Diseño y simulación de un autómata de control para la apertura de una puerta accionada por un código secreto utilizando maquinas de estados

Universidad Nacional Autónoma de México
Facultad de Estudios Superiores Cuautitlán
Alonso Vargas Gachuz

1. Introducción

Los sistemas digitales más tradicionales funcionan por medio de la combinación de las entradas de las que disponen, así es posible obtener una salida de una puerta lógica por ejemplo, combinando las entradas para obtener un resultado, un problema del que disponen esta clase de sistemas es que son usados para obtener salidas muy especificas a partir de sus entradas por lo que siempre serán condicionados a respuestas definidas y limitadas a la cantidad de elementos con los que se puedan construir. Durante mucho tiempo se ha buscado la automatización en las tareas cotidianas y en procesos de trabajo de modo que estas se realicen independientemente de nuestra supervisión, para lograrlo se han usado y probado diferentes métodos de entre los cuales los sistemas digitales son los que más se han acercado, pero ¿cómo es posible que un sistema limitado a resultados concretos pueda ser autónomo?, el principal problema de los sistemas combinacionales es que carecen de memoria y no pueden realizar acciones en base a los resultados que se tienen de modo que pueda tomar la mejor decisión, para solucionar esto es que se han diseñado los sistemas tipo "latchçapaces de guardar y mantener registros de las acciones que se lleven, con ayuda de este tipo de dispositivos es que se han creado las llamadas máquinas de estados las cuales dependen de las entradas y los estados anteriores creando así los sistemas combinacionales. Con esta clase de sistemas es posible crear autómatas que cumplan desempeñen tareas de forma más cómoda en las que entreguen mejores salidas en función de las señales que vaya presentando. Para comprobar la funcionalidad de los sistemas combinacionales se diseñará uno que controle la apertura de una puerta por medio de un código secreto el cual debe ser programado, para abrir o mantener cerrado el mecanismo se deben ingresar todos los dígitos del código antes de mostrar si se ingresaron los correctos o incorrectos. Para facilitar la comprobación todo el sistema será diseñado por medio de un lenguaje de descripción de hardware.

2. Descripción del método

Para el diseño del sistema se usó la metodología de arriba hacia abajo por lo que partimos de una idea general del sistema desde donde podremos diseñar cada uno de los elementos que componen al sistema, el diagrama de la solución es el siguiente:

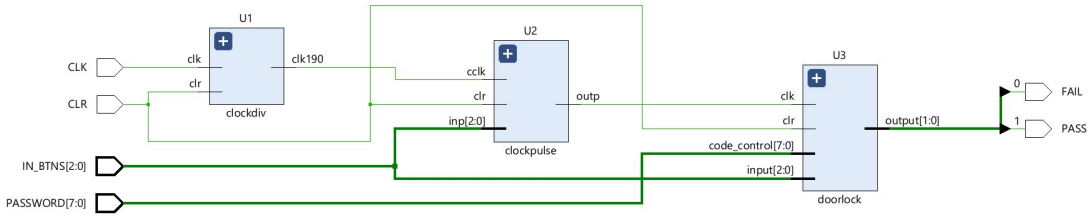

Figura 1: Diagrama general de la solución

Esencialmente el funcionamiento del autómata dependerá de una máquina de estados que será quien determine si el código es correcto o incorrecto. Dado que los sistemas secuenciales necesitan trabajar con señales de reloj que registran los cambios de estados y que el sistema está

pensado para implementarse con un reloj de 100MHz será necesario agregar un divisor de frecuencia y un sistema de generación de pequeños pulsos cada vez que se oprima cualquier botón. El divisor de frecuencia nos permite trabajar con una velocidad de reloj más pequeña, si se trabajara con la velocidad normal seria imposible registrar los cambios adecuadamente debido a la velocidad, la frecuencia a la que se planea reducir el reloj es a 190 Hz que se explicará más adelante.
El problema de utilizar un reloj que oscile libremente radica en que los circuitos secuenciales se mantendrán receptivos a ella constantemente lo cual no es deseable pues unicamente nos interesa registrar los cambios cada vez que se oprima un botón, para adecuar la señal de reloj y volverla sensible a las señales de entrada se implementará un generador de pulsos el cual creará retardos en la señal de reloj de entrada por cada vez que se pulse un botón, para que se pueda generar un pulso utilizando los retardos se debe trabajar con una velocidad de reloj adecuada ya que si fuera muy lenta el sistema será incapaz de generar los pulsos debido a la velocidad, es por ello que el mínimo requerido es de 190Hz razón por la que el divisor empleado reduce el reloj principal a dicha frecuencia.
Una vez resueltos los sistemas de reloj para registrar los cambios pasamos al diseño del autómata que controlará la apertura de la puerta. Para diseñar el diagrama de estados partimos primero por la ruta correcta que abrirá la puerta con los dígitos correctos, dado que el código es programable no podemos establecer valores únicos para los cambios entre estados por lo que debemos tratarlos como variables y dado que existe una única forma correcta de introducir los datos podemos simplemente hacer un caso particular para los valores correctos e intercambiar entre el caso en que sean incorrectos cuando se ingrese cualquier valor incorrecto, para poder ingresar los cuatro valores antes de saber si el código es correcto se utilizan cuatro estados en caso de que sean correctos y otros cuatro en caso que sean incorrectos más un estado inicial por lo que se necesitaron nueve estados para la solución, el diagrama resultante es el siguiente:

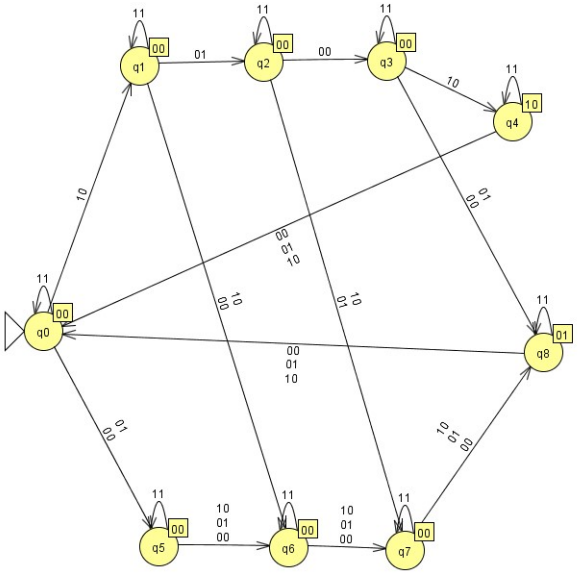

Figura 2: Diagrama de estados: Para el ejemplo se usó el código "2102", se pueden observar las dos rutas planteadas.

De esta forma se registran las cuatro entradas para cualquiera de los dos casos, el diagrama también fue planeado para que se cambie de ruta en caso de que los primeros valores sean correctos pero los siguientes no. Para registrar y programar la contraseña se usaron señales que guardarán los valores ingresados por medio de los switches siete a cero. Por último se realiza la descripción del autómata en VHDL. Para ello se requieren de dos unidades principales: la red combinacional y el registro de estados. Para detalles el código se encuentra en el apéndice B.

3. Experimentos

El diseño e implementación del sistema fue realizado en la herramienta Vivado la cual permite visualizar el esquema del diseño además de los diagramas de tiempo de las señales, las capturas de los resultados obtenidos se encontrarán en el apéndice C de este documento.

4. Conclusiones

Como se mencionó antes los sistemas combinacionales carecen de registros de memoria con los cuales seria imposible solucionar problemas que requieran de esta para resolver problemas, con la adición de los sistemas tipo latch es posible crear autómatas y todo tipo de sistemas combinacionales como una cerradura en la que se guarden los dígitos ingresados tal como el ejemplo que hemos desarrollado. Debido a que este tipo de circuitos es muy difícil de conseguir ademas de ser costosos es por lo que podemos optar por usar un lenguaje de descripción de hardware con el que podamos describir el esta clase de dispositivos por lo que es importante saber como se deben generar e implementar. En el desarrollo de este trabajo hemos comprado la ventaja del uso de dispositivos programables en el diseño de sistemas digitales en este caso de un sistema secuencial.

Referencias

M. M. Mano, *Digital Design*, 2a ed. Englewood Cliffs, N.J: Prentice-Hall, 1991.

5. Apéndice A:

Diagramas detallados de la solución:

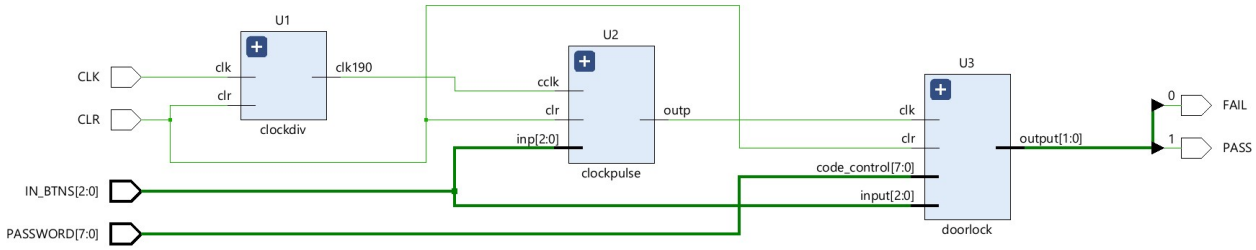

Figura 3: Diagrama general de la solución

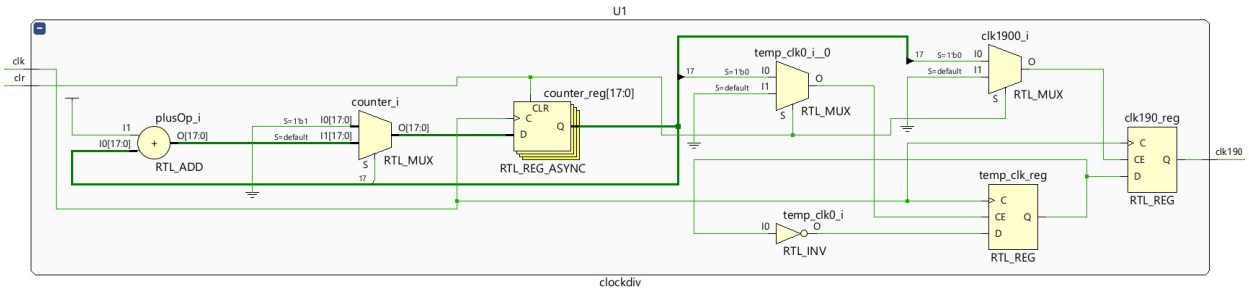

Figura 4: Divisor de Frecuencia

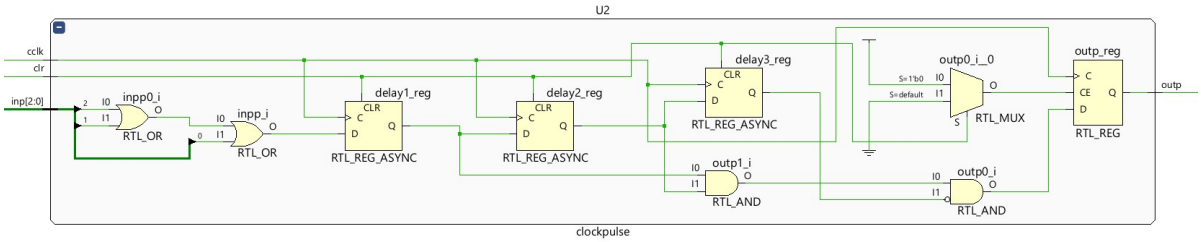

Figura 5: Generador de pulsos

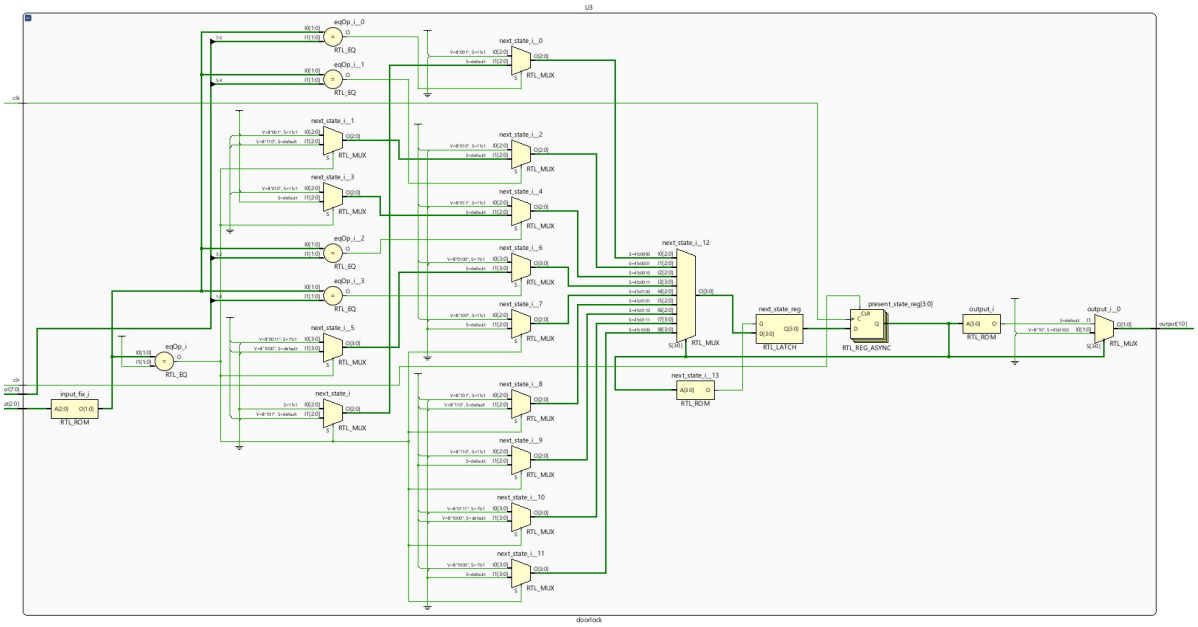

Figura 6: Sistema de control para la apertura de la puerta

6. Apéndice B

Código en VHDL
Modulo pass_control_top:

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

entity pass_control_top is
    port(   CLK : in std_logic;
            CLR : in std_logic;
            PASSWORD : in std_logic_vector (7 downto 0);
            IN_BTNS : in std_logic_vector (2 downto 0);
            FAIL : out std_logic;
            PASS : out std_logic);
end pass_control_top;

architecture Behavioral of pass_control_top is
signal out_clk190 : std_logic;
signal pulse : std_logic;

begin
U1: entity work.clockdiv port map(
        clk => CLK,
        clr => CLR,
        clk190 => out_clk190
);

U2: entity work.clockpulse port map(
        cclk => out_clk190,
        clr => CLR,
        inp => IN_BTNS,
        outp =>  pulse
);

U3: entity work.doorlock port map(
        clk => pulse,
        clr => CLR,
        code_control => PASSWORD,
        input => IN_BTNS,
        output(1) => PASS,
        output(0) => FAIL
);
end Behavioral;
```

Divisor de frecuencia:

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

entity clockdiv is
    port(   clk : in std_logic;
            clr : in std_logic;
            clk190: out std_logic);
end clockdiv;

architecture Behavioral of clockdiv is
signal counter : std_logic_vector (17 downto 0);
signal temp_clk : std_logic := '1';

begin
clock: process(clk,clr)
    begin
        if(clr = '1') then
            counter <= (others => '0');
        elsif (clk'event and clk = '1') then
            counter <= counter+1;
            if (counter(17) = '1') then
                temp_clk <= not temp_clk;
                clk190 <= temp_clk;
                counter <= (others => '0');
            end if;
        end if;
    end process clock;

end Behavioral;
```

Generador de pulsos:

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

entity clockpulse is
    port(   cclk : in std_logic;
            clr : in std_logic;
            inp : in std_logic_vector (2 downto 0);
            outp : out std_logic);
end clockpulse;

architecture Behavioral of clockpulse is
signal inpp : std_logic;
signal delay1, delay2, delay3 : std_logic;

begin
    inpp <= inp(2) or inp(1) or inp(0);
```

```
process(cclk,clr)
    begin
    if(clr = '1') then
        delay1 <= '0';
        delay2 <= '0';
        delay3 <= '0';
    elsif (rising_edge(cclk)) then
        delay1 <= inpp;
        delay2 <= delay1;
        delay3 <= delay2;
        outp <= delay1 and delay2 and (not delay3);
    end if;
end process;
end Behavioral;
```

Control de cerradura:

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

entity doorlock is
    port(   clk : in std_logic;
            clr : in std_logic;
            code_control : in std_logic_vector (7 downto 0);
            input : in std_logic_vector (2 downto 0);
            output : out std_logic_vector (1 downto 0));
end doorlock;

architecture Behavioral of doorlock is
type state_type is (q0, q1, q2, q3, q4, q5, q6, q7, q8);
signal present_state, next_state : state_type;
signal input_fix: std_logic_vector (1 downto 0);
signal digit3, digit2, digit1, digit0 : std_logic_vector (1 downto 0);

begin
--Programacion del c digo de la cerradura
    digit3 <= code_control(7) & code_control(6);
    digit2 <= code_control(5) & code_control(4);
    digit1 <= code_control(3) & code_control(2);
    digit0 <= code_control(1) & code_control(0);
--Arreglo para las entradas de los botones
with input select
    input_fix <=    "00" when "001",
                    "01" when "010",
                    "10" when "100",
                    "11" when others;
--Registro de estados
state_register: process(clk,clr)
    begin
    if (clr = '1') then
        present_state <= q0;
    elsif rising_edge(clk) then
```

```
        present_state <= next_state;
    end if;
end process state_register;
-- Proceso Combinacional de estados
C1: process(present_state,input_fix,digit3,digit2,digit1,digit0)
    begin
    case (present_state) is
        when q0 =>
            if (input_fix = digit3) then
                next_state <= q1;
            elsif (input_fix = "11") then
                next_state <= q0;
            else
                next_state <= q5;
            end if;
        when q1 =>
            if (input_fix = digit2) then
                next_state <= q2;
            elsif (input_fix = "11") then
                next_state <= q1;
            else
                next_state <= q6;
            end if;
        when q2 =>
            if (input_fix = digit1) then
                next_state <= q3;
            elsif (input_fix = "11") then
                next_state <= q2;
            else
                next_state <= q7;
            end if;
        when q3 =>
            if (input_fix = digit0) then
                next_state <= q4;
            elsif (input_fix = "11") then
                next_state <= q3;
            else
                next_state <= q8;
            end if;
        when q4 =>
            if (input_fix = "11") then
                next_state <= q4;
            else
                next_state <= q0;
            end if;
        when q5 =>
            if (input_fix = "11") then
                next_state <= q5;
            else
                next_state <= q6;
            end if;
```

```
        when q6 =>
            if (input_fix = "11") then
                next_state <= q6;
            else
                next_state <= q7;
            end if;
        when q7 =>
            if (input_fix = "11") then
                next_state <= q7;
            else
                next_state <= q8;
            end if;
        when q8 =>
            if (input_fix = "11") then
                next_state <= q8;
            else
                next_state <= q0;
            end if;
        when others =>
            null;
    end case;
end process C1;
-- Proceso secuencial para registrar la salida
C2: process(present_state)
    begin
    if (present_state = q4) then
        output <= "10";
    elsif (present_state = q8) then
        output <= "01";
    else
        output <= "00";
    end if;
end process C2;
end Behavioral;
```

7. Apéndice C

Resultados de la simulación

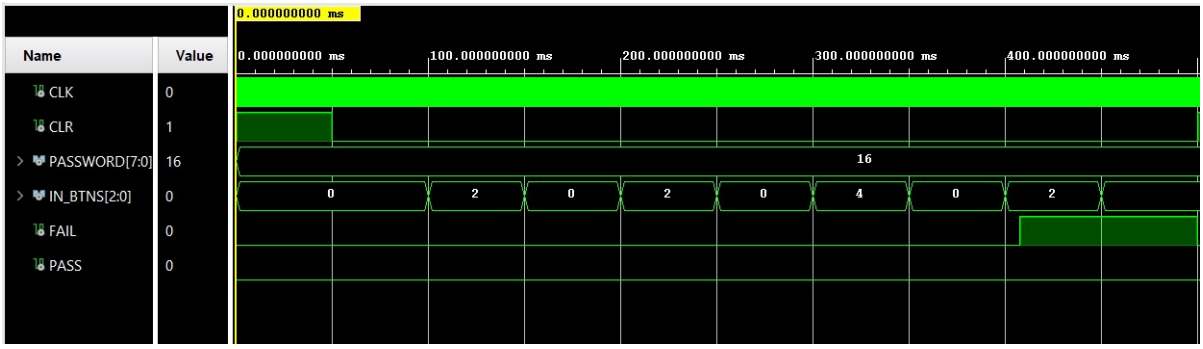

Figura 7: Simulación para el código "0112", los valores en la simulación están codificados según la entrada y su representación es la siguiente : 4 representa al dígito 2, 2 representa al dígito 1, 1 representa al dígito 0, el valor 0 representa que no se está ingresando ningun valor. Para este caso los dígitos ingresados son incorrectos

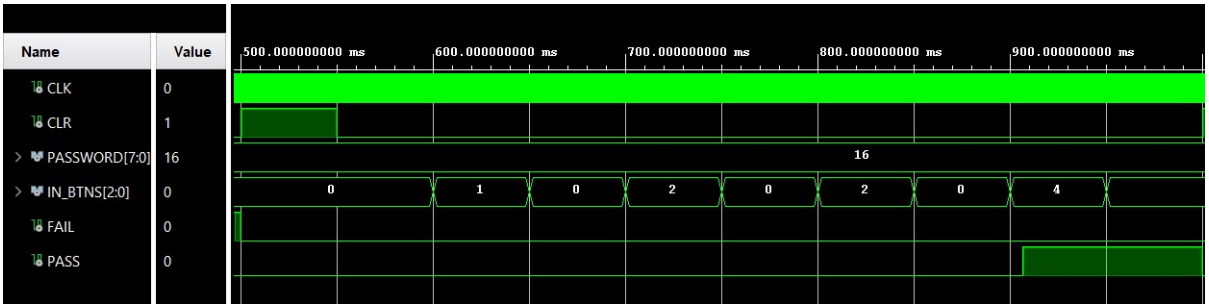

Figura 8: Simulación ingresando los datos correctos

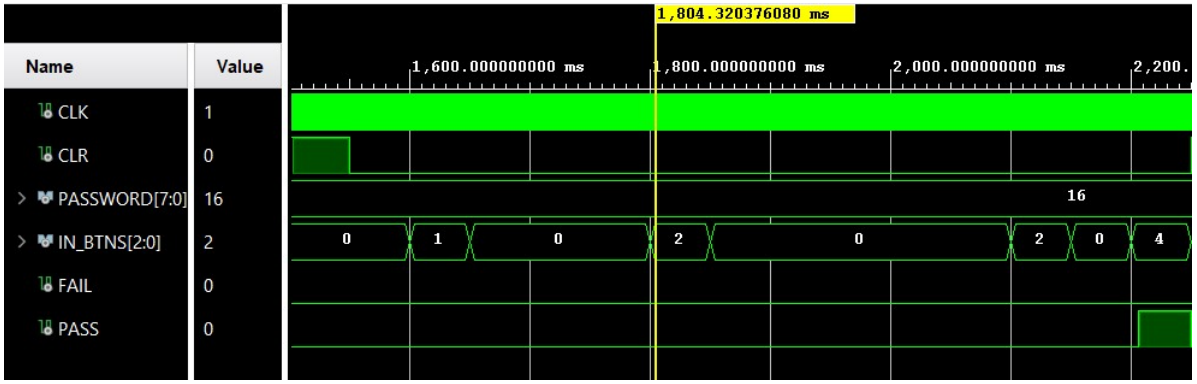

Figura 9: Simulación del sistema con retardos entre entradas, se observa que el sistema continua funcionando