ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

SISTEMAS DIGITALES 2

PARALELO 103

PRESENTACIÓN FINAL

TEMA: CIRCUITO CONTROLADOR DE TARJETAS DE LA METROVÍA

MG. SARA RÍOS

CRUZ OCHOA ALBERTO JOSÉ

SÁNCHEZ VALVERDE JOSUÉ STEEVEN

16-01-2019

2T-2018

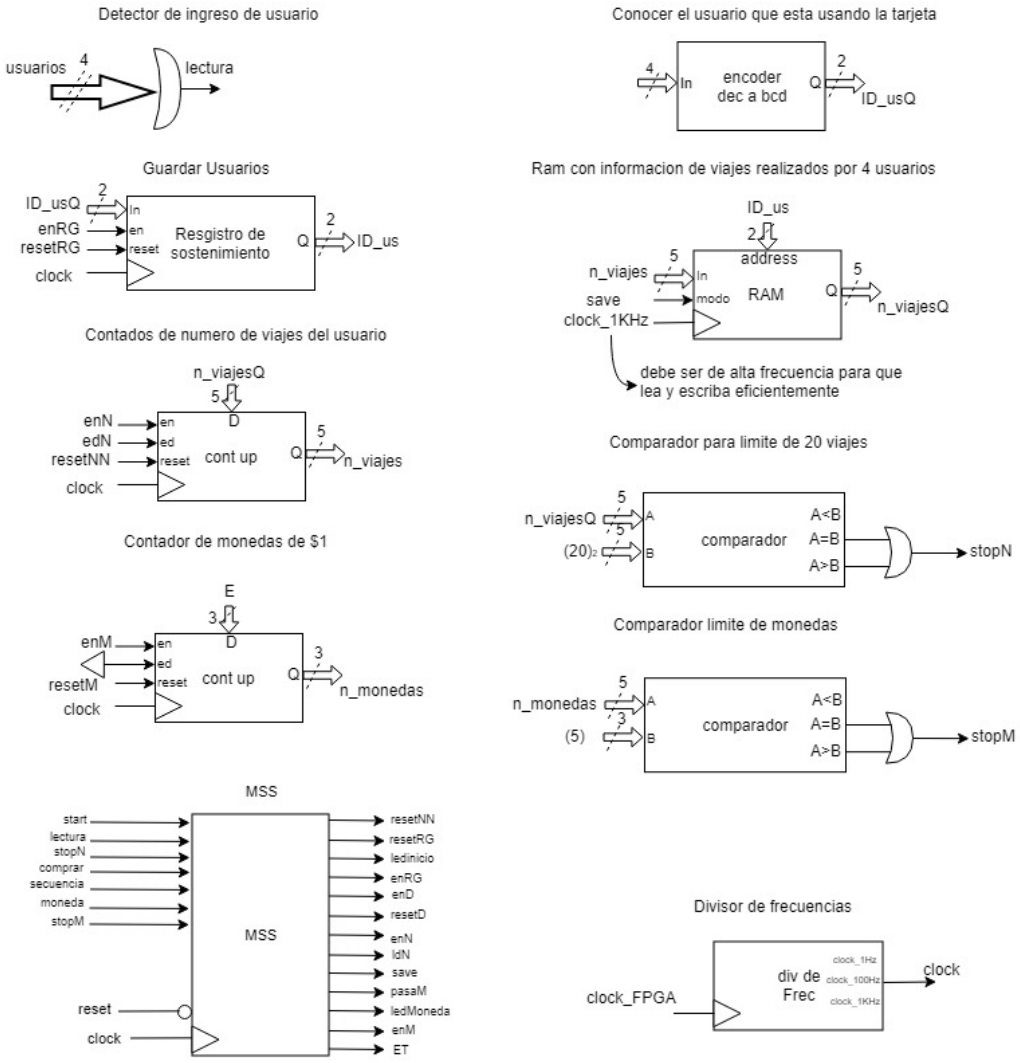
Implementación

1.- Explicación de la solución escogida.

Para la realización de nuestro proyecto se propusieron dos alternativas para la solución del problema de acceso a la metrovia, donde la primera solución fue utilizar una RAM para almacena los valores de viajes de los usuarios y la segunda utilizar registro de sostenimiento para almacenar los viajes de cada usuario.

La solución escogida por el grupo fue la primera propuesta de utilizar una RAM, debido a que si se tiene una gran cantidad de usuarios se utilizaría una RAM de mayor tamaño. Mientras que con la otra propuesta se tendría al final una cantidad enorme de registros de sostenimientos haciéndolo ineficiente.

2.- Diagrama de Bloques del Sistema Digital con su respectiva explicación a detalle.

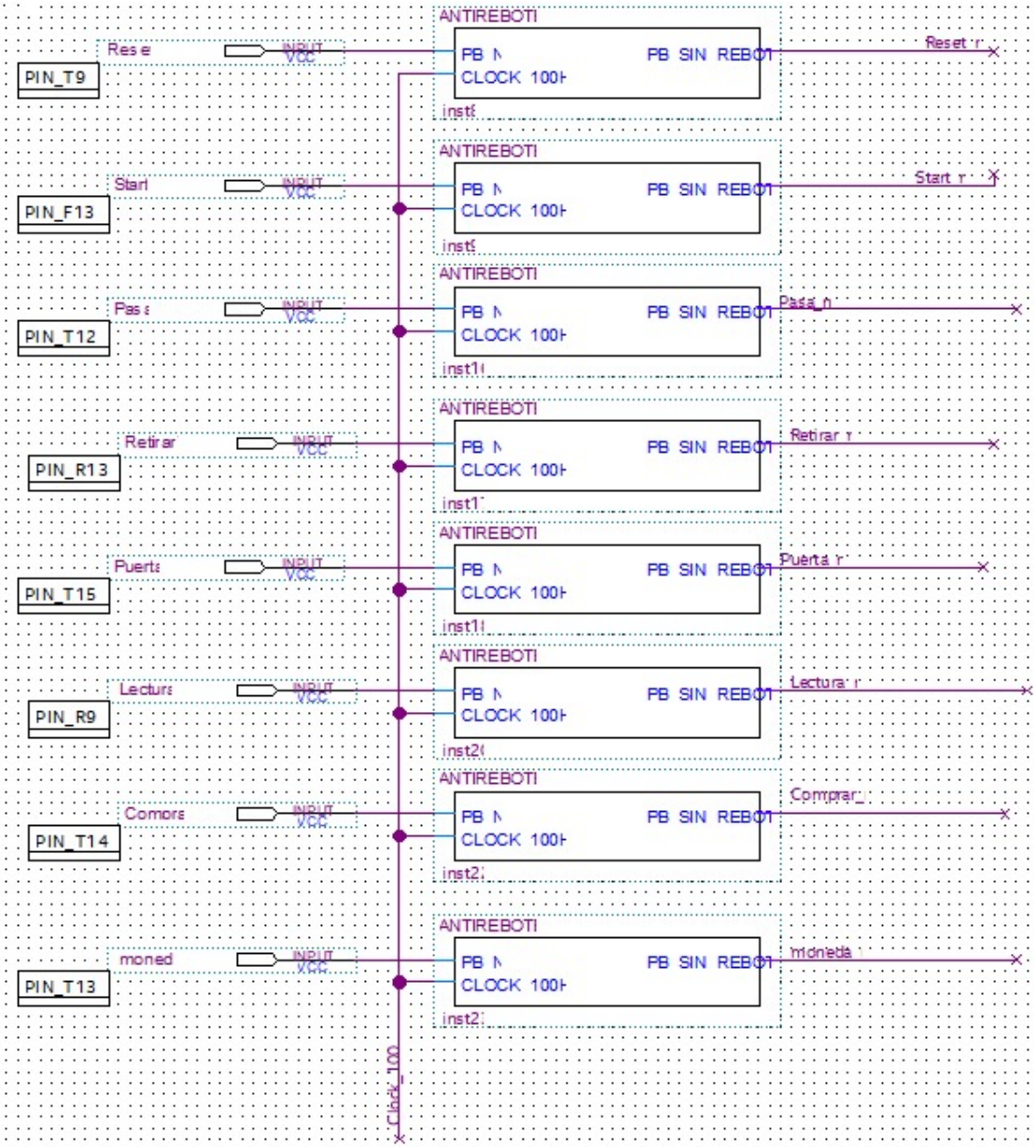


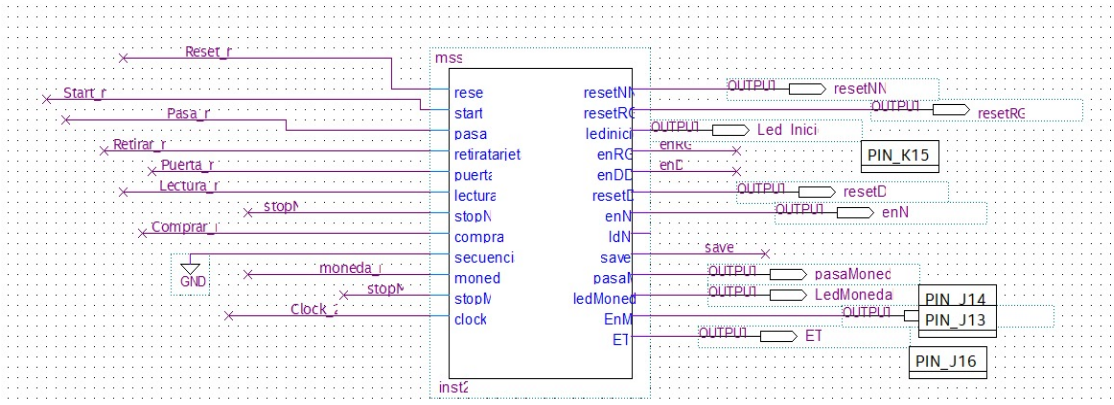
Explicación:

El sistema lee una tarjeta con la dirección del usuario en la memoria RAM para obtener el numero de viajes que posee, en caso de tener cupo para realizar el viaje se activara

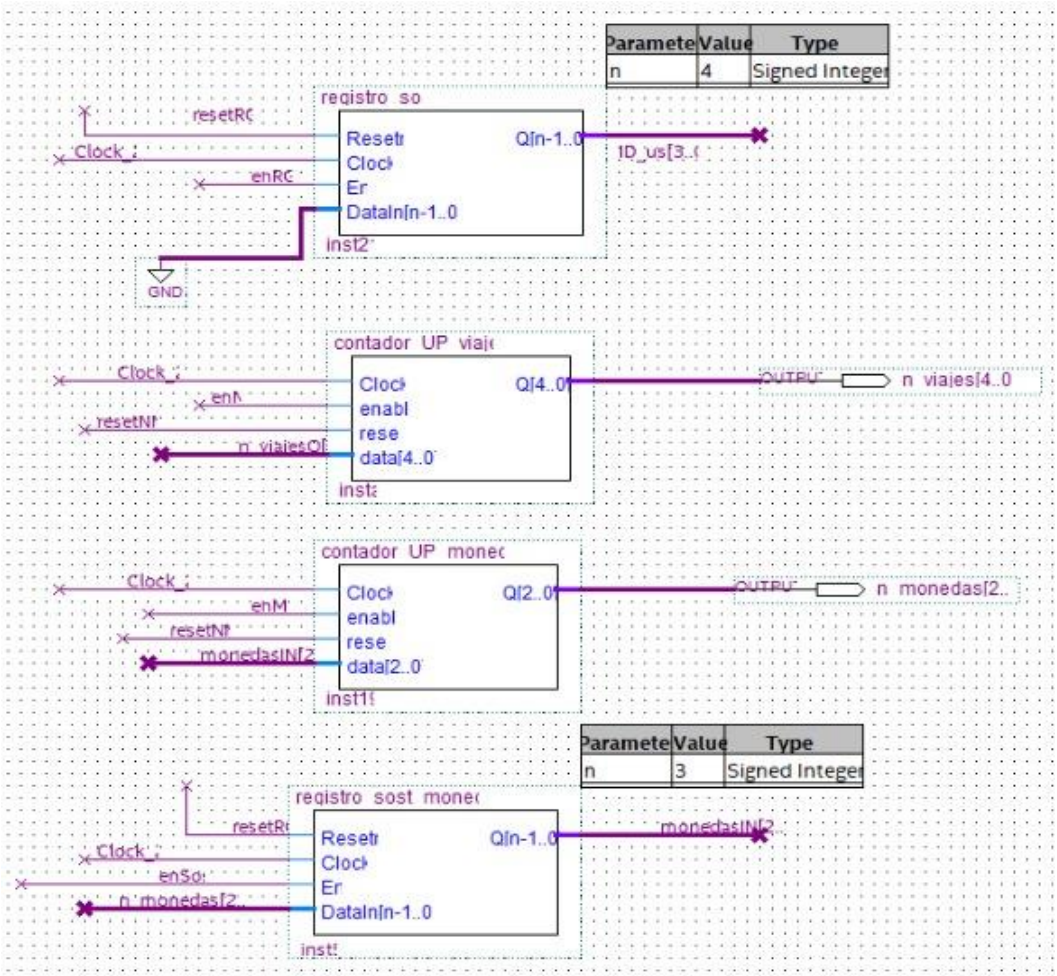
el contador de viajes para incrementar y guardarlo respectivamente en la RAM. Cuando el usuario haya realizado su ultimo viaje y ya no posea cupo, el usuario tendrá que comprar otra tarjeta introduciendo monedas de un dólar hasta completar 5 dólares y que pueda retirar su nueva tarjeta.

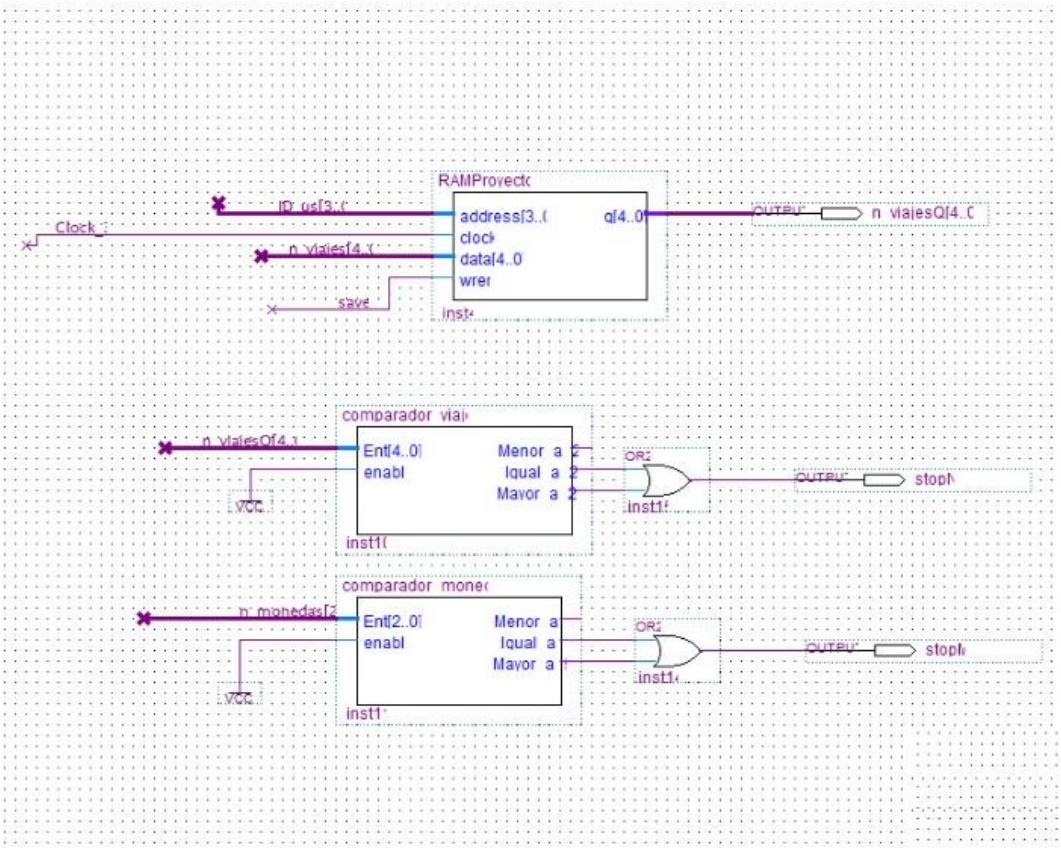
3.- Partición Funcional del Sistema Digital con su respectiva explicación a detalle.

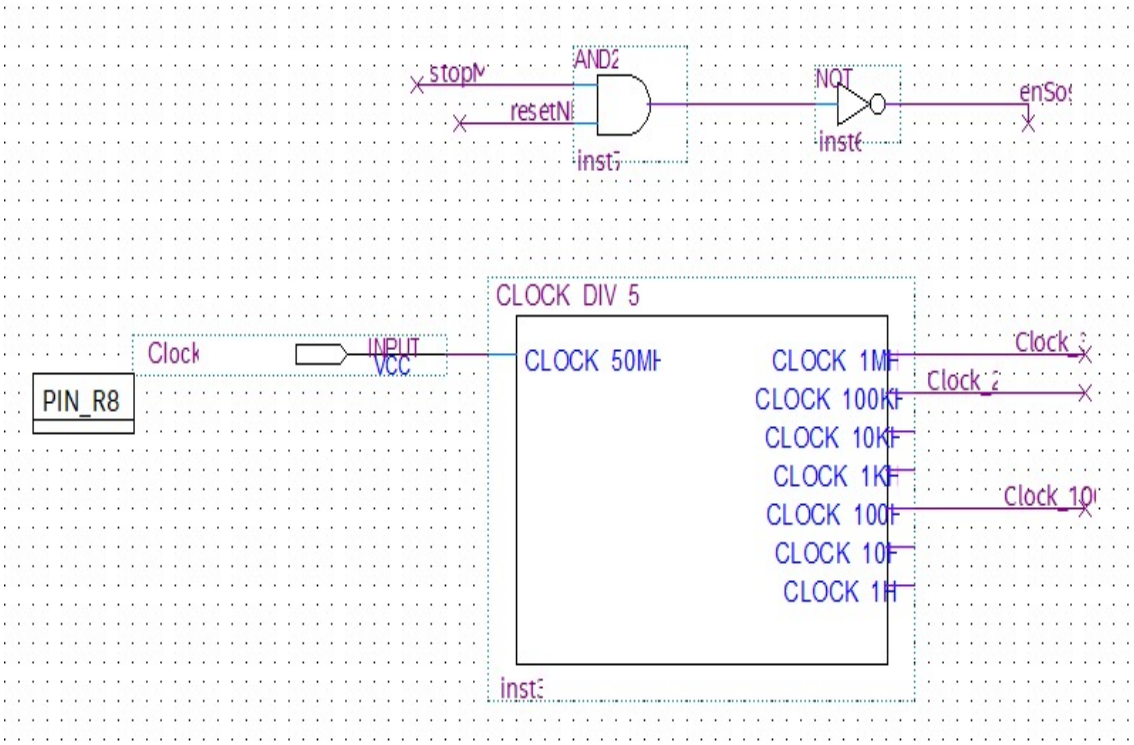




mete	Value	Type
	4	Signed Integer





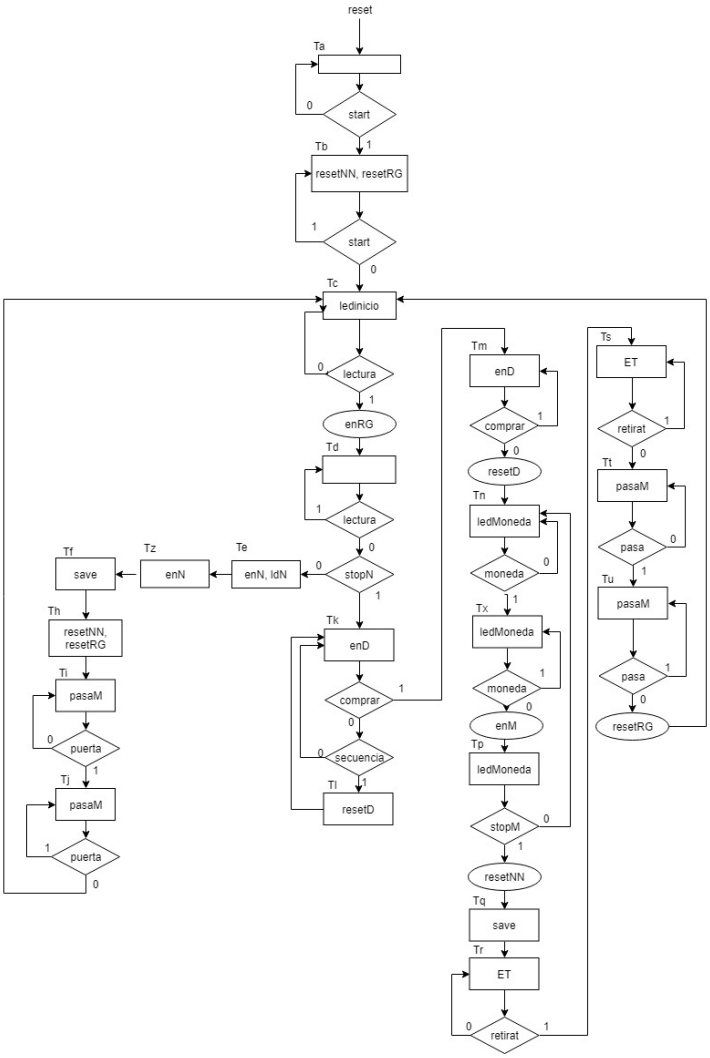


Explicación:

El sistema digital posee bloques de antirebote y un divisor de frecuencia, los cuales nos permitirán que el sistema registre una pulsación del botón y obtener diferentes

velocidades de un reloj respectivamente. El circuito inicia cuando el usuario presiona el botón de start haciendo que la mss encienda el led de inicio, cuando se ingresa una tarjeta se procederá a ingresar el usuario al registro de sostenimiento y a la RAM y el valor almacenado ira al comparador de viajes el cual no dirá si el usuario posee cupo o no. En caso de tener el cupo suficiente el número de viajes aumentara en uno gracias al contador up y ser almacenada en la RAM nuevamente una vez terminado se activará el led pasaM para que indicarle que puede pasar y presionar el botón de puerta y una vez pulsado el sistema espera que ingrese otro usuario. Si no posee cupo tendrá que comprar otra tarjeta, comprándola con el botón comprar, y cuando ya pueda ingresar las monedas se encenderá el led moneda y cuando ingrese una moneda se encenderá un led que a ingresado una moneda, aumentando el contador up de monedas y gracias al comparador de moneda se puede saber cuando el usuario a ingresado la cantidad suficiente de monedas para la nueva tarjeta. Una vez con la cantidad de monedas se activará el led ET para indicarle que puede retirar la tarjeta y pasar a realizar su viaje en la metrovía y el sistema esperara que ingrese otro usuario.

4.- Diagrama ASM del Controlador en Visio o cualquier otra herramienta utilizada para elaborar diagramas de flujo, indicando claramente todas las señales de entrada y salida utilizadas (con su respectiva explicación a detalle).



Explicación:

El sistema inicia en el estado Ta, luego pregunta por el botón de start para iniciar el sistema reiniciando todos los bloques, en el estado Tb, y terminando en el estado Tc una vez el usuario termine de presionar el botón. En este estado se preguntará por una tarjeta de lectura de usuario, pasando del estado Tc al estado Td. Si posee cupo la señal stopN será cero y el sistema pasará por los estados Te, Tz, Tf, Th y Ti; donde se aumentará el cupo de viaje y se actualizará el valor en la RAM. Cuando este proceso acabe el circuito preguntará por el botón de puerta para que el usuario pase a la metrovía, el sistema ira por los estados Ti, Tj y volverá al estado Tc a la espera de otro usuario. En caso de no poseer el cupo suficiente la señal stopN será 1 y pasara al estado Tk donde preguntara por comprar, si el usuario desee comprar pasara al estado Tm y Tn, donde preguntara por las monedas para comprar la tarjeta nueva pasando por los estados Tx, Tp Tq y Tr; al final el sistema espera que el usuario retire la tarjeta y pase a la metrovía, lugar en el cual el sistema pasara por los estados Ts, Tt, Tu y volverá al estado Tc a la espera de otro usuario.

5.- Código VHDL completo de la MSS, usar un process para decodificador de estados siguiente–memoria de estados, y un process para el decodificador de salidas (con su respectiva explicación a detalle).

```
library ieee;
use ieee.std_logic_1164.all;
entity mss is
   port( reset, start, pasar, retiratarjeta, puerta, lectura, stopN, comprar, secuencia, moneda, stopM, clock: in std_logic;
         resetNN, resetRG, ledinicio, enRG, enDD, resetD, enN, ldN, save, pasaM, ledMoneda, EnM, ET: out std_logic);

end mss;

architecture solucion of mss is
type estado is (Ta,Tb,Tc,Td,Te,Tf,Th,Ti,Tj,Tk,Tl,Tm,Tn,Tp,Tq,Tr,Ts,tt,Tu,Tx,Tz);
signal y: estado;
begin
process(clock,reset)
   begin
   if reset = '0' then y<= Ta;
   elsif(clock'event and clock='1') then
      case y is
         when Ta => if start = '0' then y <= Ta; else y <= Tb; end if;
         when Tb => if start = '1' then y <= Tb; else y <= Tc; end if;
         when Tc => if lectura = '0' then y <= Tc; else y <= Td; end if;
         when Td => if lectura = '1' then y <= Td;
                    elsif lectura = '0' and stopN = '0' then y <= Te;
                    else y<= Tk; end if;
         when Te => y <= Tz;
         when Tz => y <= Tf;
         when Tf => y <= Th;
         when Th => y <= Ti;
         when Ti => if puerta = '0' then y <= Ti; else y <= Tj; end if;
         when Tj => if puerta = '1' then y <= Tj; else y <= Tc; end if;
         when Tk => if comprar = '0' and secuencia = '0' then y <= Tk;
                    elsif comprar ='0' and secuencia = '1' then y <= tl;
                    else y <= Tm; end if;
         when Tl => y <= Tk;
         when Tm => if comprar = '1' then y <= Tm; else y <= Tn; end if;
         when Tn => if moneda = '0' and stopM = '0' then y <= Tn; else y <= Tx; end if;
         when Tx => if moneda = '1' then y <= Tx; else y <= Tp; end if;
         when Tp => if stopM = '0' then y <= Tn; else y <= Tq; end if;
         when Tq => y <= Tr;
```

```
         when Tr => if retiratarjeta = '0' then y <= Tr; else y <= Ts; end if;
         when Ts => if retiratarjeta = '1' then y <= Ts; else y <= Tt; end if;
         when Tt => if pasar = '0' then  y <= Tt; else y <= Tu; end if;
         when Tu => if pasar = '1' then y <= Tu; else y <= Tc; end if;
      end case;
   end if;
end process;
process(y)
begin
ledinicio<='0';pasaM<='0';ET<='0';ledMoneda<='0';save <= '0';enM<= '0';enN<= '0';
   case y is
      when Ta =>
      when Tb => resetNN <= '1'; resetRG <= '1';
      when Tc => ledinicio <= '1';if lectura ='1' then enRG <= '1'; else enRG <= '0'; end if;
      when Td =>
      when Te => enN <= '1'; ldN <= '1';
      when Tf => save <= '1';
      when Th =>resetNN <= '0'; resetRG <= '0';
      when Ti => pasaM <= '1';
      when Tj => pasaM <= '1';
      when Tk => enDD <= '1';
      when Tl => resetD <= '1';
      when Tm => enDD <= '1';resetRG<= '1';resetNN<= '1'; if comprar = '0' then resetD <= '1'; else resetD <= '0'; end if;
      when tn => ledMoneda <= '1';
      when Tx => ledMoneda <= '1'; if moneda = '0' then enM <= '1'; else enM <= '0'; end if;
      when Tp => ledMoneda <= '1'; if stopM = '1' then resetNN <= '1'; end if;
      when Tq => save <= '1';resetRG<= '0';resetNN<= '0';
      when Tr => ET <= '1';
      when Ts => ET <= '1';
      when Tt => pasaM <= '1';
      when Tu => pasaM <= '1';
      when Tz => enN <= '1';
   end case;
end process;
end solucion;
```

Explicación:

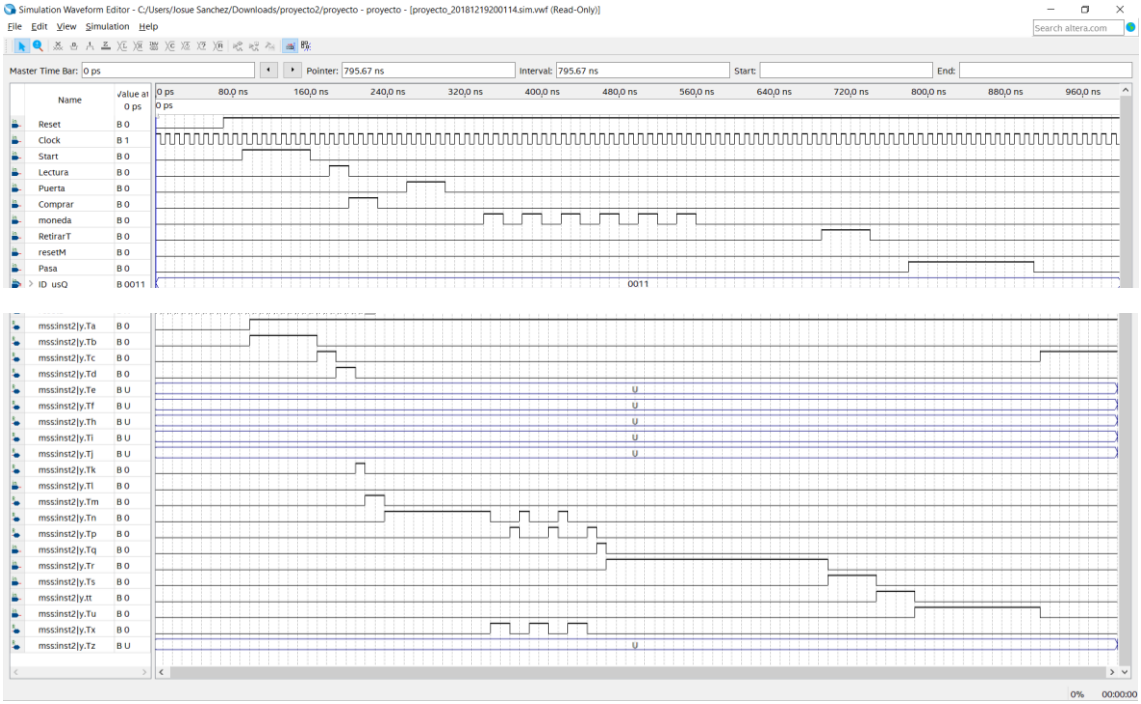
El código mostrado representa al bloque de la MSS de nuestro circuito, el cual permitirá el cambio de estados de la máquina secuencial con sus respectivas salidas siempre y cuando exista un flanco positivo del reloj.

La máquina iniciará cuando el botón de start haya sido pulsado una vez, lo que hará que la maquina pase del estado inicial Ta al estado Tb y posteriormente al estado Tc. Luego se preguntará por lectura, entrada que representa la lectura de una tarjeta, donde se obtendrá información de los viajes realizados por el usuario la maquina cambiará del estado Tc hasta el estado Td. En caso de que el usuario tenga saldo disponible la señal stopN no se activara haciendo que la maquina pase por los siguientes estados Te, Tz, Tf, Th, Ti, Tj y regrese al estado Tc para preguntar por otro usuario. Pero si el usuario no posee saldo se activará la señal stopN y la maquina pasará al estado Tk, el cual le preguntará al usuario si desea comprar otra tarjeta donde se validará que el usuario pulse una sola vez el botón comprar pasando del estado Tk al estado Tm y después al estado Tn. Luego el usuario debe ingresar monedas hasta completar el valor de la tarjeta donde pasara por los estados Tx, Tp, Tq, Tr. Estando en el estado Tr el usuario pulsara una vez el botón retiratarjeta para retirar la tarjeta nueva y poder pasar a la metrovía, donde la maquina pasara del estado Tt, al estado Tu y luego regresara al estado Tc a la espera de otro usuario.

Las salidas de la MSS dependerán del estado en que se encuentre la maquina secuencial, en los estados Ta y Td las señales de salida serán cero. Luego en los estados Tb, Th se activarán las salidas resetNN y resetRG, los cuales están encargados de reiniciar el contador y el registro de sostenimiento respetivamente. Después en el estado Tc se activarán las salidas ledincio, la cual indicara que la maquina está funcionando, y enRG, donde se habilitara el registro de sostenimiento y además esta salida solo se activara si lectura es uno. En el estado Te se encenderán las salidas enN y ldN, quienes habilitarán al contador y se le cargara un valor inicial a dicho contador. En los estados Tf y Tq se habilitará la señal de save, que cambiará de estado de lectura y escritura de la RAM. En los estados Ti y Tj se habilitará el led pasaM indicando al usuario que ya está autorizado para pasar. En el estado Tk se habilitará la señal enDD, la cual permitirá habilitar el funcionamiento del registro de desplazamiento. En el estado Tl se habilita resetD, el cual es el encargado de reiniciar el registro de desplazamiento. En el estado Tm se habilitarán las señales de enD y resetD los cuales están encargados de reiniciar el registro de desplazamiento. En el estado Tn se habilitará la señal ledmoneda, quien es un indicador de que el usuario puede ingresar monedas. En el estado Tx se habilitarán las salidas ledmoneda y enM quienes indicaran que el usuario puede ingresar monedas y habilita el contador up respectivamente, además de que enM solo se activara si moneda es cero. En el estado Tp se habilitará ledmoneda y resetNN, quienes indicaran que el usuario puede ingresar moneda y que se reiniciar el número de viajes de usuario. Además, que resetNN solo se activara si stopM se activa. En los estados Tr y Ts se activará ET, el cual se encenderá un led para entregar la tarjeta comprada. En los estados Tt y Tu se activará la señal pasaM, quien encenderá un led indicando que el usuario puede ingresar. Por último, el estado Tz encenderá la señal de enM, donde se activará el contador up para los viajes de usuarios.

6.- Diagrama de tiempo del controlador generado en el archivo .VWF (Vector Waveform File), en el que demuestre todos los estados de la MSS (con su respectiva explicación a detalle).

- **Usuario sin cupos (Flujo de Estados)**

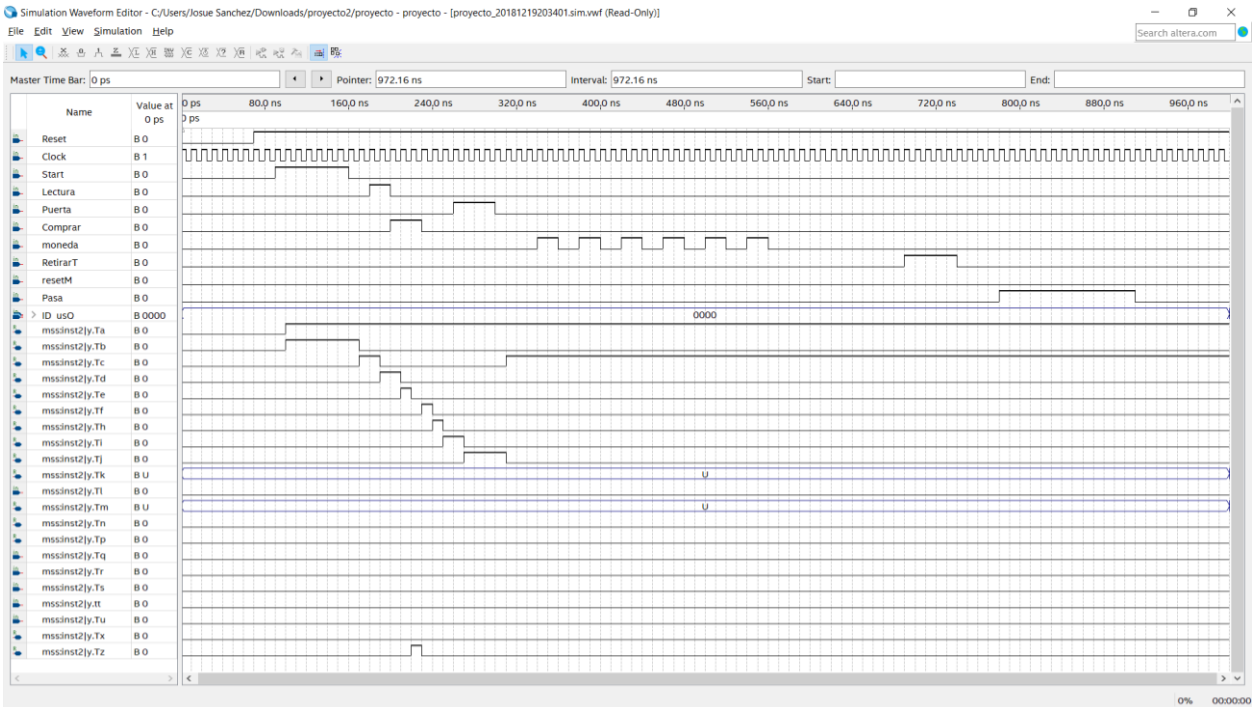


Explicación:

El presente diagrama .vwf nos muestra el comportamiento del sistema digital propuesto teniendo en cuenta que el usuario que intenta acceder al servicio de metrovia ya ha cumplido con los 20 viajes limite que puede realizar por la compra de una tarjeta. Se debe tener en cuenta que la señal de entrada ID_usQ hace referencia a una dirección de memoria dentro de la RAM que se implementó para este sistema. En dicha dirección se almacena mediante 5 bits la cantidad de viajes que supuestamente un usuario puede tener en su tarjeta. En este caso ese valor será 10100 (máximo número de viajes posibles realizables) por lo que el sistema se comportaría de la siguiente manera. El botón Start permitirá inicializar todo el sistema al presionar y soltar una sola vez dicho botón, por lo que se puede ver como del estado Ta (inicial) al presionarlo se dirige al Tb para luego validar que se dejó de presionar el botón en cuestión. La señal lectura hace alusión a un sensor que detecta una tarjeta, y que al igual que la señal Start, se validará con antirrebote desde el estado Tc al estado Td. Luego se puede apreciar como de este último estado se procede al estado Tk, esto se debe a que no cumplió con la validación realizada por el comparador de viajes realizados, es decir la señal interna stopN tomó un valor de 1 lógico que lo llevará por ende al estado Tk y para luego proceder con una compra de tarjeta pasará por los estados Tm y Tn. En el estado Tn corresponde a la recepción de monedas por parte del usuario por lo que se validó que la moneda pase por un sensor. Se puede apreciar que la señal de entrada fue configurada 6 veces, para llegar al punto de haber ingresado las 5 monedas de dólar y llegar al estado Tq donde se recargará los pasajes adquiridos. Tr corresponde a otro sensor que valida que la tarjeta ha sido retirada del compartimiento. La señal Retirar tarjeta en el diagrama muestra como en el estado Tr se espera hasta que la tarjeta haya sido retirada

(antirrebote con el estado Ts). Finalmente, los estados Tt y Tu representan la validación de haber presionado un botón de finalización de la compra y recarga.

- **Usuario con cupos (Flujo de Estados)**



Explicación:

El presente diagrama .vwf nos muestra el comportamiento del sistema digital propuesto teniendo en cuenta que el usuario que intenta acceder al servicio de metrovia posee cupos para realizar los viajes con este servicio de transporte. Se debe tener en cuenta que la señal de entrada ID_usQ hace referencia a una dirección de memoria dentro de la RAM que se implementó para este sistema. En dicha dirección se almacena mediante 5 bits la cantidad de viajes que supuestamente un usuario puede tener en su tarjeta. En este caso ese valor será 00100 (número de viajes realizados) por lo que el sistema se comportaría de la siguiente manera. El botón Start permitirá inicializar todo el sistema al presionar y soltar una sola vez dicho botón, por lo que se puede ver como del estado Ta (inicial) al presionarlo se dirige al Tb para luego validar que se dejó de presionar el botón en cuestión. La señal lectura hace alusión a un sensor que detecta una tarjeta, y que al igual que la señal Start, se validará con antirrebote desde el estado Tc al estado Td. Cabe mencionar que las señales Comprar, Moneda y RetirarT no se toman en cuenta ni afectan el comportamiento del sistema con las consideraciones iniciales. Una vez validada la presencia de una tarjeta, el sistema verifica también que el usuario posea aun cupos para los viajes por lo que en el estado Td la variable stopN toma un valor de 0, indicando que puede acceder. Posteriormente, del estado Td se pasa al estado Te que permitirá habilitar señales internas que inicializará el registro de un nuevo viaje realizado la memoria de la tarjeta, esto una vez llegado al estado Tf. Cumplido el objetivo de agregar un viaje a la tarjeta del usuario el contador del sistema se resetea en el estado Th para ser usado por otro usuario. Se comprueba finalmente que el usuario haya pasado por el sensor mediante los estados de antirrebote Ti y Tj.

7.- Código VHDL completo de todos los bloques a utilizar en el Sistema Digital (con su respectiva explicación a detalle).

- **Registro de Sostenimiento**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY registro_sost IS
            Generic(n:Integer:=4);
            Port(Resetn,Clock,En: in std_logic;
                        DataIn: in std_logic_vector(n-1 downto 0);
                        Q: out std_logic_vector(n-1 downto 0));

END registro_sost;

ARCHITECTURE sol OF registro_sost IS
SIGNAL temp: std_logic_vector(n-1 downto 0);
BEGIN
   Process (Clock,Resetn)
   BEGIN
         if Resetn='0' then temp<=(OTHERS =>'0');
         elsif(Clock'event and Clock='1') then
               if(En='1') then
                     temp<=Datain;
               end if;
         end if;
   end process;
   Q<=temp;
END sol;
```

Explicación:

El registro de sostenimiento es el encargado de mantener un valor a la salida del bloque y este valor cambiara cuando la entrada habilitadora se encuentre en un estado alto y a su vez en la entrada de DataIn sea un valor diferente a la salida. En cuanto al proyecto se utilizaron dos registros de sostenimiento el primero para mantener el id del usuario que está deseando ingresar y el segundo se utilizó para mantener el número de monedas ingresadas para la compra de la nueva tarjeta.

- **Comparador número de viajes guardados en memoria de la tarjeta**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity comparador_viajes is
   Port ( Ent: in STD_LOGIC_VECTOR(4 downto 0);
      enable : in std_logic;
      Menor_a_20 : OUT STD_LOGIC;
      Igual_a_20 : OUT STD_LOGIC;
      Mayor_a_20 : OUT STD_LOGIC
      );
end comparador_viajes;

Architecture sol of comparador_viajes is
Begin
   Menor_a_20<='1' when (Ent<"10100" and enable='1') else '0';
   Igual_a_20<='1' when (Ent="10100" and enable='1') else '0';
   Mayor_a_20<='1' when (Ent>"10100" and enable='1') else '0';
end sol;
```

Explicación:

El bloque comparador se utilizó para comparar el número de viajes que el usuario a realizado para poder activar la señal stopN, la cual nos indicara si el usuario actual posee saldo para poder realizar el viaje. En este caso el número máximo de viajes a realizar por el usuario es 20.

- **Comparador número de monedas ingresadas por el cliente para comprar tarjeta**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity comparador_monedas is
    Port ( Ent: in STD_LOGIC_VECTOR(2 downto 0);
           enable : in std_logic;
           Menor_a_5 : OUT STD_LOGIC;
           Igual_a_5 : OUT STD_LOGIC;
           Mayor_a_5 : OUT STD_LOGIC
           );
end comparador_monedas;

Architecture sol of comparador_monedas is
Begin
    Menor_a_5<='1' when (Ent<"101" and enable='1') else '0';
    Igual_a_5<='1' when (Ent="101" and enable='1') else '0';
    Mayor_a_5<='1' when (Ent>"101" and enable='1') else '0';
end sol;
```

Explicación:

El bloque comparador nos permitirá comparar una entrada de 3 bits, las cuales serán el número de monedas ingresadas por el usuario para realizar el pago respectivo por la nueva tarjeta cuyo costo posee un valor de 5 dólares. Una vez el valor sea mayor o igual a se activará la señal stopM la cual nos indicará que el usuario pago en su totalidad el valor de la tarjeta y podrá ingresar a la metrovía.

- **Contador de viajes para ser agregados a la tarjeta**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY contador_UP_viajes IS
            PORT(Clock,enable,reset: IN std_logic;
                                     data: IN std_logic_vector(4 downto 0);
                                     Q:Buffer Std_logic_vector(4 downto 0));
END contador_UP_viajes;

ARCHITECTURE sol OF contador_UP_viajes IS
BEGIN
   Process(Clock,reset)
   BEGIN
         If reset='0' then Q<="00000";
         elsif (Clock'event and Clock='1') then
            If enable='1' then Q<=data+'1'; end if;
         end if;
   END Process;
END sol;
```

Explicación:

El bloque contador permitirá aumentar un valor desde cero o en caso de que se cargue un valor este empezara a subir desde el valor indicado, en nuestro proyecto se lo utilizo para ir aumentando los viajes del usuario por cada viaje que realiza en la metrovia hasta un valor máximo de 20.

- **Contador de monedas ingresadas para comprar o recargar una tarjeta**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY contador_UP_monedas IS
            PORT(Clock,enable,reset: IN std_logic;
                                    data: IN std_logic_vector(2 downto 0);
                                    Q:Buffer Std_logic_vector(2 downto 0));
END contador_UP_monedas;

ARCHITECTURE sol OF contador_UP_monedas IS
BEGIN
   Process(Clock,reset)
   BEGIN
         If reset='0' then Q<="000";
         elsif (Clock'event and Clock='1') then
            If enable='1' then Q<=data+'1'; end if;
         end if;
   END Process;
END sol;
```

Explicación:

Este bloque contador permite hacer una cuenta de las monedas que el usuario ingresa para adquirir una nueva tarjeta. La señal que mantiene este valor se representa con 3 bits, ya que el máximo número al que puede llegar el contador será 5 dólares. El contador solo se activa si se verifica que una moneda a ingresado al sistema.

- **RAM para almacenar la información por cliente**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY RAMProyecto IS
   PORT
   (
      address     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
      clock     : IN STD_LOGIC  := '1';
      data      : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
      wren      : IN STD_LOGIC ;
      q     : OUT STD_LOGIC_VECTOR (4 DOWNTO 0)
   );
END RAMProyecto;

ARCHITECTURE SYN OF ramproyecto IS

   SIGNAL sub_wire0  : STD_LOGIC_VECTOR (4 DOWNTO 0);


BEGIN
   q    <= sub_wire0(4 DOWNTO 0);

   altsyncram_component : altsyncram
   GENERIC MAP (
      clock_enable_input_a => "BYPASS",
      clock_enable_output_a => "BYPASS",
      init_file => "proyecto.mif",
      intended_device_family => "Cyclone V",
      lpm_hint => "ENABLE_RUNTIME_MOD=NO",
      lpm_type => "altsyncram",
      numwords_a => 16,
      operation_mode => "SINGLE_PORT",
      outdata_aclr_a => "NONE",
      outdata_reg_a => "CLOCK0",
      power_up_uninitialized => "FALSE",
      read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
      widthad_a => 4,
      width_a => 5,
      width_byteena_a => 1
   )
   PORT MAP (
      address_a => address,
      clock0 => clock,
      data_a => data,
      wren_a => wren,
      q_a => sub_wire0
   );


END SYN;
```

Explicación:

El bloque de la RAM permitirá el almacenamiento de los viajes realizados por los diferentes usuarios en donde la posición del address corresponde al ID del usuario registrado para el ingreso de la metrovia, además el valor almacenado en la memoria ir aumentando hasta un máximo de 20 viajes a realizar por el usuario.

- **Antirebote para los botones**

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

   -- Debounce Pushbutton: Filters out mechanical switch bounce for around 40Ms.
ENTITY ANTIREBOTE IS
   PORT(PB_N, CLOCK_100Hz  : IN  STD_LOGIC;
        PB_SIN_REBOTE     : OUT STD_LOGIC);
END ANTIREBOTE;

ARCHITECTURE a OF ANTIREBOTE IS
   SIGNAL SHIFT_PB       : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN

   -- Debounce clock should be approximately 10ms or 100Hz
   PROCESS
   BEGIN
      WAIT UNTIL (clock_100Hz'EVENT) AND (clock_100Hz = '1');
      -- Use a shift register to filter switch contact bounce
      SHIFT_PB(2 DOWNTO 0) <= SHIFT_PB(3 DOWNTO 1);
      SHIFT_PB(3) <= NOT PB_N;
      IF SHIFT_PB(3 DOWNTO 0)="0000" THEN
            PB_SIN_REBOTE <= '0';
      ELSE
            PB_SIN_REBOTE <= '1';
      END IF;
   END PROCESS;
END a;
```

Explicación:

Los bloques antirebote del sistema nos permiten asegurar que la señal que es ingresada por las botoneras proyecto sean leídas por el controlados como si fueran pulsadas una sola vez, de lo contrario el controlador pensara que el botón ha sido pulsado más de una vez afectando negativamente la respuesta del sistema.

- **Divisor de frecuencia**

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY CLOCK_DIV_50 IS
    PORT
    (  CLOCK_50Mhz    :IN   STD_LOGIC;
       CLOCK_1Mhz     :OUT  STD_LOGIC;
       CLOCK_100Khz   :OUT  STD_LOGIC;
       CLOCK_10Khz    :OUT  STD_LOGIC;
       CLOCK_1Khz     :OUT  STD_LOGIC;
       CLOCK_100hz    :OUT  STD_LOGIC;
       CLOCK_10hz     :OUT  STD_LOGIC;
       CLOCK_1hz      :OUT  STD_LOGIC);
END CLOCK_DIV_50;

ARCHITECTURE a OF CLOCK_DIV_50 IS
    SIGNAL   count_1Mhz: STD_LOGIC_VECTOR(5 DOWNTO 0);
    SIGNAL   count_100Khz, count_10Khz, count_1Khz: STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL   count_100hz, count_10hz, count_1hz: STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL   clock_1Mhz_int, clock_100Khz_int, clock_10Khz_int, clock_1Khz_int: STD_LOGIC;
    SIGNAL   clock_100hz_int, clock_10hz_int, clock_1hz_int: STD_LOGIC;

BEGIN
    PROCESS
    BEGIN
-- Divide by 50
       WAIT UNTIL clock_50Mhz'EVENT and clock_50Mhz = '1';  -- 24 Mhz
          IF count_1Mhz < 49 THEN
             count_1Mhz <= count_1Mhz + 1;
          ELSE
             count_1Mhz <= "000000";
          END IF;
          IF count_1Mhz < 4 THEN
             clock_1Mhz_int <= '0';
          ELSE
             clock_1Mhz_int <= '1';
```

```
        END IF;
    -- Ripple clocks are used in this code to save prescalar hardware
    -- Sync all clock prescalar outputs back to master clock signal
        clock_1Mhz <= clock_1Mhz_int;
        clock_100Khz <= clock_100Khz_int;
        clock_10Khz <= clock_10Khz_int;
        clock_1Khz <= clock_1Khz_int;
        clock_100hz <= clock_100hz_int;
        clock_10hz <= clock_10hz_int;
        clock_1hz <= clock_1hz_int;
    END PROCESS;
        -- Divide by 10
    PROCESS
    BEGIN
        WAIT UNTIL clock_1Mhz_int'EVENT and clock_1Mhz_int = '1';
            IF count_100Khz /= 4 THEN
                count_100Khz <= count_100Khz + 1;
            ELSE
                count_100Khz <= "000";
                clock_100Khz_int <= NOT clock_100Khz_int;
            END IF;
    END PROCESS;
        -- Divide by 10
    PROCESS
    BEGIN
        WAIT UNTIL clock_100Khz_int'EVENT and clock_100Khz_int = '1';
            IF count_10Khz /= 4 THEN
                count_10Khz <= count_10Khz + 1;
            ELSE
                count_10Khz <= "000";
                clock_10Khz_int <= NOT clock_10Khz_int;
            END IF;
    END PROCESS;
        -- Divide by 10
```

```
    PROCESS
    BEGIN
        WAIT UNTIL clock_10Khz_int'EVENT and clock_10Khz_int = '1';
            IF count_1Khz /= 4 THEN
                count_1Khz <= count_1Khz + 1;
            ELSE
                count_1Khz <= "000";
                clock_1Khz_int <= NOT clock_1Khz_int;
            END IF;
    END PROCESS;
        -- Divide by 10
    PROCESS
    BEGIN
        WAIT UNTIL clock_1Khz_int'EVENT and clock_1Khz_int = '1';
            IF count_100hz /= 4 THEN
                count_100hz <= count_100hz + 1;
            ELSE
                count_100hz <= "000";
                clock_100hz_int <= NOT clock_100hz_int;
            END IF;
    END PROCESS;
        -- Divide by 10
    PROCESS
    BEGIN
        WAIT UNTIL clock_100hz_int'EVENT and clock_100hz_int = '1';
            IF count_10hz /= 4 THEN
                count_10hz <= count_10hz + 1;
            ELSE
                count_10hz <= "000";
                clock_10hz_int <= NOT clock_10hz_int;
            END IF;
    END PROCESS;
        -- Divide by 10
```

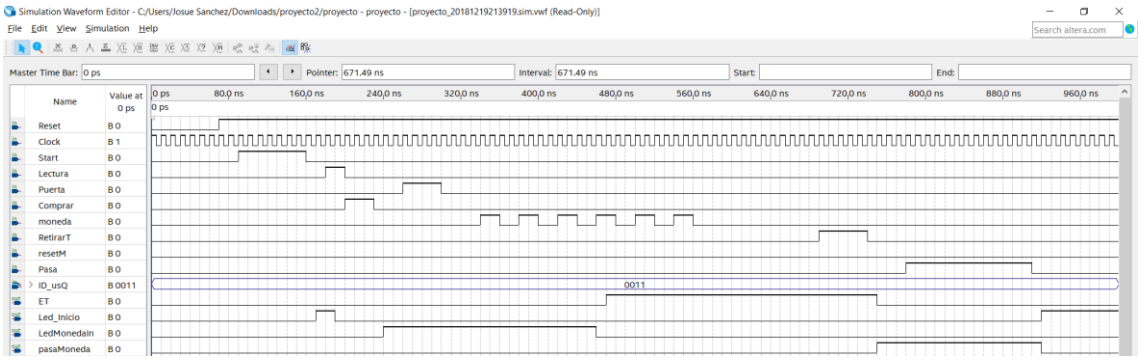
```
    PROCESS
    BEGIN
        WAIT UNTIL clock_10hz_int'EVENT and clock_10hz_int = '1';
            IF count_1hz /= 4 THEN
                count_1hz <= count_1hz + 1;
            ELSE
                count_1hz <= "000";
                clock_1hz_int <= NOT clock_1hz_int;
            END IF;
    END PROCESS;
END a;
```

Explicación:

El bloque de divisor de frecuencia nos permite obtener diferentes velocidades de tiempo a partir de una señal de reloj. Las señales que se pueden obtener son 1Mhz, 100KHz, 10KHz, 1KHz, 100Hz y 1Hz, con el cual en nuestro proyecto se utilizan las frecuencias de el de 1Mhz para la RAM, 100Khz para los bloques que requieran una señal de reloj y la de 100Hz para los bloques antirebote.

8.- Diagrama de tiempo del sistema digital generado en el archivo .VWF (Vector Waveform File), en el que demuestre todos los instantes de su funcionamiento (con su respectiva explicación a detalle).

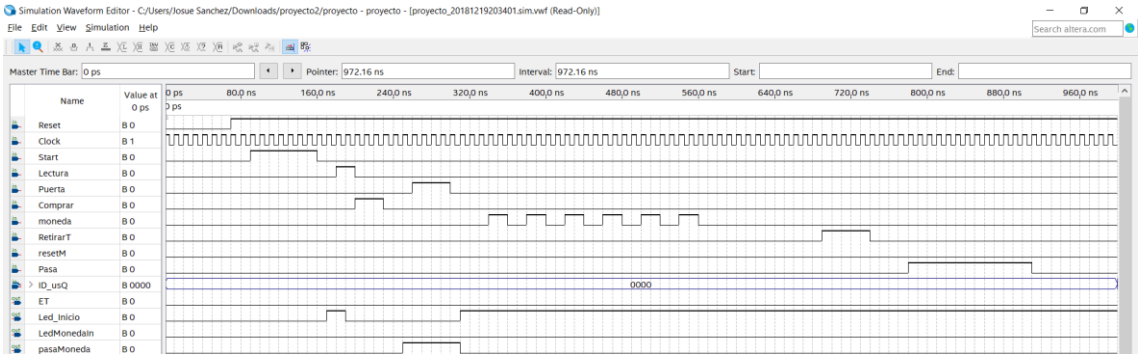
- **Usuario sin cupo de viaje (Salidas)**



Explicación:

El presente diagrama muestra las señales de salida considerando en el sistema a un usuario cuyo número de viajes a llegado a su máximo por lo que tendrá que recargar su tarjeta. Las señales Start y Lectura son validadas por antirrebote. Se debe considerar la señal Comprar misma que habilita el paso a introducir las monedas. Cada moneda que se introduce se representa con la señal moneda que se muestra, la cual verifica el paso de una a la vez. Cuando se inicia el sistema se muestra mediante un led llamado Led_Inicio con un parpadeo. Si el cliente desea comprar o recargar su tarjeta se habilitará un led que permanecerá encendido hasta que haya completado el ingreso de los 5 dólares, valor de la tarjeta o recarga. Una vez retirada la tarjeta el sistema de contará ya un pasaje por lo que la señal de pasarM hace alusión a un led de acceso.

- **Usuario con cupos (Salidas)**



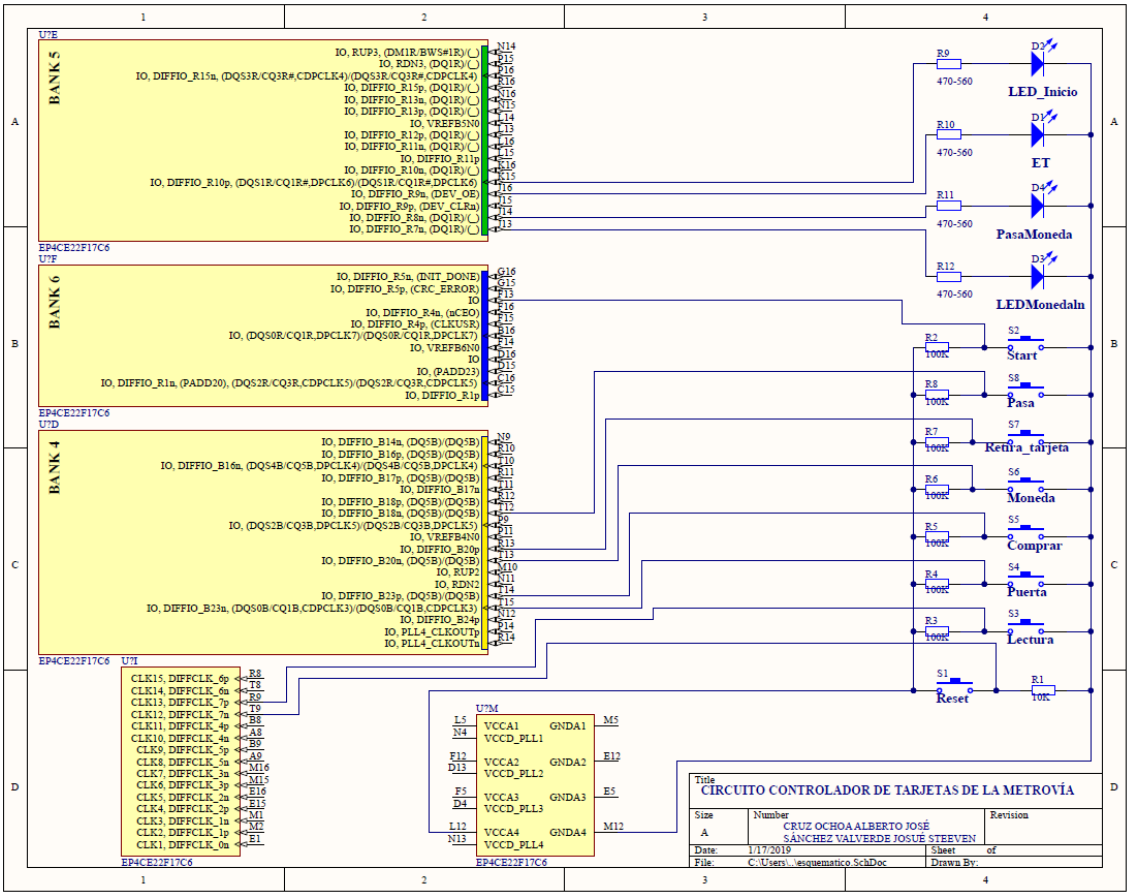
Explicación:

El presente diagrama muestra las señales de salida considerando en el sistema a un usuario con viajes disponibles en su tarjeta. Las señales Start y Lectura son validadas por antirrebote para inicializar y verificar la presencia de una tarjeta respectivamente. No se debe tomar en cuenta las señales comprar, moneda y RetirarT ya que estas son útiles con las condiciones iniciales descritas en la sección anterior.

Debido a que el usuario inicializa el sistema se le mostrará mediante el Led_Inicio una señal de parpadeo. Luego ya que el sistema verifica que posee cupos para realizar el

viaje se le habilitará la señal de pasar mediante un led indicador, esta señal se representa el diagrama adjunto como pasaM.

9.- Diagrama Esquemático del proyecto en físico elaborado con la herramienta ALTIUM Designer (con su respectiva explicación a detalle).



Explicación:

La imagen muestra el diagrama esquemático del proyecto en físico mediante el uso del programa de ALTIUM Designer, en el cual el cuadro verde representa a la palca FPGA, utilizada en el laboratorio durante las prácticas, con el número de los pines del bus de datos que se utilizaran para realizar las conexiones pertinentes. Además, en la parte izquierda de la placa se puede observar los botones que se utilizaran, para el correcto funcionamiento de la máquina. También en la parte derecha se encuentran las salidas de la MSS conectadas a led para indicar cuales se encuentran activas y cuáles no. Por último, la palca FPGA nos proporciona una tierra y una fuente de alimentación para los elementos externos de la placa.

10.- Link del repositorio

https://github.com/albjcruz/Proyecto_digitales_CIRCUITO_CONTROLADOR_DE_TARJETAS_DE_LA_METROV-A.git