PASSWORD SYSTEM

Trabajo de prácticas de Tecnología de computadores.
Realizado de manera individual.
Entregado 05/01/2020.

Rocío Ruiz Ruiz

Universidad Antonio de Nebrija

1. Introducción

El sistema está formado por una maquina de estados y un sistema de memoria que simulan el funcionamiento de un comprobador de contraseña. La maquina parte de un estado inicial en el que se realiza una copia de los datos de la memoria ROM, en este caso la contraseña inicial, a la memoria RAM. A partir de este momento, el usuario debe introducir una contraseña, y tiene dos opciones. Si pulsa el botón de CHECK, se comprobará si la contraseña introducida es correcta, y habrá hasta 3 intentos. Si acierta la contraseña, se encenderá un led, y luego otro que indica que estas en un estado de bloqueo hasta que pulses el botón de RESET. Esto ultimo también ocurrirá si gastas todos los intentos. La otra opción es que el usuario quiera cambiar la contraseña, en este caso deberá pulsar el botón de WRITE y la contraseña será cambiada en la memoria RAM.

2. Descripción del trabajo realizado detallando los pasos seguidos

En primer lugar, elaboré un diseño con los componentes y señales que iba a necesitar.
Definí un sistema de copia de memoria ROM a memoria RAM. De esta manera, el usuario solo tendría acceso a la memoria RAM, y la ROM quedaría protegida, y como método de seguridad ante posibles errores.

Por ello, los datos de entrada a la RAM deben ser los previamente seleccionados por un componente selector. Este componente elige si los datos que recibe la RAM son los datos de la ROM(solo ocurre en el estado inicial) o los datos que mete el usuario.
A continuación hay un comparador que comprueba si la constraseña(que hay en la RAM) es correcta. Si es correcta el led de salida se encenderá.

Todo lo mencionado anteriormente está almacenado en un componente llamado MemSys. El cual está conectado con otro componente llamado FSM.
FSM es una maquina de estados que se comporta de la siguiente manera:

Estado S0: Este estado es un estado de inicialización del sistema. Se dedica exclusivamente a realizar la copia de ROM a RAM, y pasar al estado S1.

Estado S1: Consta de dos posibilidades

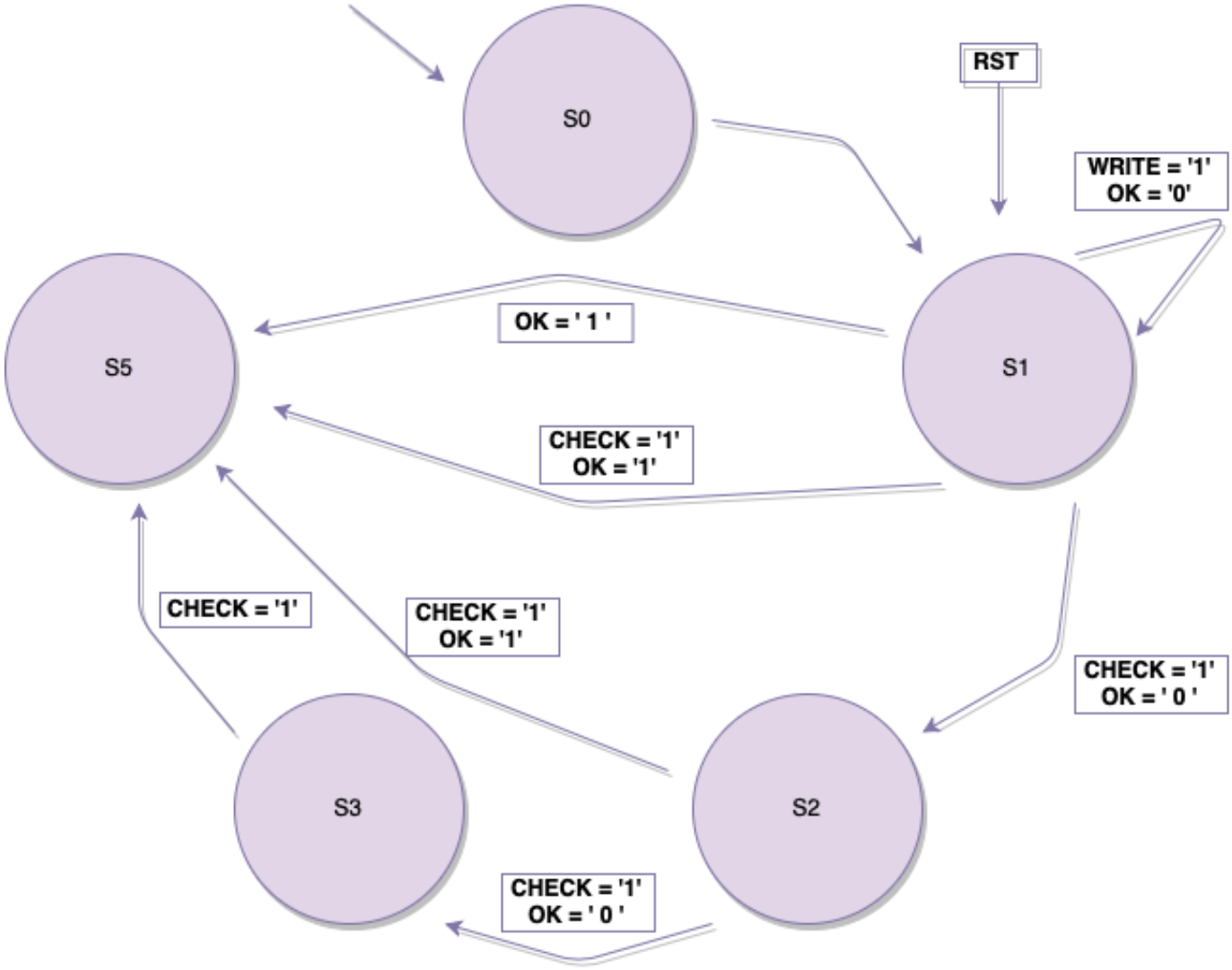
- CHECK = ‘1’: Comprobación de contraseña, de manera que si esta bien, el estado siguiente será S5, y sino, el estado siguiente será S2.
- WRITE = ‘1’: Se cambiará la contraseña con los datos introducidos.

Estados S2, S3, S4: Estos estados solo constan de la posibilidad de CHECK. No se acepta cambio de contraseña.

Estado S5: Cuando se alcanza este estado, se enciende otro led (señal “bomb”) y permanece así hasta que se pulsa RESET.

Para definir el comportamiento de la máquina de estados me ayudé un diagrama de estados.

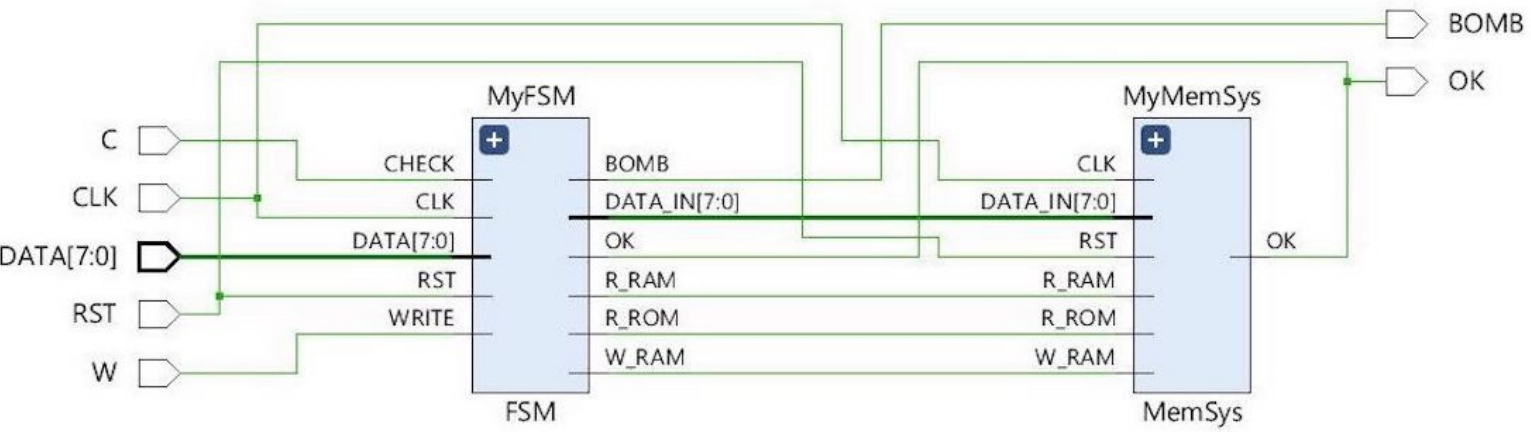
2.1 Diagrama de estados



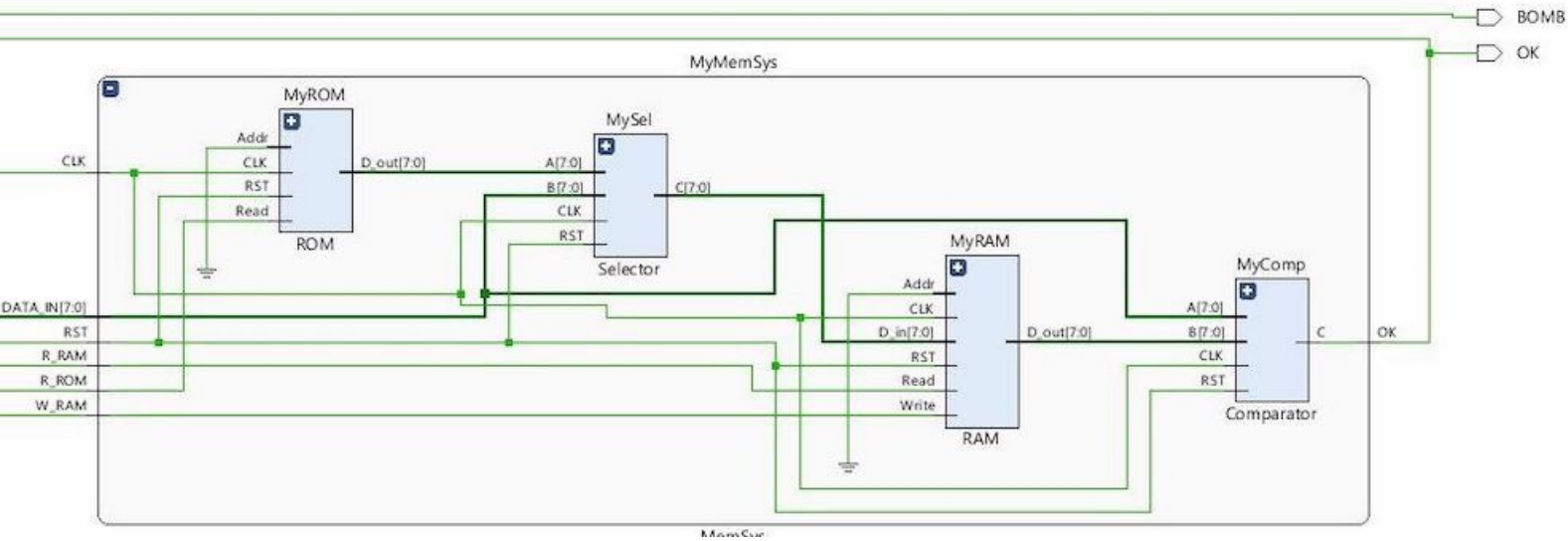
El diagrama nos muestra que hasta que el usuario no pulsa CHECK o WRITE no se realiza ningún cambio de estado, excepto en el estado inicial, que se realiza en primer lugar sin ninguna condición, y su estado siguiente siempre será S1. Además, con el botón de RESET también se retorna al estado S1, ya que si el usuario hubiera cambiado la contraseña, no queremos perderla.

2.2 Componentes definidos

El componente System contiene una FSM y un MemSys.



El componente MemSys contiene dos memorias: ROM y RAM, un selector para la entrada de datos a la RAM, ya que en el estado inicial usara datos de la ROM y en el resto estados solo usará datos del usuario. También contiene un comparador para hacer la comprobación de contraseña. Su salida es OK, y sirve a su vez de entrada para FSM.



3. Código VHDL

La maquina de estados:

```
17      architecture Structural of FSM is
18          -- Define FSM States
19          type state_type is (S0, S1, S2, S3, S4, S5);
20          signal currentState, nextState : state_type;
21          signal top : integer := 0;
22          -- Define intern signals
23
24      begin
25              -- State Change Process
26          process(CLK, RST)
27          begin
28              if (rising_edge(CLK)) then
29                  if (RST = '1') then
30                      currentState <= S0;
31                  else
32                      currentState <= nextState;
33                  end if;
34              end if;
35          end process;
36
37          -- FSM Process
38          process(currentState,CHECK, WRITE, OK)
39          begin
40              case currentState is
41                  -- Transfer Data Content from ROM to RAM
42                  when S0 =>
43
44                      R_ROM <= '1'; W_RAM <= '1'; R_RAM <= '0';
45                      BOMB <= '0';
46                      DATA_IN <= "ZZZZZZZZ";
47                      top <= top + 1;
48                      if(top < 2)then
49                          nextState <= S0;
50                      else
51                          nextState <= S1;
52                          top <= 0;
53                      end if;
54                  when S1 =>
55                  if(OK = '1') then
56                      nextState <= S5;
57                  else
58                      if(CHECK = '1' and WRITE = '0') then
59                          R_RAM <= '1'; W_RAM <= '0'; R_ROM <= '0';
60                          DATA_IN <= DATA;
61                          if(OK = '1') then
62                              nextState <= S5;
63                          else
64                              nextState <= S2;
65                          end if;
66                      elsif(WRITE = '1' and CHECK = '0') then
67                          R_RAM <= '0'; W_RAM <= '1'; R_ROM <= '0';
68                          nextState <= S1;
69                      else
70                          nextState <= S1;
71                      end if;
72                  end if;
```

```
73                  when S2 =>
74                      if(OK = '1') then
75                              nextState <= S5;
76                      else
77                          if(WRITE = '0') then
78                              if(CHECK = '1' )then
79                                  R_RAM <= '1'; W_RAM <= '0'; R_ROM <= '0';
80                                  DATA_IN <= DATA;
81                                  if(OK = '1') then
82                                      nextState <= S5;
83                                  else
84                                      nextState <= S3;
85                                  end if;
86                              end if;
87                          end if;
88                      end if;
89                  when S3 =>
90                  if(OK = '1') then
91                              nextState <= S5;
92                  else
93                      if(WRITE = '0') then
94                          if(CHECK = '1' )then
95                              R_RAM <= '1'; W_RAM <= '0'; R_ROM <= '0';
96                              DATA_IN <= DATA;
97                              if(OK = '1') then
98                                nextState <= S5;
99                             else
100                                nextState <= S4;
101                            end if;
102                        end if;
103                    end if;
104                end if;
105                when S4 =>
106                if(OK = '1') then
107                         nextState <= S5;
108                else
109                    if(WRITE = '0') then
110                        if(CHECK = '1' )then
111                            R_RAM <= '1'; W_RAM <= '0'; R_ROM <= '0';
112                            DATA_IN <= DATA;
113                            nextState <= S5;
114                        end if;
115                    end if;
116                end if;
117                when S5 =>
118                -- Check
119                    BOMB <= '1';
120                    report "Sanity check : OK = " & std_logic'image(OK);
121                    R_RAM <= '0'; W_RAM <= '0'; R_ROM <= '0';
122                    DATA_IN <= "ZZZZZZZZ";
123                    nextState <= S5;
124                when others =>
125                    nextState <= S0;
126            end case;
127        end process;
128    end Structural;
129
```

3.1 Test-bench

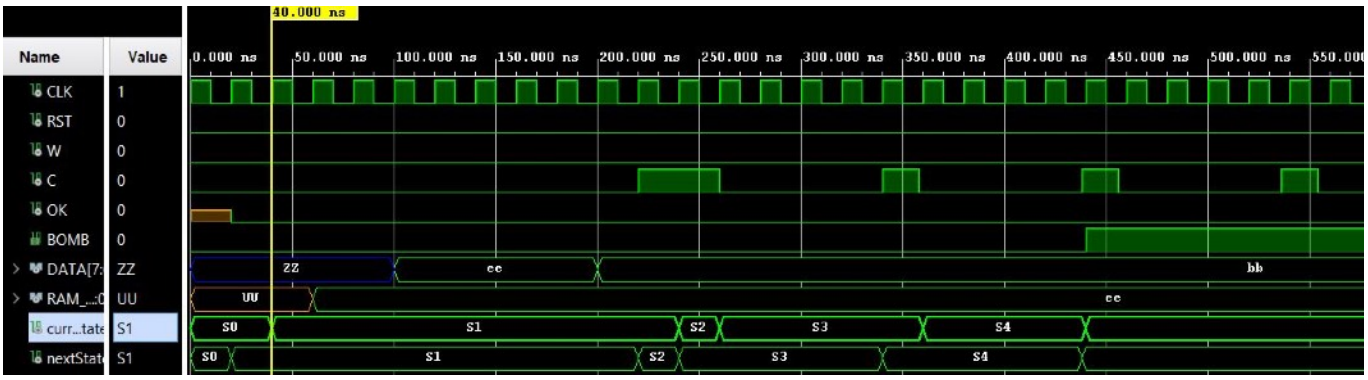
Estos son los dos usados para la simulación explicada a continuación:

```
9           Port ( CLK, RST, W, C : in std_logic;
10            DATA      : in std_logic_vector(7 downto 0);
11            OK        : out std_logic
12            );
13      end component;
14      signal CLK, RST, W, C, OK : std_logic;
15      signal DATA  : std_logic_vector(7 downto 0);
16  begin
17      MySystem: System port map(CLK=>CLK, RST=>RST, W=>W,
18                                C=>C, DATA=>DATA, OK=>OK);
19    process begin
20          CLK <= '1'; wait for 10 ns;
21          CLK <= '0'; wait for 10 ns;
22      end process;
23      process begin
24          RST <= '0'; wait;
25      end process;
26      process begin
27          C <= '0'; wait for 220 ns;
28          C <= '1'; wait for 40 ns;
29          C <= '0'; wait for 80 ns;
30          C <= '1'; wait for 18 ns;
31          C <= '0'; wait for 80 ns;
32          C <= '1'; wait for 18 ns;
33          C <= '0'; wait for 80 ns;
34          C <= '1'; wait for 18 ns;
35          C <= '0'; wait for 80 ns;
36          C <= '1'; wait for 18 ns;
37          C <= '0'; wait for 80 ns;
38
39
40
41      end process;
42      process begin
43          W <= '0'; wait;
44      end process;
45      process begin
46          DATA <= "ZZZZZZZZ"; wait for 100ns;
47          DATA <= X"EE"; wait for 100 ns;
48          DATA <= X"BB"; wait;
```

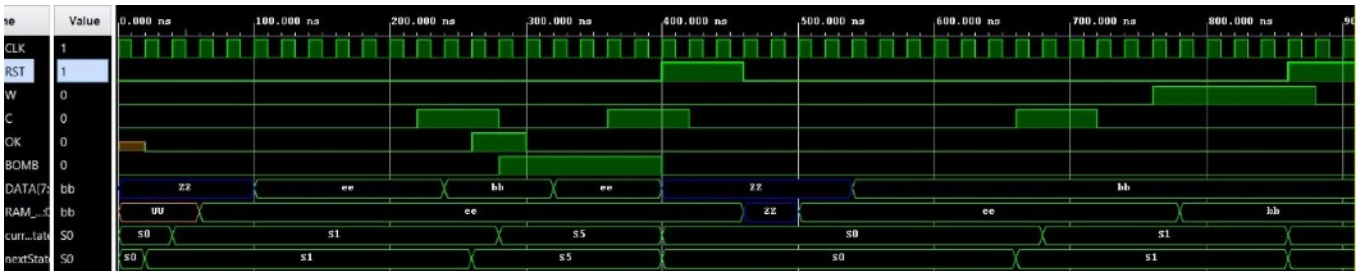
```
17          MySystem: System port map(CLK=>CLK, RST=>RST, W=>W,
18                                    C=>C, DATA=>DATA, OK=>OK);
19        process begin
20            CLK <= '1'; wait for 10 ns;
21            CLK <= '0'; wait for 10 ns;
22        end process;
23        process begin
24            RST <= '0'; wait for 400 ns;
25            RST <= '1'; wait for 60 ns;
26        end process;
27        process begin
28            C <= '0'; wait for 220 ns;
29            C <= '1'; wait for 60 ns;
30            C <= '0'; wait for 80 ns;
31            C <= '1'; wait for 60 ns;
32            C <= '0'; wait for 80 ns;
33            --primera parte hasta el final del reset
34            --segunda parte 500 ns.
35            C <= '0'; wait for 160 ns;
36            C <= '1'; wait for 60 ns;
37            C <= '0'; wait;
38
39
40        end process;
41        process begin
42            W <= '0'; wait for 760 ns;
43            W <= '1'; wait for 120 ns;
44            W <= '0'; wait;
45        end process;
46        process begin
47            DATA <= "ZZZZZZZZ"; wait for 100ns;
48            DATA <= X"EE"; wait for 140 ns;
49            DATA <= X"BB"; wait for 80 ns;
50            DATA <= X"EE"; wait for 80 ns;
51            DATA <= "ZZZZZZZZ"; wait for 140ns;
52            --parte2:
53            DATA <= X"BB"; wait for 400 ns;
54            DATA <= X"EE"; wait for 80 ns;
55        end process;
56    end Behavioral;
57
```

Con el test-bench compruebo el correcto funcionamiento de todas las funciones del sistema. Han sido probadas las tres posibilidades. En todos ellos, el sistema se incializa con la copia de ROM a RAM.

1. Introducir tres veces la contraseña incorrecta, de manera que pasamos por los 5 estados de la maquina de estado. Bomb se activa indicando el estado de bloqueo. Ok siempre permanece apagada.



2. El usuario mete la contraseña correcta, el sistema te lleva al estado de bloqueo activando de esta manera Bomb. El usuario pulsa reset para salir del estado de bloqueo. El usuario, esta vez, cambia la contraseña. Se puede observar como RAM content pasa de 0xEE a 0xBB.



4. Análisis de tiempos, área y consumo

1. Análisis de área

   Se observa que este circuito implementa bastante cantidad de de LUTs, Registers y IOBs.

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)
N System	40	42	14	40	18	14	1
MyFSM (FSM)	22	18	9	22	8	0	0
MyMemSys (MemSys)	17	20	7	17	8	0	0

2. Análisis de tiempos(*STA: Static time Analyze*)

   Se ha establecido un periodo de 4,620 de manera que se pudiera reducir al máximo pero teniendo en cuenta que el Worst Negative Slack nunca quede por debajo de 0. Esto implicaría que los datos no permanezcan estables durante el suficiente tiempo y llevaría a error.

Position	Clock Name	Period (ns)	Rise At (ns)	Fall At (ns)	Add Clock	Source Objects
1	Clock	4.620	0.000	2.310	☐	[get_ports CLK]

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,004 ns	Worst Hold Slack (WHS):	0,176 ns	Worst Pulse Width Slack (WPWS):	1,810 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	20	Total Number of Endpoints:	20	Total Number of Endpoints:	27

All user specified timing constraints are met.

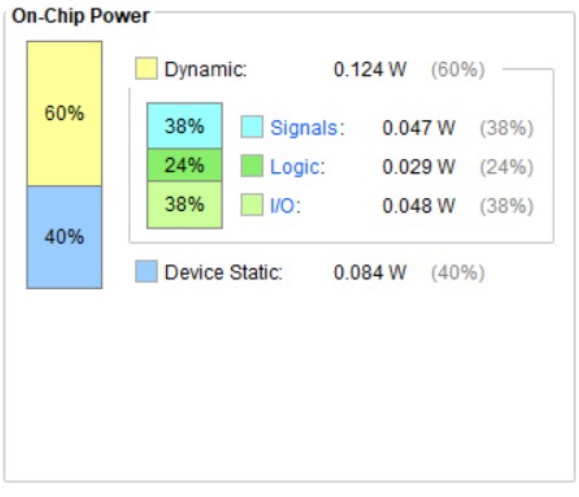
3. Análisis de consumo

   Aquí podemos observar tanto el consumo on-chip (consumo total), como el estático y dinámico y porcentaje que ocupan cada uno. Ademas se muestra la temperatura de unión, que es la temperatura que alcanza la fuga durante su funcionamiento.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.208 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,0°C**
Thermal Margin:	59,0°C (12,8 W)
Effective ϑJA:	4,6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



5. Conclusiones

Este trabajo ha consistido en la creación de una máquina de estados finita que sea capaz de realizar el funcionamiento de un sistema de comprobación de contraseña. Tras mucho trabajo, ha sido posible contemplar todas las posibilidades que se pedían en el trabajo y la máquina realiza todas las funciones.
He encontrado bastantes dificultades a la hora de querer reflejar mi idea, pero he aprendido muchas cosas durante este proceso y sobre todo, las muchas formas de plantear una misma idea y llevarlas acabo.