UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Arquitectura de Computadoras

PRACTICA #3

"PROGRAMACIÓN Y EMULACIÓN DE UNA MEMORIA DE MICROPROGRAMA."

- EGUIARTE MORETT LUIS ANDRÉS

SEMESTRE: 2018-1

Práctica 3.

Objetivo.

Diseñar, programar y emular una memoria de microprograma empleando lenguajes de descripción de hardware y una tarjeta lógico-programable basada en un FPGA o un CPLD.

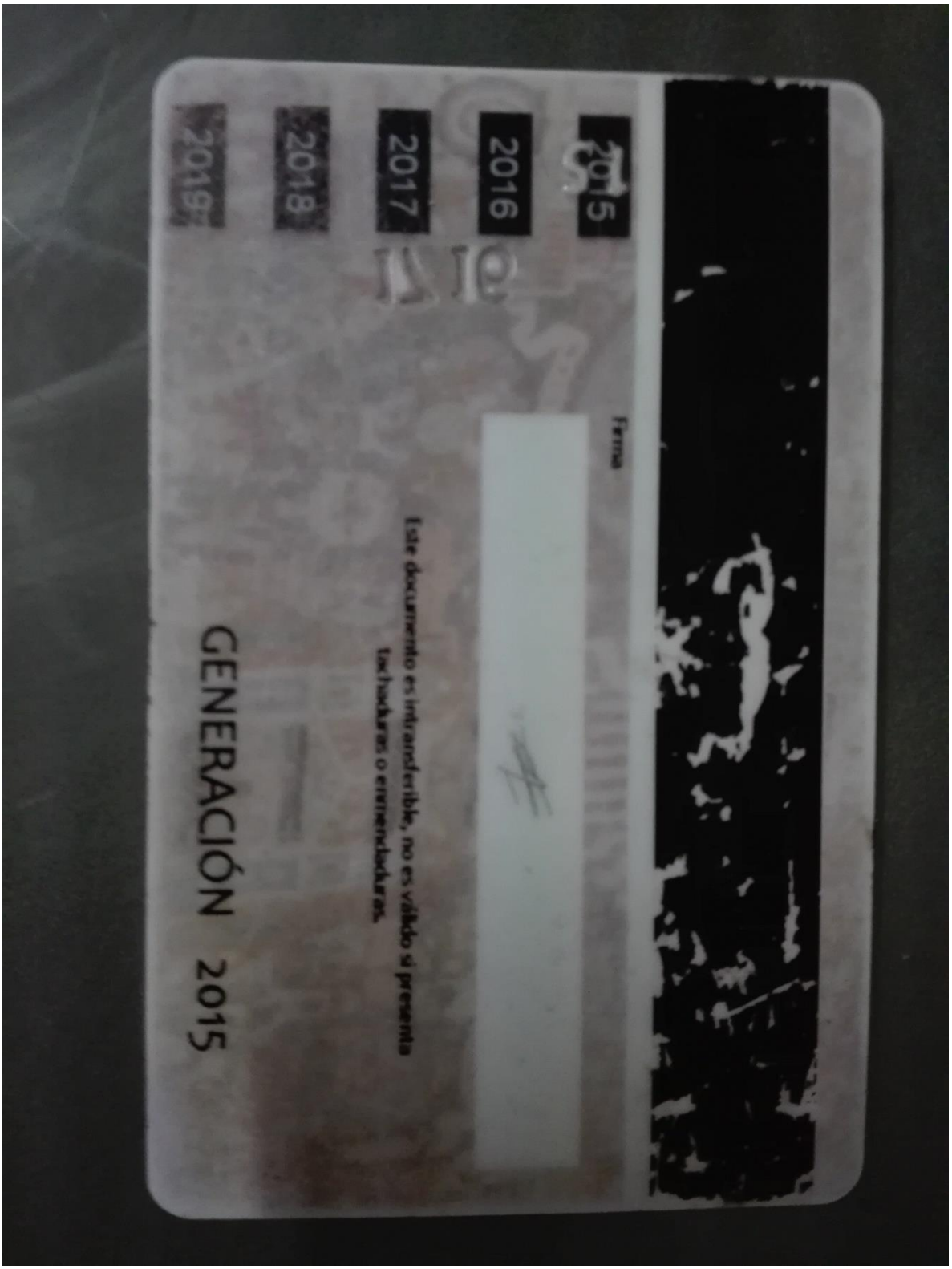
Desarrollo.

Es requisito **INDISPENSABLE** presentar un reporte en formato digital para cada práctica donde aparezca la siguiente información:

1) Copia de una identificación con fotografía

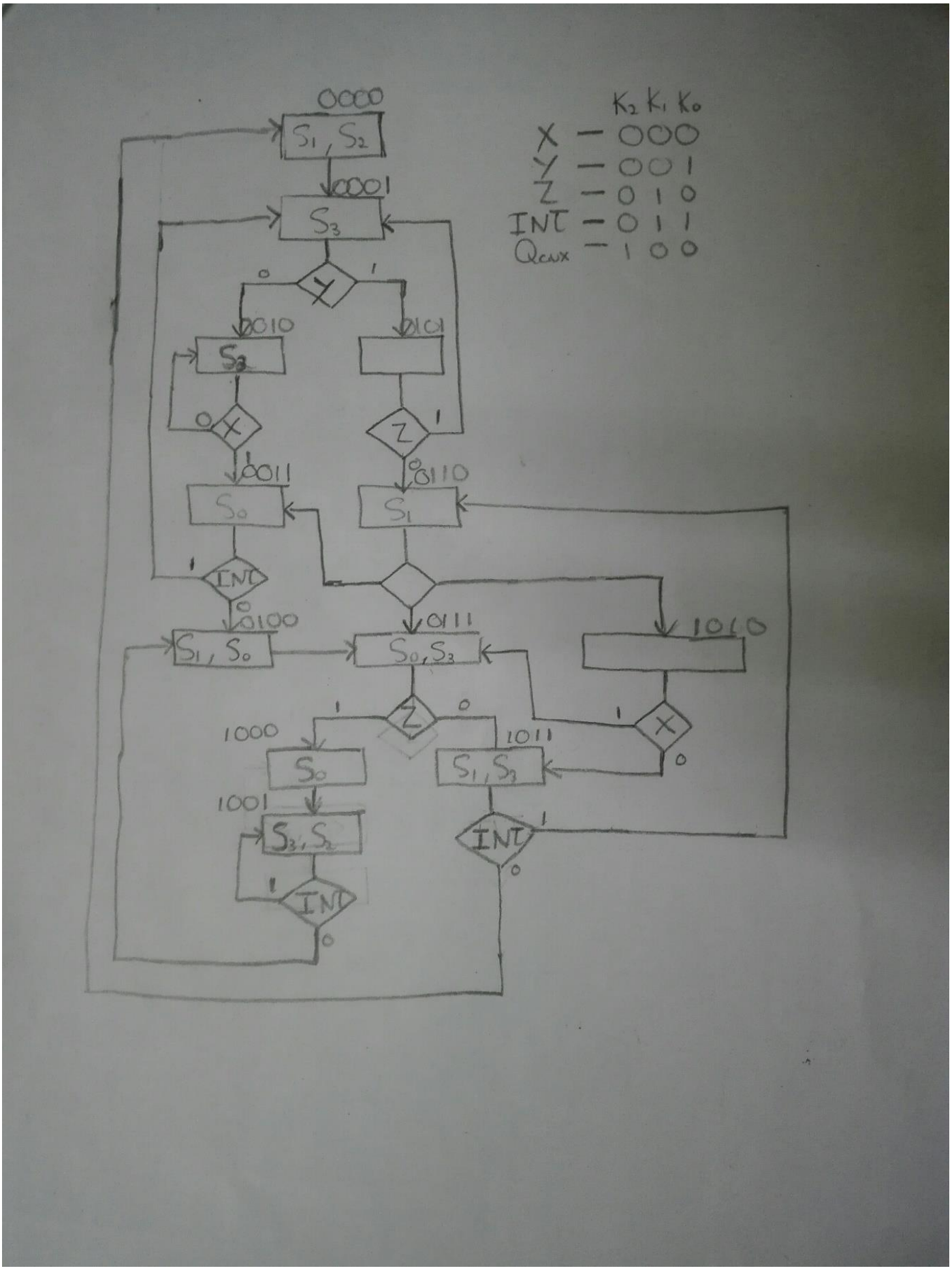
- preferentemente la credencial de estudiante de los miembros del equipo. (incluir correo electrónico de los integrantes).


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Nombre
LUIS ANDRES
EGUIARTE MORETT
No. De Cuenta
31070963 - 7
CURP EUML931106HDFGRS06
Fecha de Emisión
02/08/2014
FACULTAD DE INGENIERIA
ING EN COMPUTACION



Correo: leguiart93@gmail.com

2) Diagrama del diseño de la carta ASM.


0000
S1, S2
0001
S3
Y
0
1
0010
S2
0101
X
0
1
Z
1
0
0011
S0
0110
S1
INT
1
0
0100
S1, S0
0111
S0, S3
1010
Z
1
0
1000
S0
1011
S1, S3
X
1
0
1001
S3, S2
INT
1
0
INT
1
0
K2 K1 K0
X — 000
Y — 001
Z — 010
INT — 011
Qaux — 100

3) Tabla de verdad de la memoria de microprograma.

P3	P2	P1	P0	L3	L2	L1	L0	K2	K1	K0	V.F.	I1	I0	S3	S2	S1	S0
0	0	0	0	*	*	*	*	*	*	*	*	0	0	0	1	1	0
0	0	0	1	0	1	0	1	0	0	1	1	0	1	1	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0
0	0	1	1	*	*	*	*	0	1	1	1	1	1	0	0	0	1
0	1	0	0	0	1	1	1	1	0	0	0	0	1	0	0	1	1
0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	0	0	0
0	1	1	0	*	*	*	*	*	*	*	*	0	0	0	0	1	0
0	1	1	1	1	0	1	1	0	1	0	0	0	1	1	0	0	1
1	0	0	0	*	*	*	*	*	*	*	*	0	0	0	0	0	1
1	0	0	1	*	*	*	*	0	1	1	1	1	1	1	1	0	0
1	0	1	0	0	1	1	1	0	0	0	1	0	1	0	0	0	0
1	0	1	1	*	*	*	*	0	1	1	1	1	1	1	0	1	0
1	1	*	*	0	0	0	0	*	*	*	*	0	1	0	0	0	0

4) El código programado en VHDL, VERILOG, etc.

memoria_top.vhd

```
--Top Entity
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity memoria_top is
    Port ( edo_presente : in  STD_LOGIC_VECTOR(3 downto 0);
           prueba : out std_logic_vector(2 downto 0);
                vf : out std_logic;
                microinstruccion : out std_logic_vector(1 downto
0);
                liga : out std_logic_vector(3 downto 0);
```

```
           salida : out  STD_LOGIC_vector(3 downto 0);
           salida_deco : out STD_LOGIC_VECTOR(6 downto 0));
end memoria_top;

architecture arqmemoria_top of memoria_top is
begin
u3: entity work.deco(arqdeco) port map(edo_presente, salida_deco);
u4: entity work.memoria(arqmemoria) port map(edo_presente,prueba,
vf, microinstruccion, liga, salida);

end arqmemoria_top;
```

memoria.vhd

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

entity memoria is
     Port(estado_presente : in std_logic_vector(3 downto 0);
               prueba : out std_logic_vector(2 downto 0);
               vf : out std_logic;
          microinstruccion : out std_logic_vector(1 downto 0);
               liga : out std_logic_vector(3 downto 0);
               salida : out std_logic_vector(3 downto 0));
end memoria;

architecture arqmemoria of memoria is
```

```
signal int : std_logic_vector(13 downto 0);
begin
      process(estado_presente)
      begin
            case estado_presente is
                        when "0000" => int <= "--------000110";
                        when "0001" => int <= "01010011011000";
                        when "0010" => int <= "00100000010100";
                        when "0011" => int <= "----0111110001";
                        when "0100" => int <= "01111000010011";
                        when "0101" => int <= "00010101010000";
                        when "0110" => int <= "--------100010";
                        when "0111" => int <= "10110100011001";
                        when "1000" => int <= "--------000001";
                        when "1001" => int <= "----0111111100";
                        when "1010" => int <= "01110001010000";
                        when "1011" => int <= "----0111111010";
                        when "1100" => int <= "0000----010000";
                        when "1101" => int <= "0000----010000";
                        when "1110" => int <= "0000----010000";
                        when "1111" => int <= "0000----010000";
                        when others => int <= "00000000000000";
            end case;
            liga <= int(13 downto 10);
            prueba <= int(9 downto 7);
            vf <= int(6);
            microinstruccion <= int(5 downto 4);
            salida <= int(3 downto 0);
```

```
      end process;
end arqmemoria;
```

deco.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use ieee.std_logic_unsigned.all;

entity deco is
port(
      bcd: in STD_LOGIC_VECTOR (3 downto 0);
      display: out STD_LOGIC_VECTOR (6 downto 0));
end deco;

architecture arqdeco of deco is
begin
      with bcd select
      display <= "1000000" when "0000",
                        "1111001" when "0001",
                        "0100100" when "0010",
                        "0110000" when "0011",
                        "0011001" when "0100",
                        "0010010" when "0101",
                        "0000010" when "0110",
                        "1111000" when "0111",
                        "0000000" when "1000",
                        "0011000" when "1001",
                        "0001000" when "1010",
```

```
                    "0000011" when "1011",
                    "1000110" when "1100",
                    "0100001" when "1101",
                    "0000100" when "1110",
                    "0001110" when "1111",
                    "0000110" when others;
end arqdeco;
```

5) Diagrama esquemático del diseño en su caso.

Todo fue programado

6) Tabla de verdad de la lógica que administra los habilitadores de tercer estado.

En esta práctica únicamente se programó la memoria de microprograma y el deco, sin embargo para la práctica cuatro si fue necesaria.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Arquitectura de Computadoras

PRACTICA #4

"PROGRAMACIÓN Y EMULACIÓN DE UN SECUENCIADOR BÁSICO DE 4 MICROINSTRUCCIONES "

- EGUIARTE MORETT LUIS ANDRÉS

SEMESTRE: 2018-1

Práctica 4.

Objetivo.

Diseñar, programar y emular una secuenciador básico de 4 microinstrucciones empleando lenguajes de descripción de hardware y una tarjeta lógico-programable basada en un FPGA o un CPLD.

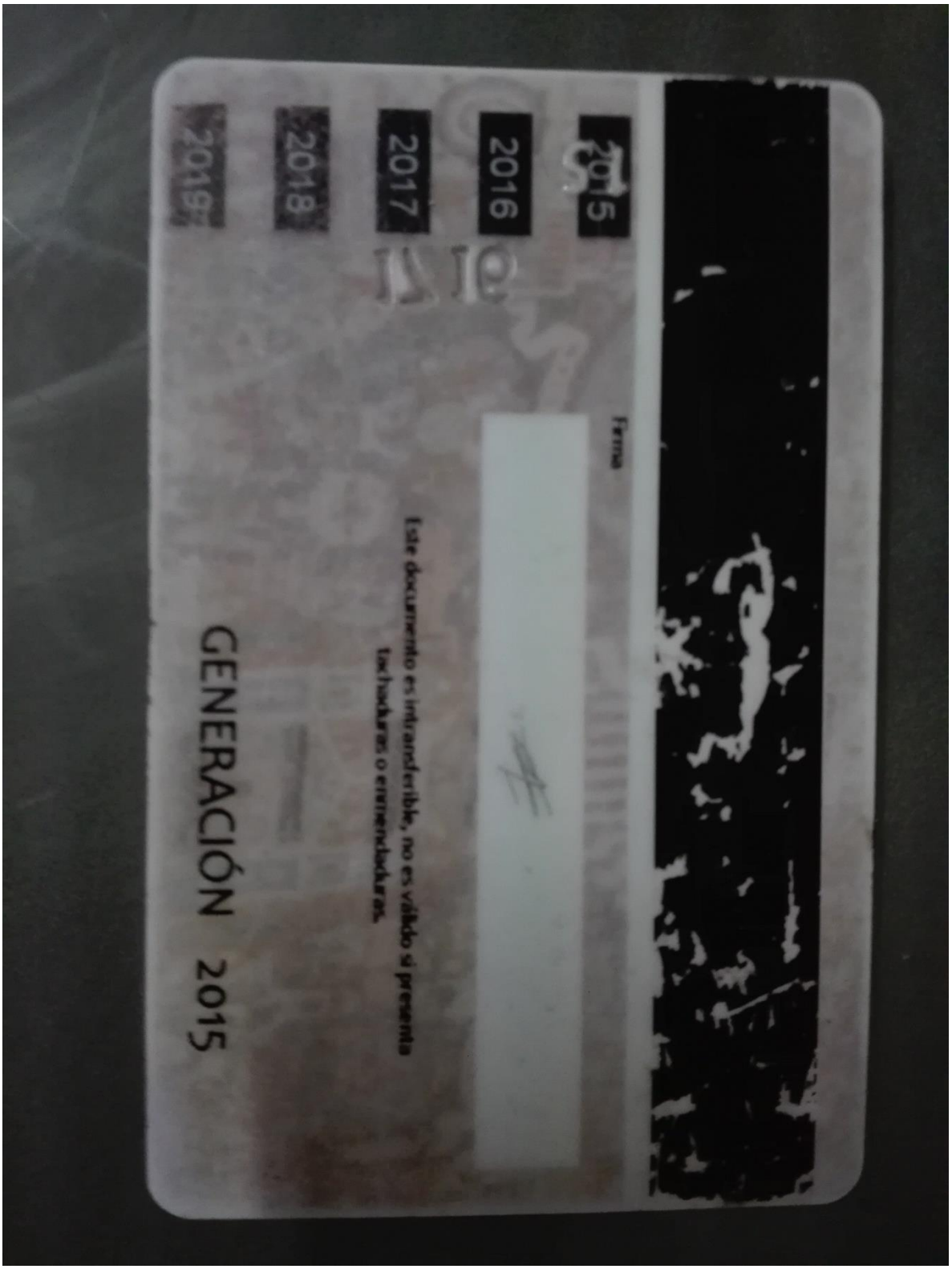
Desarrollo.

Es requisito **INDISPENSABLE** presentar un reporte en formato digital para cada práctica donde aparezca la siguiente información:

1) Copia de una identificación con fotografía

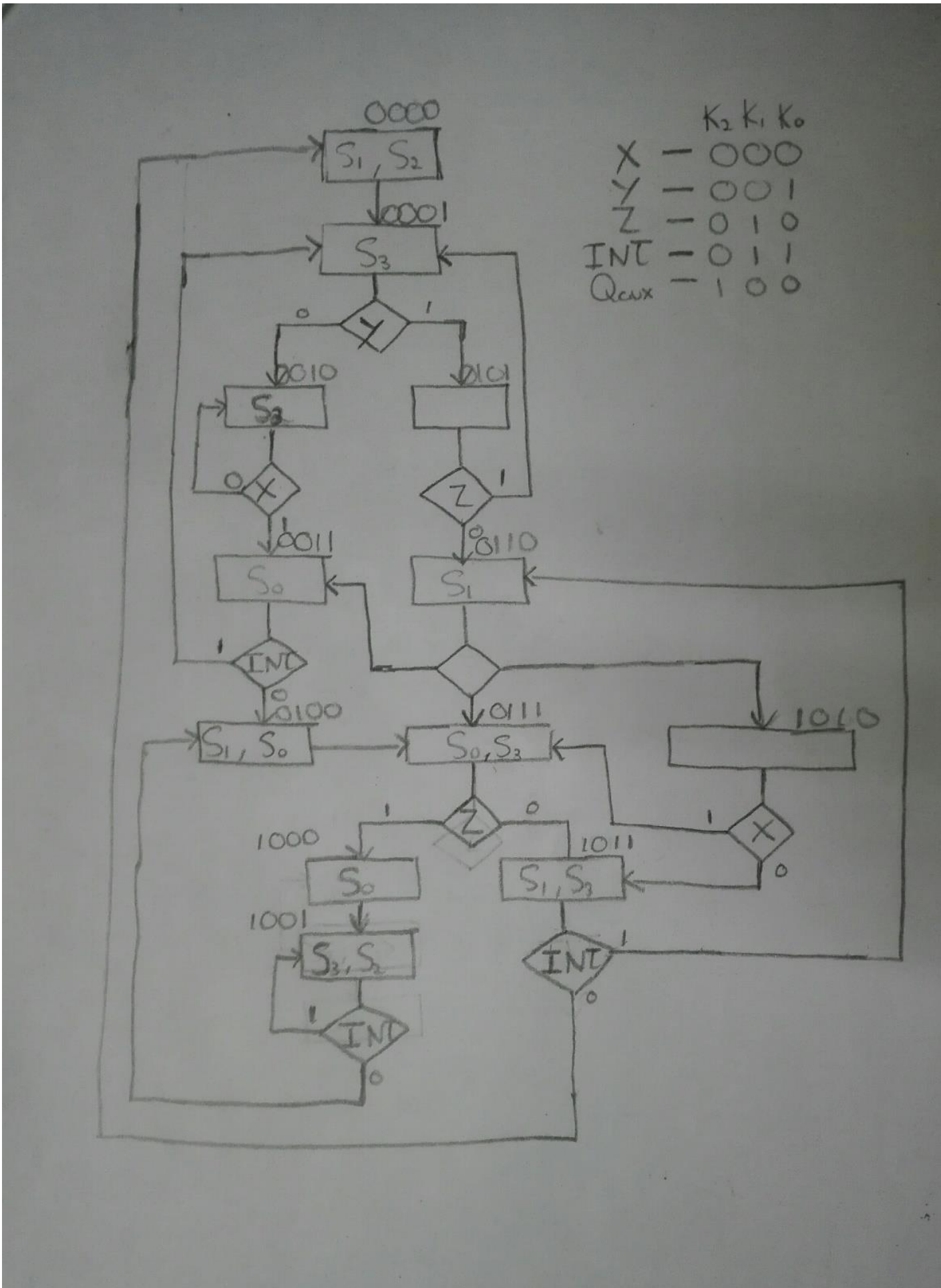
- preferentemente la credencial de estudiante de los miembros del equipo. (incluir correo electrónico de los integrantes).


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Nombre
LUIS ANDRES
EGUIARTE MORETT
No. De Cuenta
31070963 - 7
CURP EUML931106HDFGRS06
Fecha de Emisión
02/08/2014
FACULTAD DE INGENIERIA
ING EN COMPUTACION



Correo: leguiart93@gmail.com

2) Diagrama del diseño de la carta ASM.


K2 K1 K0
X — 000
Y — 001
Z — 010
INT — 011
Qaux — 100
0000
S1, S2
0001
S3
Y
0
1
0010
S2
0101
X
Z
0011
S0
0110
S1
INT
0100
S1, S0
0111
S0, S3
1010
Z
X
1000
S0
1011
S1, S3
1001
S3, S2
INT
INT

3) Tabla de verdad de la memoria de microprograma.

P3	P2	P1	P0	L3	L2	L1	L0	K2	K1	K0	V.F.	I1	I0	S3	S2	S1	S0
0	0	0	0	*	*	*	*	*	*	*	*	0	0	0	1	1	0
0	0	0	1	0	1	0	1	0	0	1	1	0	1	1	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0
0	0	1	1	*	*	*	*	0	1	1	1	1	1	0	0	0	1
0	1	0	0	0	1	1	1	1	0	0	0	0	1	0	0	1	1
0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	0	0	0
0	1	1	0	*	*	*	*	*	*	*	*	0	0	0	0	1	0
0	1	1	1	1	0	1	1	0	1	0	0	0	1	1	0	0	1
1	0	0	0	*	*	*	*	*	*	*	*	0	0	0	0	0	1
1	0	0	1	*	*	*	*	0	1	1	1	1	1	1	1	0	0
1	0	1	0	0	1	1	1	0	0	0	1	0	1	0	0	0	0
1	0	1	1	*	*	*	*	0	1	1	1	1	1	1	0	1	0
1	1	*	*	0	0	0	0	*	*	*	*	0	1	0	0	0	0

4) El código programado en VHDL, VERILOG, etc.

Los módulos de memoria.vhd y deco.vhd quedaron iguales.

relojlento.vhd

```
-- Reloj con divisor de frecuencia
library ieee;
use ieee.std_logic_arith.all;
use ieee.std_logic_1164.all;
entity relojlento is
port(
clkl: in std_logic;
led: buffer std_logic );
end relojlento;

architecture arqrelojlento of relojlento is
signal conteo: integer range 0 to 25000000;
begin
process(clkl)
 begin
  if(clkl' event and clkl='1') then
     conteo<=conteo+1;
       if(conteo=25000000) then
          conteo<=0;
          led<=not(led);
       end if;
   end if;
 end process;
```

```
end arqrelojlento;
```

regi.v

```
module regi(entrada, hab, edo_sig);
	input[3:0] entrada;
	input hab;
	output[3:0] edo_sig;
	reg[3:0] rsalida;

	always @(entrada,hab)
	begin
		case(hab)
			1'b0 : rsalida = entrada;
			1'b1 : rsalida = 4'bzzzz;
		endcase
	end

	assign edo_sig = rsalida;
endmodule
```

secuenciador.v

```
module secuenciador(reloj, reset, cc, microinstruccion, liga, vect, vmap,
pl, map_hab, vect_hab, estado_presente);
	input reloj;
	input reset;
	input cc;
	input[1:0] microinstruccion;
	input[3:0] liga;
	input[3:0] vect;
	input[3:0] vmap;
	output pl;
	output map_hab;
	output vect_hab;
	output[3:0] estado_presente;
	reg[3:0] temporal, temporal2;
	reg[0:0] pltemp, maptemp, vecttemp;

	always@(posedge reloj)
	begin
		if(reset)
			temporal = 0;
		else if(!reset)
		begin
			temporal2 = estado_presente+1;
			case(microinstruccion)
				2'b00: //Paso continuo
				begin
					temporal=temporal2;
```

```
                        pltemp = 1;
                        maptemp = 1;
                        vecttemp = 1;
                  end
                  2'b01:
                        if(!cc)
                        begin
                              temporal=liga; //Salto condicional
                              pltemp = 1'b0;
                              maptemp = 1'b1;
                              vecttemp = 1'b1;
                        end
                        else
                        begin
                              temporal=temporal2; //Paso continuo
                              pltemp = 1'b1;
                              maptemp = 1'b1;
                              vecttemp = 1'b1;
                        end
                  2'b10:
                  begin
                        temporal=vmap; //Salto de transformacion
                        pltemp = 1'b1;
                        maptemp = 1'b0;
                        vecttemp = 1'b1;
                  end
                  2'b11:
                        if(!cc) //Salto por interrupcion
                        begin
                              temporal=vect;
                              pltemp = 1'b1;
                              maptemp = 1'b1;
                              vecttemp = 1'b0;
                        end
                        else
                        begin
                              temporal=temporal2; //Paso continuo
                              pltemp = 1'b1;
                              maptemp = 1'b1;
                              vecttemp = 1'b1;
                        end
            endcase;
      end
      else
            temporal=estado_presente;
end

assign estado_presente=temporal;
assign pl = pltemp;
assign map_hab = maptemp;
```

```
      assign vect_hab = vecttemp;
endmodule
```

lógica.v

```
module logica(vf, qseleccionada, cc);
      input vf;
      input qseleccionada;
      output cc;
      reg temporal;

      always@(qseleccionada)
      begin
            temporal = qseleccionada^vf;
      end

      assign cc = temporal;

endmodule
```

multiplexor.v

```
module multiplexor(x,y,z,interrupcion,prueba, qseleccionada);
      input x;
      input y;
      input z;
      input interrupcion;
      input[2:0] prueba;
      output qseleccionada;
      reg temporal;

      always@(qseleccionada)
      begin
            case(prueba)
                  3'b000:
                        temporal = x;
                  3'b001:
                        temporal = y;
                  3'b010:
                        temporal = z;
                  3'b011:
                        temporal = interrupcion;
                  3'b100:
                        temporal = 0; //Qauxiliar
            endcase
      end

      assign qseleccionada=temporal;
endmodule
```

secuenciador_top.vhd

```
--Top Entity
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity secuenciador_top is
    Port ( clkl, x, y, z, interrupcion, reset : in STD_LOGIC;
           reg_int, reg_trans : in STD_LOGIC_VECTOR(3 downto 0);
           salida : out  STD_LOGIC_VECTOR(3 downto 0);
           salida_deco : out STD_LOGIC_VECTOR(6 downto 0));
end secuenciador_top;

architecture arqsecuenciador_top of secuenciador_top is
component regi
      port(entrada : in STD_LOGIC_VECTOR(3 downto 0);
            hab : in STD_LOGIC;
            edo_sig : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component secuenciador
      port(reloj, reset, cc : in STD_LOGIC;
            microinstruccion : in STD_LOGIC_VECTOR(1 downto 0);
            liga, vect, vmap : in STD_LOGIC_VECTOR(3 downto 0);
          pl, map_hab, vect_hab : out STD_LOGIC;
            estado_presente : out STD_LOGIC_VECTOR(3 downto 0));
end component;

component logica
      port(vf, qseleccionada : in STD_LOGIC;
             cc : out STD_LOGIC);
end component;

component multiplexor
      port(x, y, z, interrupcion : in STD_LOGIC;
            prueba : in STD_LOGIC_VECTOR(2 downto 0);
            qseleccionada : out STD_LOGIC);
end component;

signal cpresente, sentrada, svect, smap, spl, sliga: STD_LOGIC_VECTOR(3
downto 0); --señales de 4 bits
signal sprueba : STD_LOGIC_VECTOR(2 downto 0);--señales de 3 bits
signal smicroinstruccion : STD_LOGIC_VECTOR(1 downto 0);--señales de 2
bits
```

```
signal cdivisor, svect_hab, smap_hab, spl_hab, svf, sqseleccionada, scc
: STD_LOGIC;--señales de 1 bit
begin
u1: entity work.relojlento(arqrelojlento) port map(clkl, cdivisor);
u2: entity work.memoria(arqmemoria) port map(cpresente,prueba=>sprueba,
vf=>svf, microinstruccion=>smicroinstruccion, liga=>sliga,
salida=>salida);
u3: entity work.deco(arqdeco) port map(cpresente, salida_deco);
u4: regi port map(reg_int, svect_hab, edo_sig=>svect);
u5: regi port map(reg_trans, smap_hab, edo_sig=>smap);
u6: regi port map(sliga, spl_hab, edo_sig=>spl);
u7: multiplexor port map(x, y, z, interrupcion, sprueba, sqseleccionada);
u8: logica port map(svf, sqseleccionada, cc=>scc);
u9: secuenciador port map(cdivisor, reset, scc, smicroinstruccion, spl,
svect, smap, spl_hab, smap_hab, svect_hab,cpresente);


end arqsecuenciador_top;
```

5) Diagrama esquemático del diseño en su caso.

Todo fue programado

6) Tabla de verdad de la lógica que administra los habilitadores de tercer estado.

I1	I0	notCC	Sel	notPL	notMAP	notVECT
0	0	*	0	1	1	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	*	1	1	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1