVs y dos euros para Buses.

La máquina acepta únicamente monedas de 1 €, 0,50 € y 2 €, y va mostrando la cantidad ingresada en el display. Si el dinero entregado supera la tarifa requerida, el sistema calcula y devuelve el cambio automáticamente. Tras completar el pago, se activa la barrera permitiendo el paso del vehículo, y el sistema vuelve al modo de reposo, listo para el siguiente usuario.



DIAGRAMA ENTIDADES

PROYECTO



El diagrama de entidades que se presenta a continuación constituye el punto de partida para comprender la arquitectura de nuestro sistema de control de acceso vehicular. Esta representación visual nos permite identificar los componentes principales y sus interconexiones, sentando las bases para un análisis más detallado de su funcionamiento.

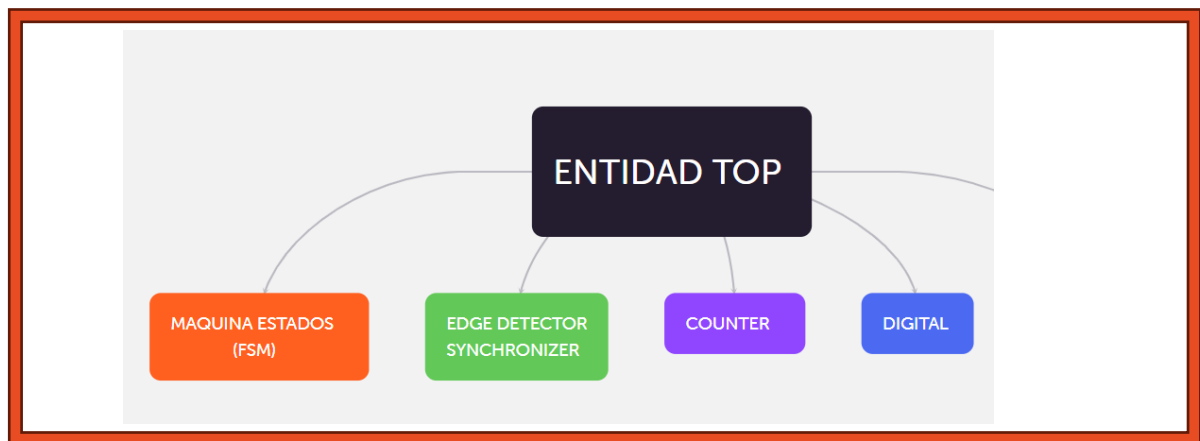
PUNTO INICIO:

- **FSM (Máquina de Estados Finita):** Es el núcleo del sistema, responsable de controlar la secuencia de operaciones de acuerdo con las condiciones detectadas en el entorno.
- **SYN:** Garantiza la sincronización de las señales de entrada y salida, permitiendo que las operaciones se ejecuten de manera coordinada y sin interferencias.
- **EDGE:** Se encarga de identificar los flancos ascendentes de las señales de reloj, facilitando la sincronización precisa de eventos dentro del sistema.
- **COUNTER** Funciona como un contador para gestionar temporizaciones o contabilizar eventos relevantes en el sistema.
- **DIGITAL:** Este módulo gestiona la conversión de señales digitales, adecuándolas para su interpretación visual o para el control de dispositivos externos vinculados al sistema.

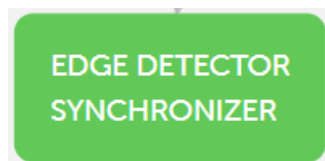
Estructura principal del código

En este apartado se explicará cada una de las entidades desarrolladas

Para explicar de forma adecuada el proyecto, comenzaremos describiendo las diferentes entidades que lo componen. Posteriormente, se muestra cada una de estas entidades y su función específica dentro del desarrollo del proyecto, destacando su utilidad y el papel que desempeñan en el conjunto



1- TRATAMIENTO DE LA SEÑAL



SYNCHRONIZER

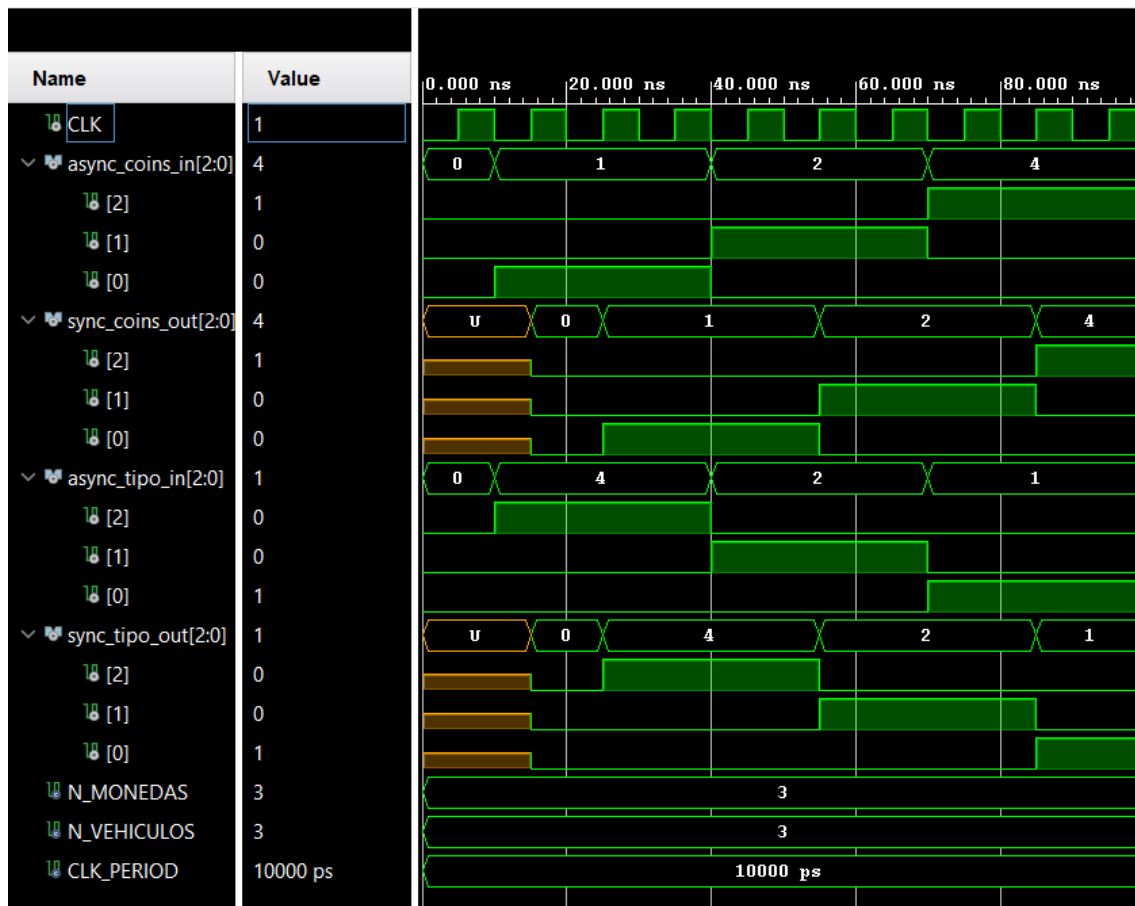
Los circuitos digitales funcionan de forma síncrona, lo que implica que los cambios en los valores de las señales se realizan únicamente durante el flanco ascendente de la señal de reloj (en nuestro proyecto denominado CLK).

CODIGO FUENTE:

En el proyecto, las entradas (monedas y tipo de vehículo) son asíncronas. Para sincronizar estas señales con el reloj, hacemos uso del “Sincronizador”. Su código es:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity SYN is
5     GENERIC (
6         N_MONEDAS: POSITIVE;
7         N_VEHICULOS: POSITIVE
8     );
9     PORT (
10         CLK : in std_logic;
11         async_coins_in : in std_logic_vector(N_MONEDAS - 1 downto 0);
12         sync_coins_out : out std_logic_vector(N_MONEDAS - 1 downto 0);
13         async_tipo_in : in std_logic_vector(N_VEHICULOS - 1 downto 0);
14         sync_tipo_out : out std_logic_vector(N_VEHICULOS - 1 downto 0)
15     );
16 end SYN;
17
18 architecture BEHAVIORAL of SYN is
19     SIGNAL SREG_1_COINS: STD_LOGIC_VECTOR(N_MONEDAS - 1 downto 0);
20     SIGNAL SREG_1_TIPO: STD_LOGIC_VECTOR (N_VEHICULOS - 1 downto 0);
21
22     SIGNAL SREG_2_COINS: STD_LOGIC_VECTOR(N_MONEDAS - 1 downto 0);
23     SIGNAL SREG_2_TIPO: STD_LOGIC_VECTOR (N_VEHICULOS - 1 downto 0);
24
25 begin
26
27     reg_1:PROCESS (CLK)
28     BEGIN
29         IF rising_edge(CLK) then
30             SREG_1_COINS <= async_coins_in;
31             SREG_1_TIPO <= async_tipo_in;
32         END IF;
33     END PROCESS;
34
35     reg_2:PROCESS (CLK)
36     BEGIN
37         IF rising_edge(CLK) then
38             SREG_2_COINS <= SREG_1_COINS;
39             SREG_2_TIPO <= SREG_1_TIPO;
40         END IF;
41     END PROCESS;
42
43     sync_coins_out <= SREG_2_COINS;
44     sync_tipo_out <= SREG_2_TIPO;
45
46 end BEHAVIORAL;
```

Para verificar el funcionamiento del sincronizador, hemos utilizado su respectivo el testbench, para simular las entradas de monedas y de tipo de vehiculo



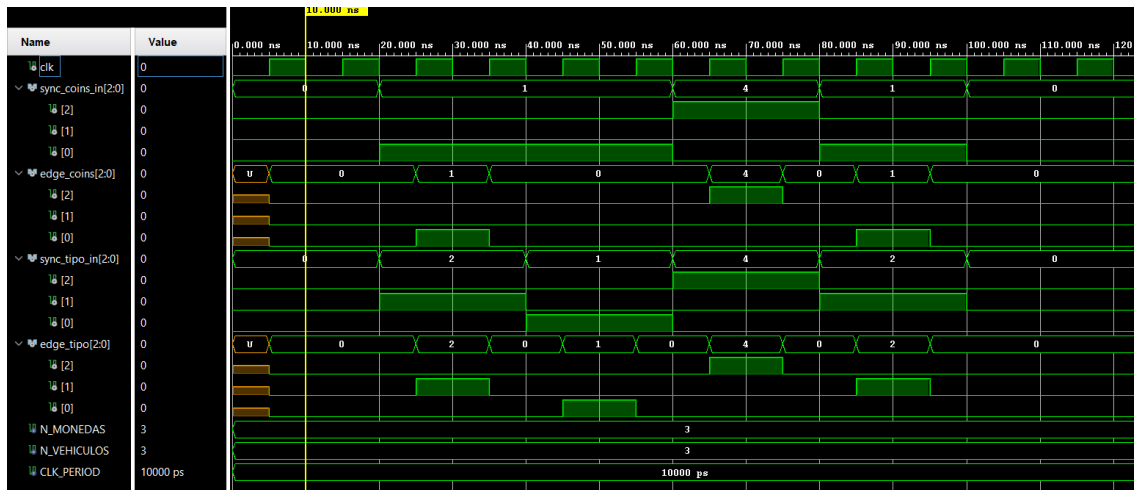
EDGE DETECTOR

Detecta transiciones o cambios en las señales de entrada sync_coins_in y sync_tipo_in. Cuando ocurre un cambio, genera una salida correspondiente en edge_coins o edge_tipo, respectivamente, y reinicia estas salidas a zeros si no hay cambios. Así, actúa como un detector de flancos o transiciones para ambas señales.

CODIGO FUENTE

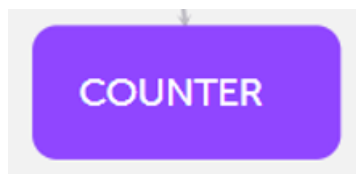
```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity EDGE is
5     generic(
6         N_MONEDAS: POSITIVE;
7         N_VEHICULOS: POSITIVE
8     );
9     port (
10         clk : in std_logic;
11         sync_coins_in : in std_logic_vector(N_MONEDAS - 1 downto 0);
12         edge_coins : out std_logic_vector(N_MONEDAS - 1 downto 0);
13         sync_tipo_in : in std_logic_vector(N_VEHICULOS - 1 downto 0);
14         edge_tipo : out std_logic_vector(N_VEHICULOS - 1 downto 0)
15     );
16 end EDGE;
17
18 architecture Behavioral of EDGE is
19
20     signal DATA_1 : STD_LOGIC_VECTOR(N_MONEDAS - 1 downto 0) := (OTHERS => '0');
21     signal DATA_2 : STD_LOGIC_VECTOR(N_VEHICULOS - 1 downto 0) := (OTHERS => '0');
22
23     begin
24     process (CLK)
25
26         begin
27             if rising_edge (CLK) then
28                 --Procesamiento de monedas
29                 if sync_coins_in /= DATA_1 then
30                     edge_coins <= sync_coins_in;
31                 else
32                     edge_coins <= (OTHERS => '0');
33                 end if;
34                 DATA_1 <= sync_coins_in;
35                 --Procesamiento de tipo de vehiculo
36                 if sync_tipo_in /= DATA_2 then
37                     edge_tipo <= sync_tipo_in;
38                 else
39                     edge_tipo <= (others => '0');
40                 end if;
41                 DATA_2 <= sync_tipo_in;
42             end if;
43
44         end process;
45     end Behavioral;
```

Para verificar el funcionamiento del Edge , hemos utilizado el testbench :



Este testbench simula el funcionamiento de un detector de transiciones sincronizado con el reloj. Cada vez que las señales de entrada (sync_coins_in y sync_tipo_in) cambian, las salidas correspondientes (edge_coins y edge_tipo) generan un impulso de un ciclo de reloj para reflejar ese cambio. Si no hay transiciones en las entradas, las salidas permanecen en cero. Todo el proceso está sincronizado con los flancos de subida de la señal de reloj (clk), verificando que el detector responde correctamente a los cambios en las entradas.

2- COUNTER



Este diseño implementa un sistema contador que gestiona pagos mediante monedas para distintos tipos de vehículos, determinando tarifas y cambios. Detecta las monedas insertadas, actualiza el saldo y calcula si el monto es suficiente para cubrir la tarifa según el tipo de vehículo (coche, jeep o bus). Si el saldo cumple con la tarifa, activa la señal de pago válido (PAGO_OK), calcula el cambio y reinicia el saldo. Además, resetea todos los valores cuando se activa la señal de reinicio (RESET).

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity COUNTER is
6     Generic(
7         N_MONEDAS: POSITIVE:=3;
8         N_VEHICULOS: POSITIVE:= 3;
9         SIZE_CUENTA: POSITIVE
10    );
11
12    Port(
13        CLK: in std_logic;
14        RESET: in std_logic;
15        MONEDAS: in std_logic_vector(N_MONEDAS - 1 downto 0);
16        TIPO_VEHICULO: in std_logic_vector(N_VEHICULOS - 1 downto 0);
17        PAGO_OK: out std_logic;
18        SALDO: out integer;
19        CHANGE: out std_logic_vector(SIZE_CUENTA - 1 downto 0)
20    );
21 end COUNTER;
22
23 architecture Behavioral of COUNTER is
24     -- Constantes para definir tarifas (en céntimos)
25     constant TARIFA_COCHE : integer := 100; -- 1 euro
26     constant TARIFA_JEEP : integer := 150; -- 1.5 euros
27     constant TARIFA_BUS : integer := 200; -- 2 euros
28
29     -- Señales internas
30     signal saldo_actual : integer := 0;
31     signal tarifa : integer := 0;
32     signal pago_completo : std_logic := '0';
33     signal cambio : integer := 0;
34     signal MONEDAS_ANTERIOR : std_logic_vector(N_MONEDAS - 1 downto 0) := (others => '0');
35
36 begin
37
38     -- Proceso principal para controlar el contador
39     process(CLK, RESET)
40     begin
41         if RESET = '0' then

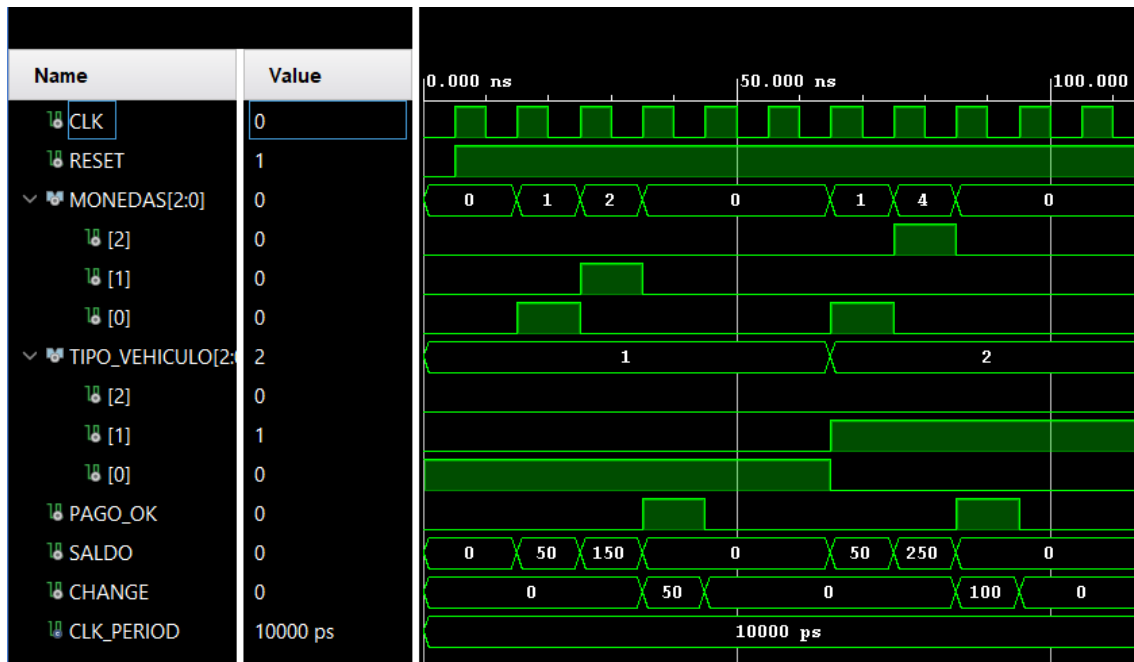
```

```

42         saldo_actual <= 0;
43         tarifa <= 0;
44         pago_completo <= '0';
45         cambio <= 0;
46         MONEDAS_ANTERIOR <= (others => '0');
47     elsif rising_edge(CLK) then
48         -- Actualización del saldo según las monedas (Detecta flanco de subida en MONEDAS)
49         if MONEDAS /= MONEDAS_ANTERIOR then
50             -- Si hay cambio en la señal de MONEDAS
51             if MONEDAS = "001" then -- Más 50 céntimos
52                 saldo_actual <= saldo_actual + 50;
53             elsif MONEDAS = "010" then -- Más 1 euro
54                 saldo_actual <= saldo_actual + 100;
55             elsif MONEDAS = "100" then -- Más 2 euros
56                 saldo_actual <= saldo_actual + 200;
57             end if;
58         end if;
59
60         -- Actualizar MONEDAS_ANTERIOR para la siguiente comparación
61         MONEDAS_ANTERIOR <= MONEDAS;
62
63         -- Determinación de la tarifa según el tipo de vehículo
64         if TIPO_VEHICULO = "001" then -- Coche
65             tarifa <= TARIFA_COCHE;
66         elsif TIPO_VEHICULO = "010" then -- Jeep
67             tarifa <= TARIFA_JEEP;
68         elsif TIPO_VEHICULO = "100" then -- Bus
69             tarifa <= TARIFA_BUS;
70         else
71             tarifa <= 0;
72         end if;
73
74         -- Verificación de pago
75         if tarifa > 0 and saldo_actual >= tarifa then
76             pago_completo <= '1';
77             cambio <= saldo_actual - tarifa;
78             saldo_actual <= 0; -- Reiniciar saldo tras el pago
79         else
80             pago_completo <= '0';
81             cambio <= 0;
82         end if;
83     end if;
84 end process;
85
86     -- Asignación de salidas
87     PAGO_OK <= pago_completo;
88     CHANGE <= std_logic_vector(to_unsigned(cambio, SIZE_CUENTA));
89     SALDO <= saldo_actual;
90
91 end Behavioral;
92

```

A través de la simulación de su testbench se observa el correcto funcionamiento y la cuenta correcta de monedas



El testbench verifica el funcionamiento del contador simulando la inserción de monedas y el cálculo de tarifas según el tipo de vehículo. A medida que se ingresan monedas en la señal MONEDAS, el saldo (SALDO) se incrementa proporcionalmente, y al seleccionar un vehículo mediante TIPO_VEHICULO, el sistema calcula la tarifa correspondiente. Si el saldo acumulado es suficiente para cubrir la tarifa, se activa la señal de pago válido (PAGO_OK) y se genera el cambio correspondiente en la señal CHANGE. Además, la señal RESET permite reiniciar el saldo y todas las operaciones, verificando que el sistema responde correctamente en diferentes escenarios de operaciones

3- MAQUINA DE ESTADOS



Para el diagrama de estados comenzamos con una distribución centrada en Estado S0 (Inicio): El sistema espera a que un vehículo sea detectado.

Estado S1: El vehículo ha sido detectado y se selecciona un tipo de vehículo.

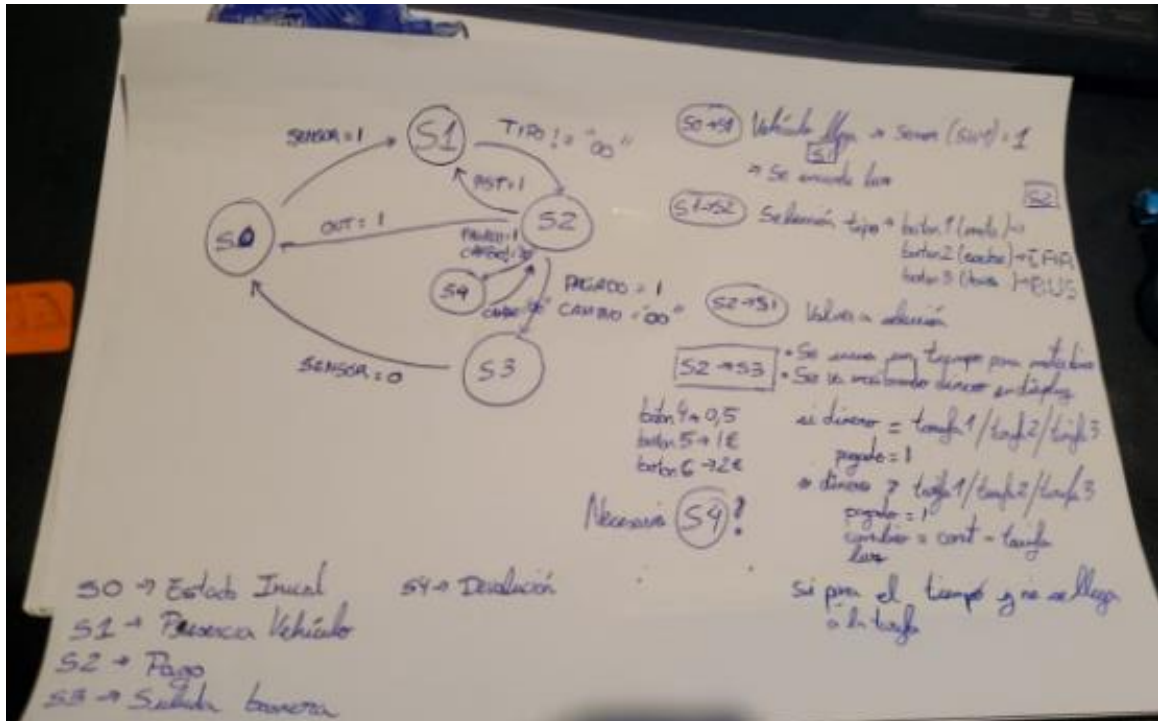
Estado S2: Se verifica si se ha realizado el pago correctamente.

Estado S3: La barrera está levantada para permitir el paso del vehículo.

Estado S4: El vehículo ha pasado y se está marchando, finalizando el ciclo.

Diseño prototipo (diagrama abstracto nivel 3)

Idea inicial presentada al profesor:



Hasta que finalmente se llegó a una máquina de estados robusta y efectiva con el siguiente código

CODIGO FUENTE

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4 entity FSM is
5     Generic(
6         N_MONEDAS: POSITIVE;
7         N_VEHICULOS: POSITIVE;
8         N_ESTADOS: POSITIVE;
9         DISPLAYS: POSITIVE;
10        SEGMENTOS: POSITIVE;
11        SIZE_CUENTA: POSITIVE
12    );
13    Port(
14        CLK : in STD_LOGIC;
15        RESET : in STD_LOGIC;
16        DETECTOR : in STD_LOGIC;
17        DETECTOR_OUT: in STD_LOGIC;
18        DETECTOR_SALIDA: in STD_LOGIC;
19        PAGO_OK : in STD_LOGIC;
20        BARRERA_UP_LED : out STD_LOGIC;
21        BARRERA_DOWN_LED : out STD_LOGIC;
22        CAMBIO_LED: out STD_LOGIC ;
23        TIPO_VEHICULO: in STD_LOGIC_VECTOR (N_VEHICULOS - 1 downto 0);
24        CHANGE: in std_logic_vector(SIZE_CUENTA - 1 downto 0);
25        LED_ESTADO : out STD_LOGIC_VECTOR(N_ESTADOS -1 downto 0)
26    );
27 end FSM;
28
29 architecture Behavioral of FSM is
30
31     type STATES is (S0,S1,S2,S3,S4);
32     signal CURRENT_STATE: STATES := S0;
33     signal NEXT_STATE: STATES;
34     signal enable : integer := 0; -- Sirve para saber en que estado me encuentro
35 begin
36

```

```

37 -----REGISTRO DE ESTADOS-----
38 process(RESET,CLK)
39 begin
40     if RESET = '0' then
41         CURRENT_STATE <= S0;
42     elsif rising_edge(clk) then
43         CURRENT_STATE <= NEXT_STATE;
44         case CURRENT_STATE is
45             when S0 => enable <= 0;
46             when S1 => enable <= 1;
47             when S2 => enable <= 2;
48             when S3 => enable <= 3;
49             when S4 => enable <= 4;
50         end case;
51     end if;
52 end process;
53
54 -----CAMBIOS DE ESTADO-----
55 nextstate: process(CLK,CURRENT_STATE)
56 begin
57     NEXT_STATE <= CURRENT_STATE;
58     case CURRENT_STATE is
59         when S0 =>
60             if DETECTOR = '1' then --Hay un vehiculo
61                 NEXT_STATE <= S1;
62             end if;
63         when S1 =>
64             if TIPO_VEHICULO /= "000" then --Se ha seleccionado un tipo de vehiculo
65                 NEXT_STATE <= S2;
66             end if;
67         when S2 =>
68             if PAGO_OK = '1' then --Se ha realizado el pago correctamente
69                 NEXT_STATE <= S3;
70             end if;
71         when S3 =>
72
73
74             if DETECTOR_OUT='1' then --Detecta que la barrera esta arriba
75                 NEXT_STATE <= S4;
76             end if;

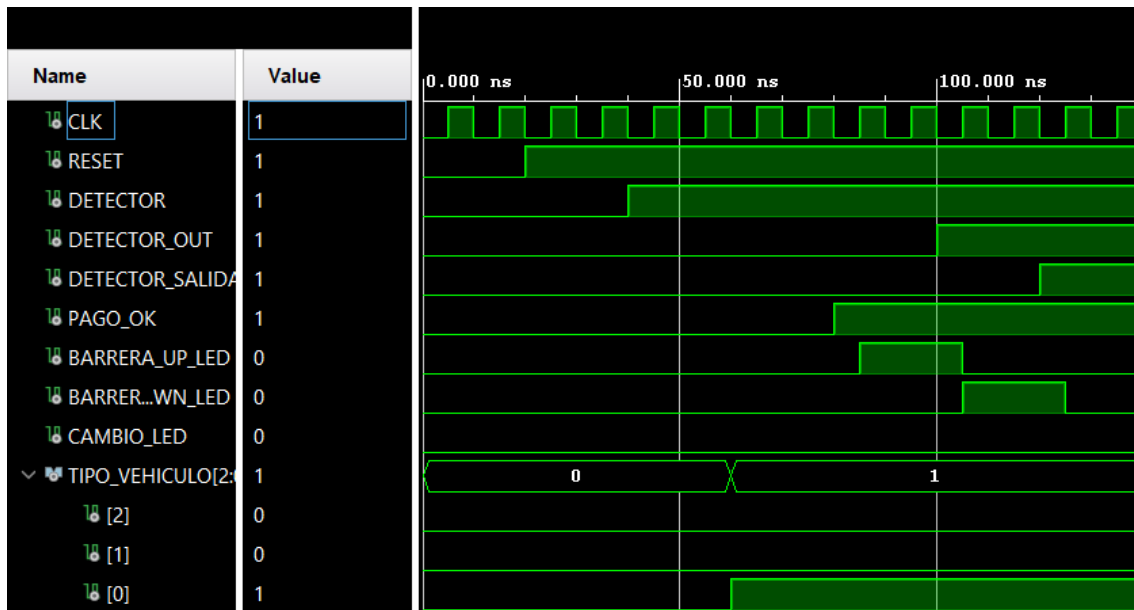
```

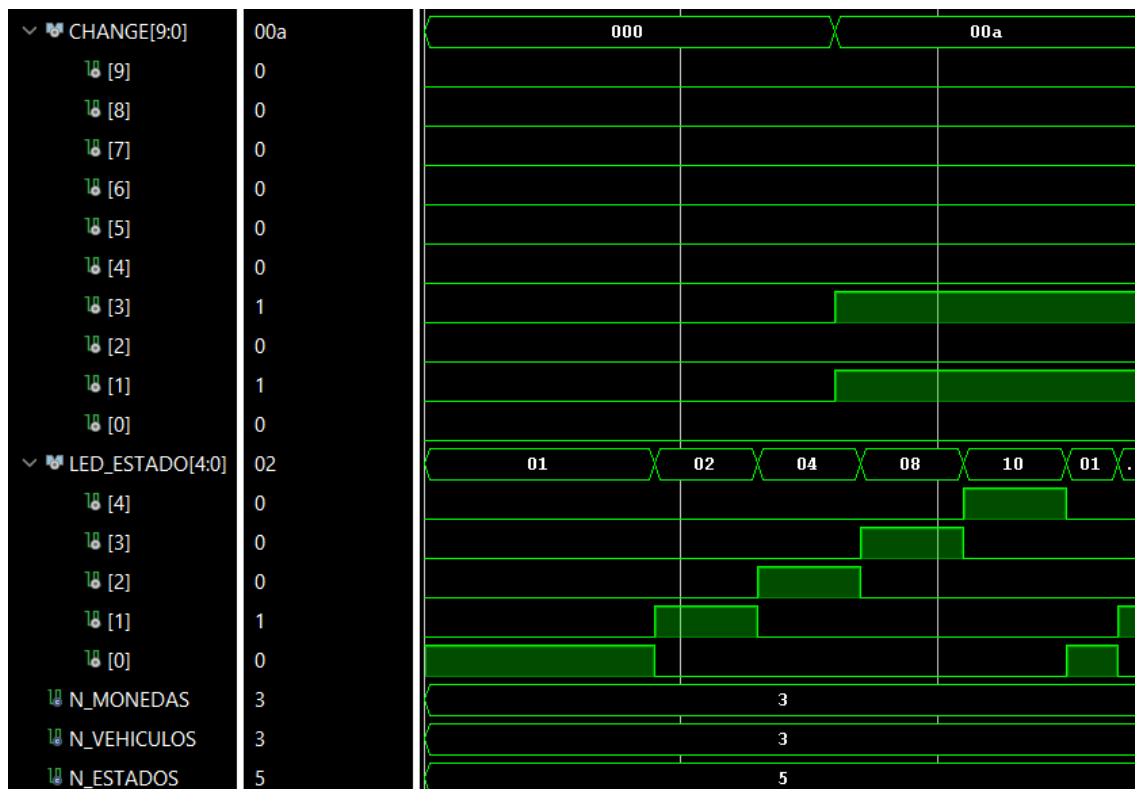
```

77
78         when s4 =>
79             if DETECTOR_SALIDA='1' then          --Vehiculo se ha marchado
80                 NEXT_STATE <= s0;
81             end if;
82         end case;
83     end process;
84
85     -----SALIDAS DE ESTADO-----
86     out_control: process(CURRENT_STATE)
87     begin
88         LED_ESTADO <= (others => '0');
89         case CURRENT_STATE is
90             when S0 =>
91                 CAMBIO_LED <= '0';
92                 BARRERA_UP_LED <= '0';
93                 BARRERA_DOWN_LED <= '0';
94                 LED_ESTADO(0) <= '1'; -- Enciende LED0
95
96             when S1 =>
97                 BARRERA_UP_LED <= '0';
98                 BARRERA_DOWN_LED <= '0';
99                 LED_ESTADO(1) <= '1'; -- Enciende LED1
100
101             when S2 =>
102                 if CHANGE /= "0000000000" then
103                     CAMBIO_LED <= '1';
104                 end if;
105                 BARRERA_UP_LED <= '0';
106                 BARRERA_DOWN_LED <= '0';
107                 LED_ESTADO(2) <= '1'; -- Enciende LED2
108
109             when S3 =>
110                 BARRERA_UP_LED <= '1';
111                 BARRERA_DOWN_LED <= '0';
112                 LED_ESTADO(3) <= '1'; -- Enciende LED3
113
114             when S4 =>
115                 BARRERA_UP_LED <= '0';
116                 BARRERA_DOWN_LED <= '1';
117                 LED_ESTADO(4) <= '1'; -- Enciende LED4
118                 CAMBIO_LED <= '0';

```

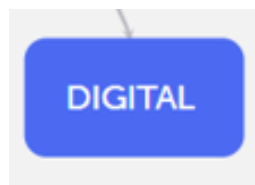
Para comprobar el funcionamiento de la máquina de estados hemos realizado un testbench específico





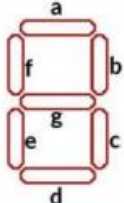
Observando cómo progresa a través de los diferentes estados, tal como se había planteado se llega a la conclusión de que es efectivo

4- DIGITAL



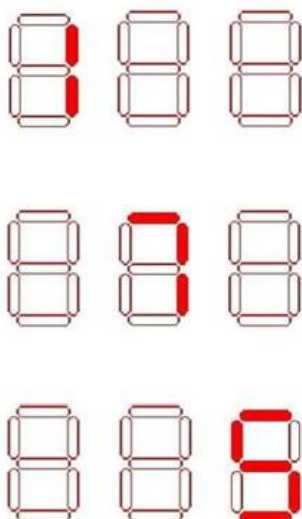
Este código controla displays y segmentos LED para mostrar información basada en el tipo de vehículo y el dinero ingresado. Utiliza un reloj auxiliar para alternar entre displays, mostrando diferentes letras y valores de euros y céntimos en función de las entradas.

REPRESENTACIÓN DE NÚMEROS EN EL DISPLAY DE 7 SEGMENTOS

										
A	0	1	0	0	1	0	0	0	0	0
B	0	0	0	0	0	1	1	0	0	0
C	0	0	1	0	0	0	0	0	0	0
D	0	1	0	0	1	0	0	1	0	0
E	0	1	0	1	1	1	0	1	0	1
F	0	1	1	1	0	0	0	1	0	0
G	1	1	0	0	0	0	0	1	0	0

El código controla los ocho displays de segmentos LED de la placa Nexxus A7 para representar las letras y los números, alternando la visualización en cada uno de ellos cada cuatro ciclos de reloj. Los displays comparten todos los segmentos, por lo que se encienden de manera que cada uno muestra un número diferente en cada ciclo, asegurando que se representen adecuadamente tres valores. Todos los números se muestran con un punto decimal, excepto el 0 y el 5, para diferenciar entre la parte entera (euros) y la parte decimal (céntimos) de una cantidad como "1.50", utilizando el segmento correspondiente para indicar las diferentes posiciones.

De manera visual ocurre lo siguiente, por ejemplo con el numero 175



CODIGO FUENTE

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity DIGITAL is
6     Generic(
7         N_MONEDAS : POSITIVE;
8         N_VEHICULOS : POSITIVE;
9         DISPLAYS : POSITIVE;
10        SEGMENTOS : POSITIVE
11    );
12    port(
13        CLK : in std_logic;
14        TIPO_VEHICULO : in std_logic_vector (N_VEHICULOS - 1 downto 0);
15        DINERO : in integer;
16        DISPLAYS_ON : out std_logic_vector(DISPLAYS - 1 downto 0);
17        SEGMENTOS_ON : out std_logic_vector(SEGMENTOS - 1 downto 0)
18    );
19 end DIGITAL;
20
21 architecture dataflow of DIGITAL is
22     signal reloj_auxiliar : std_logic := '0';
23
24
25     type seg_array is array(0 to 21) of std_logic_vector(SEGMENTOS - 1 downto 0);
26     constant SEGMENT_TABLE : seg_array := (
27         "10000001", -- 0
28         "01001111", -- 1
29         "00010010", -- 2
30         "00000110", -- 3
31         "01001100", -- 4
32         "10100100", -- 5
33         "00100000", -- 6
34         "00001111", -- 7
35         "00000000", -- 8
36         "00000100", -- 9
37         "00000001", -- 0 con punto decimal 10
38         "00100100", -- 5 con punto decimal 11
39         "10001000", --A 12
40         "10110001", --C 13
41         "10001000", --R 14
42
43         "10000000", --B 15
44         "11000001", --U 16
45         "10100100", --S 17
46         "11000111", --J 18
47         "10110000", --E 19
48         "10011000", --P 20
49         "11111111" --Nada 21
50     );
51
52 begin
53     -- Generador de reloj auxiliar
54     process(CLK)
55         subtype ciclos is integer range 0 to 10**8;
56         variable contaje : ciclos;
57     begin
58         if rising_edge(CLK) then
59             contaje := contaje + 1;
60             if contaje = 10**5 - 1 then
61                 contaje := 0;
62                 reloj_auxiliar <= not reloj_auxiliar;
63             end if;
64         end if;
65     end process;
66
67     process(reloj_auxiliar) -- DETERMINAMOS QUE DISPLAY QUEREMOS ENCENDER EN CADA MOMENTO
68         variable display : integer := 0;
69         variable letra1, letra2, letra3, letra4, euros, cent_1, cent_2 : std_logic_vector(SEGMENTOS - 1 downto 0);
70     begin
71         if rising_edge(reloj_auxiliar) then
72             case to_integer(unsigned(TIPO_VEHICULO)) is
73                 when 1 =>
74                     letra1 := SEGMENT_TABLE(13);
75                     letra2 := SEGMENT_TABLE(12);
76                     letra3 := SEGMENT_TABLE(14);
77                     letra4 := SEGMENT_TABLE(21);
78                 when 2 =>
79                     letra1 := SEGMENT_TABLE(18);
80                     letra2 := SEGMENT_TABLE(19);
81                     letra3 := SEGMENT_TABLE(19);
82                     letra4 := SEGMENT_TABLE(20);

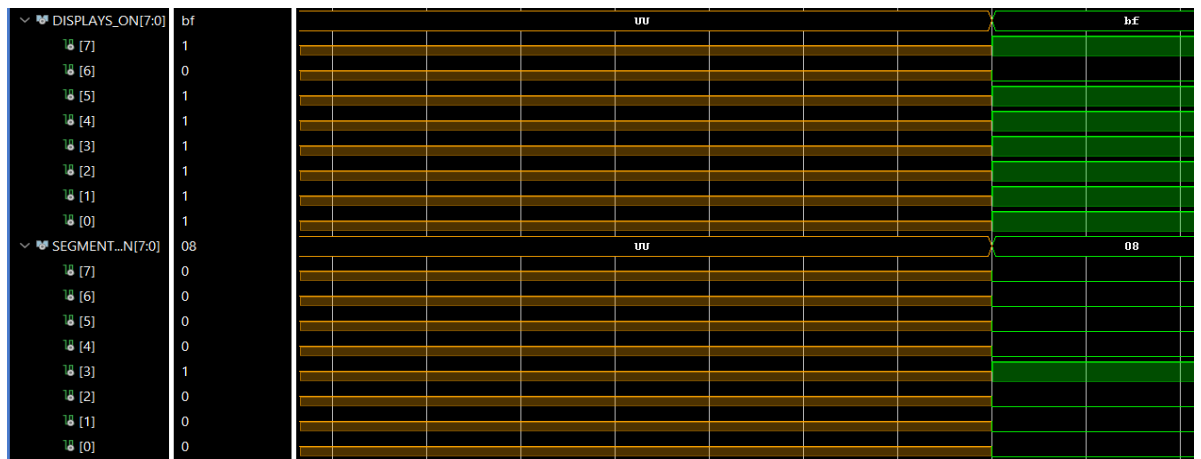
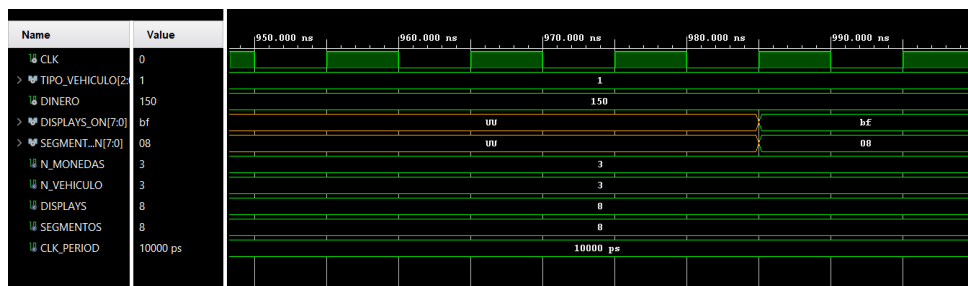
```

```


82 |         when 4 =>
83 |             letra1 := SEGMENT_TABLE(15);
84 |             letra2:= SEGMENT_TABLE(16);
85 |             letra3:= SEGMENT_TABLE(17);
86 |             letra4:= SEGMENT_TABLE(21);
87 |         when others =>
88 |             letra1 := SEGMENT_TABLE(21);
89 |             letra2:= SEGMENT_TABLE(21);
90 |             letra3:= SEGMENT_TABLE(21);
91 |             letra4:= SEGMENT_TABLE(21);
92 |             euros := "11111110"; ---
93 |             cent_1 := "11111110"; ---
94 |             cent_2 := "11111110"; ---
95 |         end case;
96 |
97 |         euros := SEGMENT_TABLE((DINERO / 100) mod 10);
98 |         cent_1:= SEGMENT_TABLE((DINERO / 10) mod 10);
99 |         cent_2:= SEGMENT_TABLE(DINERO mod 10);
100 |
101 |     display := (display + 1) mod 8; -- modificamos el display que vamos a representar
102 |     case display is
103 |     when 0 =>
104 |         DISPLAYS_ON <= "01111111";
105 |         SEGMENTOS_ON <= letra1;
106 |     when 1 =>
107 |         DISPLAYS_ON <= "10111111";
108 |         SEGMENTOS_ON <= letra2;
109 |     when 2 =>
110 |         DISPLAYS_ON <= "11011111";
111 |         SEGMENTOS_ON <= letra3;
112 |     when 3 =>
113 |         DISPLAYS_ON <= "11101111";
114 |         SEGMENTOS_ON <= letra4;
115 |     when 4 =>
116 |         DISPLAYS_ON <= "11110111";
117 |         SEGMENTOS_ON <= "11111111";
118 |     when 5 =>
119 |         DISPLAYS_ON <= "11111011";
120 |         SEGMENTOS_ON <= euros;
121 |     when 6 =>
122 |         DISPLAYS_ON <= "11111101";
123 |
124 |         SEGMENTOS_ON <= cent_1;
125 |     when others =>
126 |         DISPLAYS_ON <= "11111110";
127 |         SEGMENTOS_ON <= cent_2;
128 |     end case;
129 | end if;
130 | end process;
131 | end architecture dataflow;

```

TEST BENCH



5-TOP



ENTIDAD TOP

La entidad TOP implementa un sistema de control para un parqueo automatizado, donde se sincronizan las señales de entrada (monedas y tipo de vehículo) utilizando los componentes syn y edge. El sistema utiliza una máquina de estados finitos (FSM) para gestionar la llegada de vehículos, el pago y la activación de la barrera, además de mostrar el tipo de vehículo y el total pagado en 8 displays de 7 segmentos. El componente COUNTER acumula el monto de las monedas ingresadas, calcula el cambio y genera una señal de pago correcto (PAGO_OK), mientras que el DIGITAL controla los displays para visualización de información. Los LEDs indican el estado del sistema (barrera levantada, barrera bajada, cambio disponible) y el estado de la FSM, todo sincronizado por el reloj.

CODIGO FUENTE

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity TOP is
5     Generic(
6         N_MONEDAS: POSITIVE := 3;           --Hay 3 tipos de monedas
7         N_VEHICULOS: POSITIVE := 3;         --Hay 3 tipos de vehiculo
8         N_ESTADOS: POSITIVE := 5;           --Numero de estados de la FSM
9         SIZE_CUENTA: POSITIVE := 10;
10        DISPLAYS: POSITIVE := 8;             --Hay 8 displays en la placa
11        SEGMENTOS: POSITIVE :=8              --Cada display tien 7 segmentos y el punto
12    );
13
14
15    Port (
16        CLK : in STD_LOGIC;                  --Reloj
17        RESET : in STD_LOGIC;                --Reset
18        DETECTOR : in STD_LOGIC;              --Detecta que llega un vehiculo
19        DETECTOR_OUT: in STD_LOGIC;
20        DETECTOR_SALIDA:in STD_LOGIC;--cuando vale 0 vuelve al inicio
21        MONEDAS : in STD_LOGIC_VECTOR (N_MONEDAS - 1 downto 0);    --Entrada para las monedas
22        TIPO_VEHICULO : in STD_LOGIC_VECTOR (N_VEHICULOS - 1 downto 0); --Entrada de vehiculos
23
24        BARRERA_UP_LED : out STD_LOGIC;        --LED que muestra barrera levantada
25        BARRERA_DOWN_LED : out STD_LOGIC;      --LED que muestra barrera bajada
26        CAMBIO_LED: out STD_LOGIC;
27        DISPLAYS_ON : out std_logic_vector(DISPLAYS - 1 downto 0);
28        SEGMENTOS_ON : out std_logic_vector(SEGMENTOS - 1 downto 0);
29        LED_ESTADO: out STD_LOGIC_VECTOR (N_ESTADOS-1 downto 0)    --LEDS que representan estado de FSM
30    );
31
32 end TOP;
33
34 architecture Behavioral of TOP is
35
36
37     -----SINCRONIZADOR-----
38     component syn is
39         generic (
40             N_MONEDAS: POSITIVE;
41             N_VEHICULOS: POSITIVE
42         );
43         port (
44             clk : in std_logic;
45             async_coins_in : in std_logic_vector(N_MONEDAS - 1 downto 0);
46             sync_coins_out : out std_logic_vector(N_MONEDAS - 1 downto 0);
47             async_tipo_in : in std_logic_vector(N_VEHICULOS - 1 downto 0);
48             sync_tipo_out : out std_logic_vector(N_VEHICULOS - 1 downto 0)
49         );
50     end component;
51     signal monedasSync: std_logic_vector (N_MONEDAS - 1 downto 0); --Signal de salida de monedas del sincronizador
52     signal tipoSync: std_logic_vector (N_VEHICULOS - 1 downto 0); --Signal de salida de tipo vehiculos del sincronizador
53
54     -----EDGE-----
55     component edge is
56         generic(
57             N_MONEDAS: POSITIVE;
58             N_VEHICULOS: POSITIVE
59         );
60     port (
61         clk : in std_logic;
62         sync_coins_in : in std_logic_vector(N_MONEDAS - 1 downto 0);
63         edge_coins : out std_logic_vector(N_MONEDAS - 1 downto 0);
64         sync_tipo_in : in std_logic_vector(N_VEHICULOS - 1 downto 0);
65         edge_tipo : out std_logic_vector(N_VEHICULOS - 1 downto 0)
66     );
67     end component;
68     signal monedasEdge: std_logic_vector (N_MONEDAS - 1 downto 0); --Signal de salida de monedas del edge
69     signal tipoEdge: std_logic_vector (N_VEHICULOS - 1 downto 0); --Signal de salida de tipo vehiculos del edge
70
71
```

```

72 -----FSM-----
73 component FSM is
74     Generic(
75         N_MONEDAS: POSITIVE;
76         N_VEHICULOS: POSITIVE;
77         N_ESTADOS: POSITIVE;
78         DISPLAYS: POSITIVE;
79         SEGMENTOS: POSITIVE;
80         SIZE_CUENTA: POSITIVE
81     );
82     Port(
83         CLK : in STD_LOGIC;
84         RESET : in STD_LOGIC;
85         DETECTOR : in STD_LOGIC;
86         DETECTOR_OUT: in STD_LOGIC;
87         DETECTOR_SALIDA: in STD_LOGIC;
88         PAGO_OK : in STD_LOGIC;
89         BARRERA_UP_LED : out STD_LOGIC;
90         BARRERA_DOWN_LED : out STD_LOGIC;
91         CAMBIO_LED: out STD_LOGIC;
92         TIPO_VEHICULO: in STD_LOGIC_VECTOR (N_VEHICULOS - 1 downto 0);
93         CHANGE: in std_logic_vector(SIZE_CUENTA - 1 downto 0);
94         LED_ESTADO : out STD_LOGIC_VECTOR(N_ESTADOS - 1 downto 0)
95     );
96 end component;
97
98 -----CONTADOR MONEDAS-----
99
100 component COUNTER is
101     Generic(
102         N_MONEDAS: POSITIVE;
103         N_VEHICULOS: POSITIVE;
104         SIZE_CUENTA: POSITIVE
105     );
106     Port(
107         CLK: in std_logic;
108         RESET: in std_logic;
109         MONEDAS: in std_logic_vector(N_MONEDAS - 1 downto 0);
110         TIPO_VEHICULO: in std_logic_vector(N_VEHICULOS - 1 downto 0);
111         PAGO_OK: out std_logic;
112         SALDO: out integer;
113         CHANGE: out std_logic_vector(SIZE_CUENTA - 1 downto 0)
114     );
115 end component;
116 signal paid : STD_LOGIC;
117 signal cambio: std_logic_vector(SIZE_CUENTA - 1 downto 0);
118 signal dinero_total: integer;
119
120 -----CONTROL DISPLAYS-----
121 component DIGITAL is
122     Generic(
123         N_MONEDAS: POSITIVE;
124         N_VEHICULOS: POSITIVE;
125         DISPLAYS: POSITIVE;
126         SEGMENTOS: POSITIVE
127     );
128     Port(
129         CLK : in std_logic;
130         TIPO_VEHICULO : in std_logic_vector (N_VEHICULOS - 1 downto 0);
131         DINERO : in integer;
132         DISPLAYS_ON : out std_logic_vector(DISPLAYS - 1 downto 0);
133         SEGMENTOS_ON : out std_logic_vector(SEGMENTOS - 1 downto 0)
134     );
135 end component;
136
137
138 begin
139

```

```

140 -----INSTANCIANDO TODOS LOS COMPONENTES-----
141 Inst_sync : syn
142     generic map (
143         N_MONEDAS => N_MONEDAS,
144         N_VEHICULOS => N_VEHICULOS
145     )
146     port map(
147         clk => CLK,
148         async_coins_in => MONEDAS,
149         sync_coins_out => monedasSync,
150         async_tipo_in => TIPO_VEHICULO,
151         sync_tipo_out => tipoSync
152     );
153
154 Inst_edge : edge
155     generic map (
156         N_MONEDAS => N_MONEDAS,
157         N_VEHICULOS => N_VEHICULOS
158     )
159     port map(
160         clk => CLK,
161         sync_coins_in => monedasSync,
162         edge_coins => monedasEdge ,
163         sync_tipo_in => tipoSync,
164         edge_tipo => tipoEdge
165     );
166 Inst_Dig: DIGITAL
167     generic map(
168         N_MONEDAS => N_MONEDAS,
169         N_VEHICULOS => N_VEHICULOS,
170         DISPLAYS => DISPLAYS,
171         SEGMENTOS => SEGMENTOS
172     )
173     port map(
174         CLK =>CLK,
175         TIPO_VEHICULO =>tipoSync,
176         DINERO => dinero_total,
177         DISPLAYS_ON => DISPLAYS_ON,
178         SEGMENTOS_ON =>SEGMENTOS_ON
179     );
180
181 Inst_Cont: COUNTER
182     generic map(
183         N_MONEDAS => N_MONEDAS,
184         N_VEHICULOS => N_VEHICULOS,
185         SIZE_CUENTA => SIZE_CUENTA
186     )
187     port map(
188         CLK => CLK,
189         RESET => RESET,
190         MONEDAS => monedasEdge,           -- Entradas desde el edge detector para monedas
191         TIPO_VEHICULO => tipoSync,         -- Entradas desde el edge detector para tipo de vehiculo
192         PAGO_OK => paid,                   -- Salida: indica si el pago es correcto
193         SALDO => dinero_total,
194         CHANGE => cambio                  -- Salida: cambio devuelto (ya está conectado en TOP)
195     );
196
197

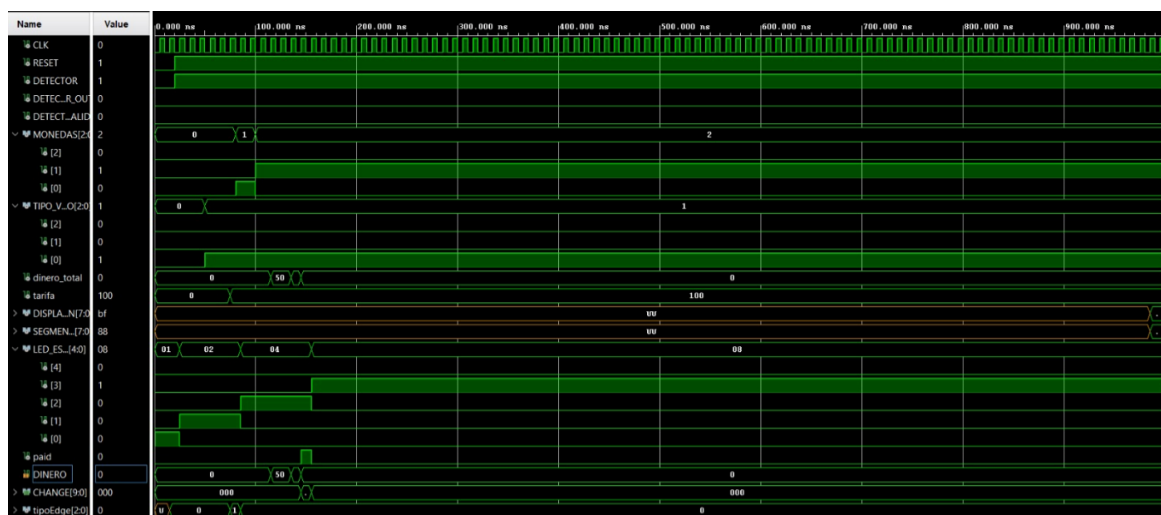
```

```

198
199 Inst_FSM: FSM
200 generic map(
201     N_MONEDAS => N_MONEDAS,
202     N_VEHICULOS => N_VEHICULOS,
203     N_ESTADOS => N_ESTADOS,
204     DISPLAYS => DISPLAYS,
205     SEGMENTOS => SEGMENTOS,
206     SIZE_CUENTA => SIZE_CUENTA
207 )
208 port map(
209     CLK => CLK,
210     RESET => RESET,
211     DETECTOR => DETECTOR,
212     DETECTOR_OUT => DETECTOR_OUT,
213     DETECTOR_SALIDA => DETECTOR_SALIDA,
214     PAGO_OK => paid,
215     BARRERA_UP_LED => BARRERA_UP_LED,
216     BARRERA_DOWN_LED => BARRERA_DOWN_LED,
217     CAMBIO_LED => CAMBIO_LED,
218     TIPO_VEHICULO => tipoEdge,
219     CHANGE => cambio,
220     LED_ESTADO => LED_ESTADO
221 );
222 end Behavioral;

```

TEST BENCH



En el tiempo inicial ($t = 0$ ns), el sistema está en reposo con el RESET activado, asegurando que todas las salidas se inicialicen correctamente. A partir de entonces, el RESET se desactiva, y el sistema comienza a procesar señales; primero se detecta un vehículo con DETECTOR, y se determina su tipo con TIPO_VEHICULO, asignando la tarifa correspondiente (tarifa = 100). Simultáneamente, se introducen monedas mediante MONEDAS, incrementando el saldo en dinero_total. Cuando el saldo alcanza el valor requerido, paid se activa para indicar que el pago es correcto, habilitando acciones como levantar la barrera (controlada por la FSM). Los displays (DISPLAYS_ON, SEGMENTOS_ON) muestran valores como el dinero ingresado y los estados, y los LED_ESTADO evolucionan reflejando las transiciones de la FSM en función de los eventos procesados.

REPOSITORIO GITHUB Y FUNCIONAMIENTO

A continuación, se obtiene el enlace, el cual lleva a el repositorio del proyecto, donde se encuentra todo su código y podrá también observar el funcionamiento del sistema de peaje en video.

https://github.com/juanignacio-martin/TRABAJO_FPGA