Automata en VHDL.
Control de un puerta a través de un código

Universidad Nacional Autónoma de México.
Facultad de Estudios Superiores Cuatitlán.
Palomino Alfonso Edgar.

18 de septiembre de 2023

1. Introducción.

El control de acceso es un aspecto fundamental en muchos sistemas o aplicaciones modernas. En este contexto, la implementación de un sistema en VHDL utilizando Vivado para controlar el acceso a una puerta mediante un código secreto es una planteamiento interesante a resolver. Este pequeño proyecto busca combinar la flexibilidad con la potencia de la programación en VHDL, así como la versatilidad de Vivado para crear un sistema que garantice la seguridad de una puerta. Esto se busca a través de explorar en detalle el diseño de un autómata en VHDL que será responsable de gestionar el acceso a una puerta. Para ello, utilizaremos un código secreto que los usuarios autorizados deberán ingresar correctamente para que la puerta se abra (esto se indicará a través de un led). Este proyecto se enfocará en los aspectos clave del diseño, incluyendo la descripción del sistema en VHDL además de la simulación en una tarjeta Basys 3 utilizando Vivado.

En este documento, abordaremos los siguientes puntos clave: presentaremos el diseño en VHDL de un autómata que gestione este acceso, detallando la lógica del código secreto además de las acciones en caso de éxito o fracaso. Discutiremos la relevancia de la simulación y verificación para garantizar la seguridad del sistema. Finalmente, resumimos los resultados, destacando las ventajas así como los desafíos, considerando posibles mejoras futuras.

2. Descripción del método.

La metodología de este sistema es poco complicada, teniendo un grado de mayor dificultad proporcionar las salidas y entradas correctas para enlazar los código entre los módulos para una única salida. Abordando los módulos en el orden que me parece adecuado, podemos considerara que se requiere realizar un prescaler para que los cambios del reloj al leer los pulsos no sean muy rápidos, lo cual nos da la holgura necesaria en los módulos para leer de manera correcta todas las señales, generando las salidas de una manera satisfactoria. El módulo de "anti-rebote" obtiene mediante el módulo principal SYSTEM unos retrasos, que se verifican en el módulo de pulsos de reloj, el cual tiene como función principal, asegurarse de que los 3 retrasos se den sobre el reloj del preescaler, así los botones utilizados para las entradas quedan condicionados,

para que solo un señal de pulso se tome en cuenta, pasando el primer pulso entre los 3 retrasos, además se condiciona esta salida para verificar que todos sean iguales, generando de manera satisfactoria un solo pulso desde los botones de entrada, también en este módulo es donde se realiza la codificación, ya que se pasan los valores de los botones a los valores que utilizaremos sobre el código final. Contamos con el módulo de las máquinas de estado que es donde se manejan las situaciones posible en base al diseño principal como en la figura 1, el cual consiste en colocar digamos 2 "caminos" diferentes por lo cuales se puede llegar a una salida, esto debido a el planteamiento del problema donde se requiere que la salida se de hasta tener las 4 iteraciones que requiere el dígito para ser correcto, que si se considera, esto tiene sentido ya que al ser un sistema de seguridad no debería de dar pista de donde están los errores, así que se limita a tener una salida de *aprobado* o *fallo* en cada iteración completa del sistema, así que si pasamos por los estados $\{q_0, q_1, q_2, q_3\}$ generamos un valor correcto y obtendremos una señal de *aprobado*, mientras que si erramos algun digito, podemos pasar por los estados $\{q_4, q_5, q_6\}$, que manejan los errores para completar la iteración de 4 digitos, así mismo tenemos una particularidad que es que los valores intermedios pasan del camino correcto al manejo de errores en los estados $\{q_0, q_1, q_2\}$, pero en el estado q_3, el error se maneja sobre el mismo estado dando salida a diferentes valores, si el codigo es correcto activaremso la señal de *aprobado*, mientras que si el codigo esta mal sobre ese ultimo dígito daremos paso a la salida de *fallo* , por ultimo como se dijo el modulo de *system*, une las señales pasando el reloj principal al preescaler, la salida de este modulo al modulo de pulsos e reloj, que a su vez da parte de las entradas al modulo de entirrebote, tambien se pasan los valores de los botones pulsados, asegurando la lectura de un solo valor y pasando los valores de estos a el modulo de las maquinas de estado, el cual como se dijo da paso a la señales aprobado o fallo, dichas señales se puede interpretar de manera fisica sobre la tarjeta al ver el led_0 para fallo y led_1 para aprobado, el valor del codigo a verificar es determinado por el vector *secret_ code* que se lee sobre este mismo modulo. dicho valor se obtiene a través de los switch 0 a 7 de la tarjeta *Basys 3*.

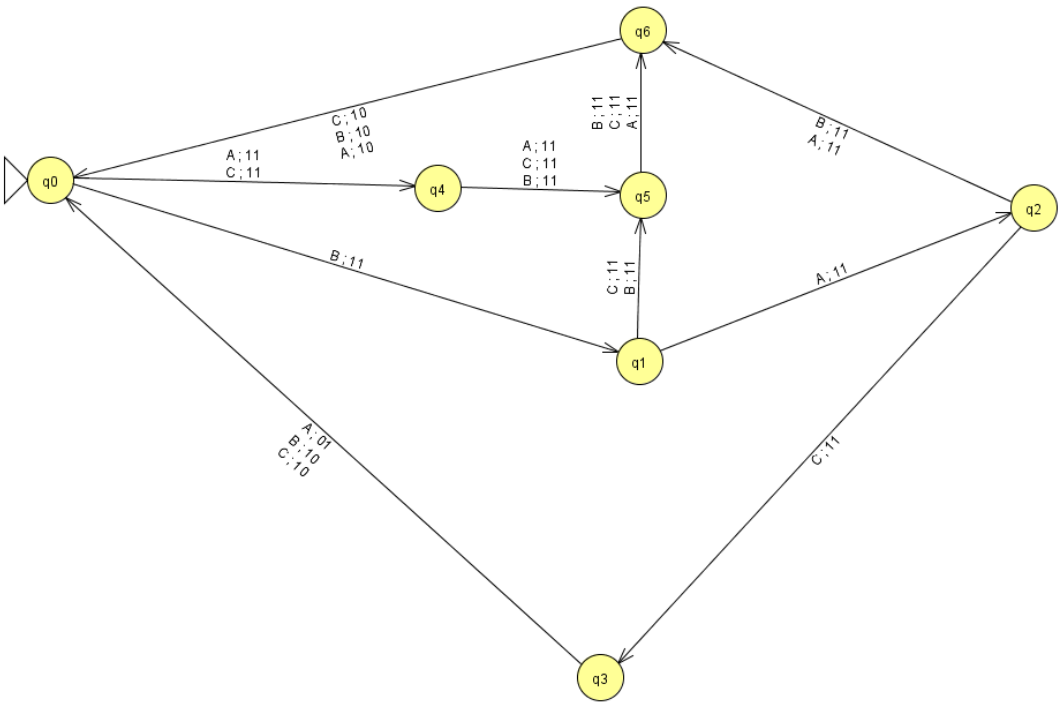

Figura 1: Diseño de la maquina de estados

3. Experimentos

La imagen de las salidas se dejan en el apéndice A con la figura 3, donde se pueden ver las diferentes señales que comprueban el correcto funcionamiento del sistema. Comenzando con la figura 4, podemos apreciar que el valor del SECRET CODE está dispuesto a 24, pero en base hexadecimal, lo que corresponde en binario a *100100*, pero como se aprecia, el valor del arreglo es de 8 bits, así que es *00100100*, como se dijo se toman los valores para comparar desde los switches 1 a 4, con lo que el primer conjunto de 2-bits da el valor de el primer *"dígito"* 00, después 01, despues 10 y por ultimo 00, así que el código secreto será *0120*, de esta manera podemos verificar que nuestro sistema funciona al pasarle dichos valores y tener un pulso en alto en la señal denominada *pass*, mientras que al pasar otro valor obtenemos un pulso en alto en la señal denominada *fail*, podemos notar también la configuración dispuesta de los botones de la tarjeta, donde 001 corresponde a un *dígito* a la salida de 0, un 010, corresponde a 1 y 100, corresponde a el dígito 2, por eso podemos observar para el código secreto los valores de entrada de *1241*, lo cual corresponde a la codificación que se ha realizado, de esta manera podemos verificar que todo el sistema es congruente, podemos pasar mas valores y jugar con ellos para analizar y determinar la funcionalida del sistema, cosa que se hizo y se deja como evidencia, solo los valores de un caso correcto y una falla, repetida sobre las iteraciones del sistema.

4. Conclusión

Este proyecto ha logrado el objetivo de crear un sistema de control de acceso seguro y eficiente mediante la combinación de VHDL en Vivado. Aunque la metodología puede parecer compleja en un principio, la modularidad de los componentes ha facilitado su desarrollo, así como el mantenimiento que se le pueda dar. Además, el sistema se ha diseñado teniendo en cuenta la seguridad y la usabilidad, lo que puede llevarlo a escalar a una solución sólida para el control de acceso a una puerta con código secreto. Como futuras mejoras, se podrían explorar posibilidades de la integración con sistemas de control más amplios, hemos logrado desarrollar un diseño modular, cuya estructura de módulos, que incluye el prescaler, el anti-rebote y las máquinas de estado, ha demostrado ser altamente efectiva en el manejo de las señales de entrada, así como en la generación de salidas adecuadas que podemos verificar mediante los diagramas de tiempos obtenidos de la simulación, también la implementación de la lógica de codificación asegura una comparación precisa con el código secreto, mediante la configuración que se pueda dar mediante los switch. La retroalimentación visual mediante LED simplifica la interacción con el sistema. La gestión de errores y la separación de caminos para aprobación y fallo mejoran aún más la seguridad, ya que como se comentó, esto deja sin pistas de donde está en el error, si bien el sistema es pequeño se puede considerar que, teniendo esto como base se podría desarrollar un sistema más robusto bajo la misma línea de pensamiento, cosa que dependiendo de la metodología de desarrollo que se tenga puede cambiar o no, que en lo personal creo que cambiaria, pero seria interesante ver cuales son esos cambios a efectuar sobre el sistema y como la mayoría de cosas en ingenieria, saber que tanto estábamos cerca de algo viable o no.

Apéndice A:
Imagenes de la simulación

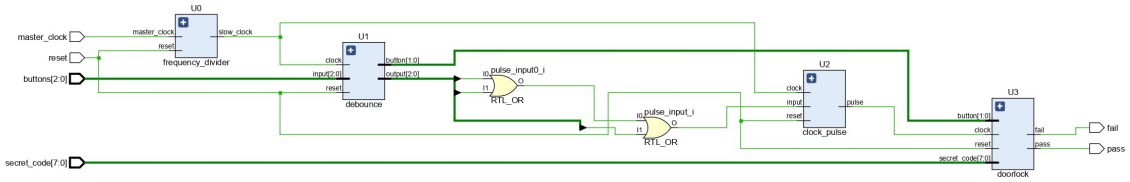

Figura 2: Diseño del Top-level

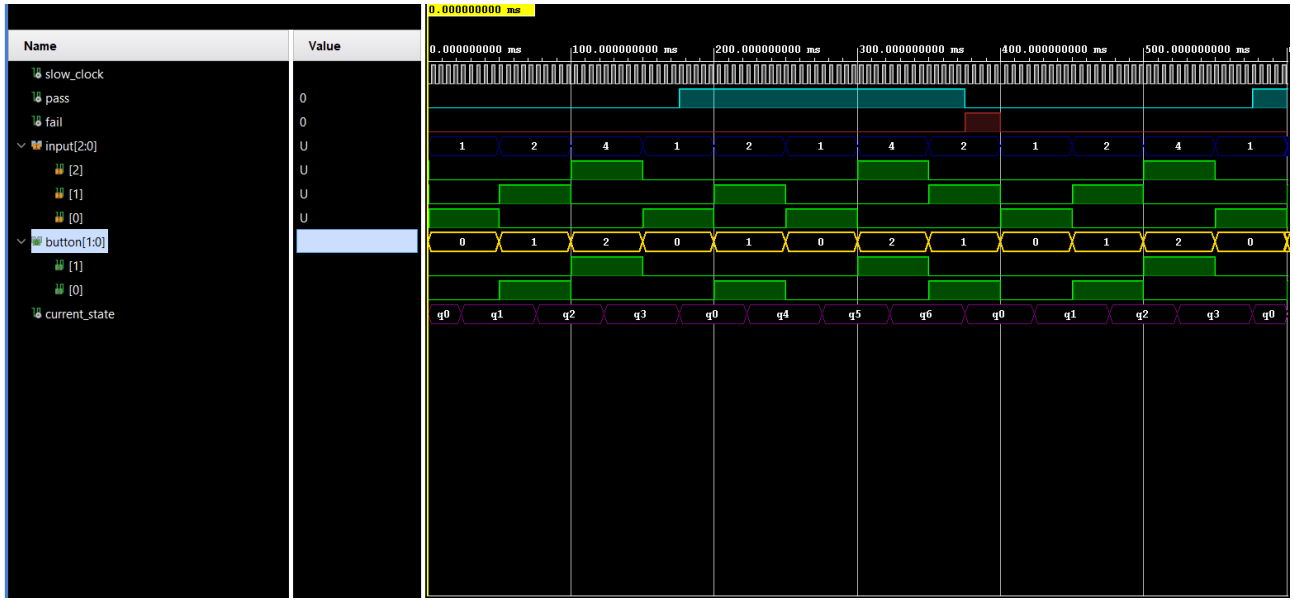

Figura 3: Señales de salida para el analisis

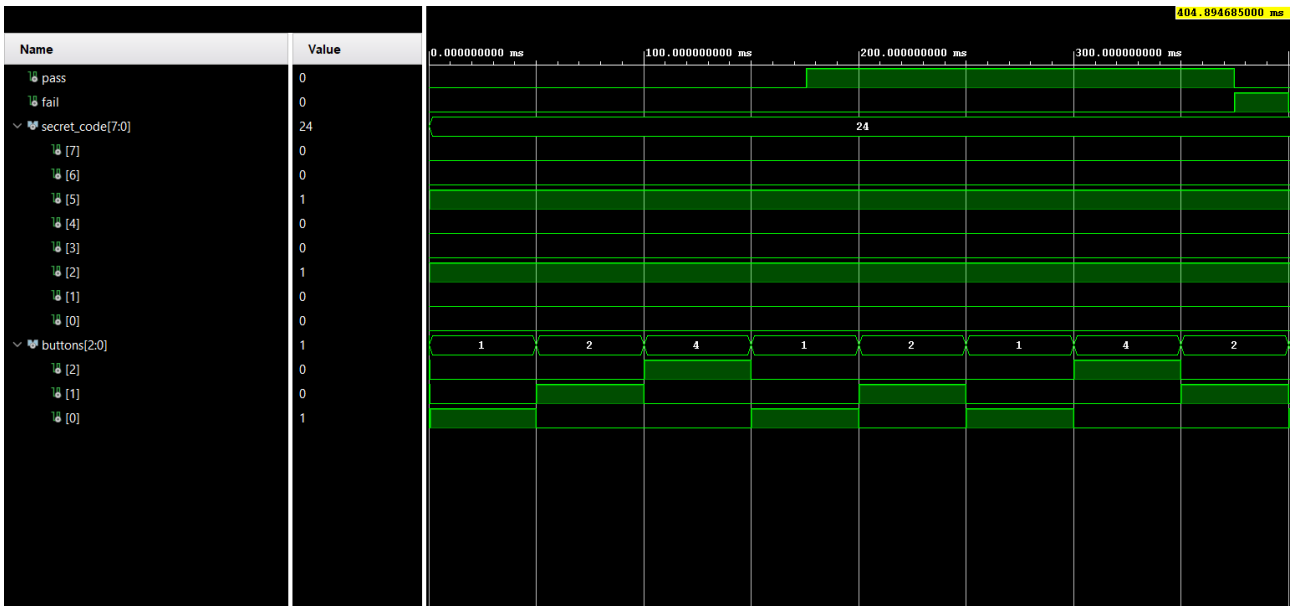

Figura 4: Valor del código secreto

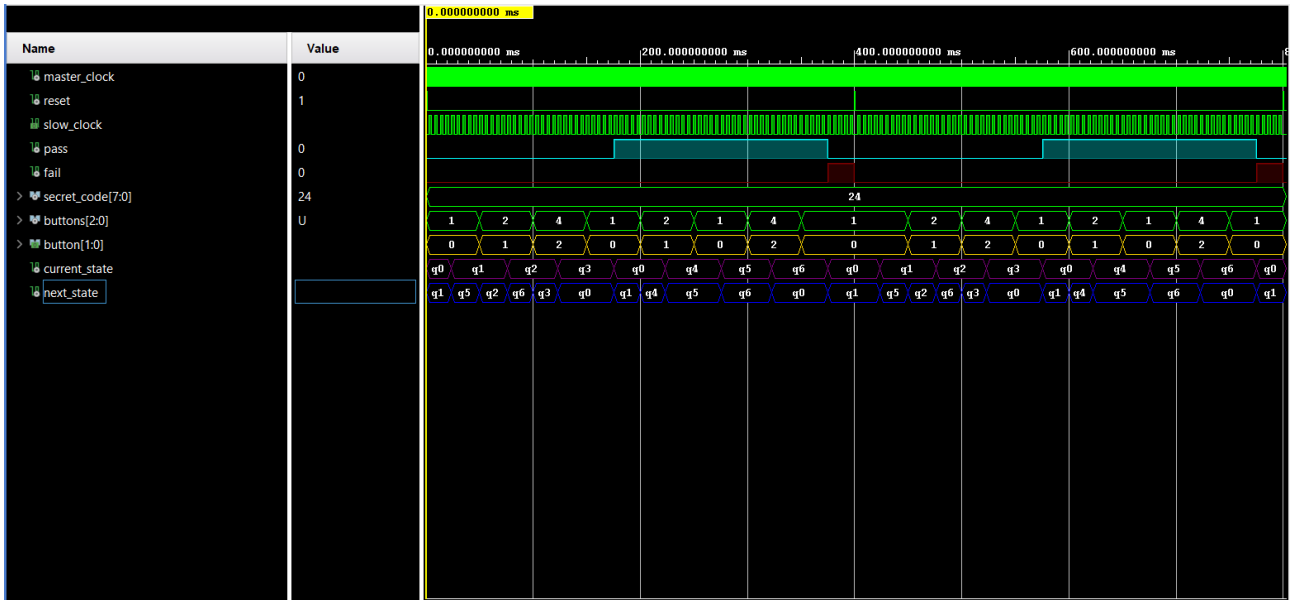

Figura 5: Señales de salida

Apéndice B: Código VHDL

Código del módulo con pulsos de reloj verificados

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

entity clock_pulse is
    port(
        clock, reset, input : in std_logic;
        pulse : out std_logic
    );
end clock_pulse;

architecture Behavioral of clock_pulse is
    signal first_delay, second_delay, third_delay : std_logic;
begin
    process(clock, reset)
    begin
        if reset = '1' then
            first_delay <= '0';
            second_delay <= '0';
            third_delay <= '0';
        elsif rising_edge(clock) then
            first_delay <= input;
            second_delay <= first_delay;
            third_delay <= second_delay;
        end if;
```

```
    end process;
    pulse <= first_delay and second_delay and (not third_delay);
end Behavioral;
```

Código del módulo eliminador de la señal de rebote de los botones

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

entity debounce is
    port(
        clock, reset : in std_logic;
        input : in std_logic_vector(2 downto 0);
        output : out std_logic_vector(2 downto 0);
        button : out std_logic_vector(1 downto 0)
    );
end debounce;

architecture Behavioral of debounce is
    signal first_delay, second_delay, third_delay :
       std_logic_vector(2 downto 0);
begin
    process(clock, reset)
    begin
        if reset = '1' then
            first_delay <= (others => '0');
            second_delay <= (others => '0');
            third_delay <= (others => '0');
        elsif rising_edge(clock) then
            first_delay <= input;
            second_delay <= first_delay;
            third_delay <= second_delay;
        end if;
    end process;
    output <= first_delay and second_delay and third_delay;
    with input select
        button <= "00" when "001",
                  "01" when "010",
                  "10" when "100",
                  "11" when others;
end Behavioral;
```

Código del módulo de la maquina de estados

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

entity doorlock is
    port(
```

```
        clock, reset : in std_logic;
        secret_code : in std_logic_vector(7 downto 0);
        button : in std_logic_vector(1 downto 0);
        pass, fail : out std_logic
    );
end doorlock;

architecture Behavioral of doorlock is
    type state_t is (q0, q1, q2, q3, q4, q5, q6);
    signal current_state, next_state : state_t;
begin
state_register:
    process(clock, reset)
    begin
        if reset = '1' then
            current_state <= q0;
        elsif rising_edge(clock) then
            current_state <= next_state;
        end if;
    end process state_register;

input:
    process(current_state, button)
        begin
            case current_state is
                when q0 =>
                    if button = secret_code(1 downto 0) then
                        next_state <= q1;
                    else
                        next_state <= q4;
                    end if;
                when q1 =>
                    if button = secret_code(3 downto 2) then
                        next_state <= q2;
                    else
                        next_state <= q5;
                    end if;
                when q2 =>
                    if button = secret_code(5 downto 4) then
                        next_state <= q3;
                    else
                        next_state <= q6;
                    end if;
                when q3 =>
                    next_state <= q0;
                when q4 =>
                    next_state <= q5;
                when q5 =>
```

```
                    next_state <= q6;
                when q6 =>
                    next_state <= q0;
                when others =>
                    null;
            end case;
    end process input;

output:
    process(clock, reset)
        begin
            if reset = '1' then
                pass <= '0';
                fail <= '0';
            elsif rising_edge(clock) then
                if (current_state = q3) and (button = secret_code
                   (7 downto 6)) then
                    pass <= '1';
                    fail <= '0';
                elsif (current_state = q3) or (current_state = q6
                   ) then
                    pass <= '0';
                    fail <= '1';
                end if;
            end if;
    end process output;
end Behavioral;
```

Código del módulo preescaler

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

entity frequency_divider is
    generic(
        prescaler : integer := 16
    );

    port(
        master_clock, reset: in std_logic;
        slow_clock : out std_logic
    );
end frequency_divider;

architecture Behavioral of frequency_divider is
    signal counter : std_logic_vector((prescaler - 1) downto 0);
begin
    process(master_clock, reset)
```

```
    begin
        if reset = '1' then
            counter <= (others => '0');
        elsif rising_edge(master_clock) then
            counter <= counter + 1;
        end if;
    end process;
    slow_clock <= counter(prescaler - 1);
end Behavioral;
```

Código del módulo top conjuntador de módulos

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

entity system is
    port(
        master_clock, reset : in std_logic;
        secret_code : in std_logic_vector(7 downto 0);
        buttons : in std_logic_vector(2 downto 0);
        pass, fail : out std_logic
    );
end system;

architecture Behavioral of system is
    signal slow_clock, pulse_input, clock_pulse: std_logic;
    signal debounced_buttons : std_logic_vector(2 downto 0);
    signal digit : std_logic_vector(1 downto 0);
begin
    U0: entity work.frequency_divider generic map(prescaler =>
       19) port map(master_clock => master_clock, reset => reset,
        slow_clock => slow_clock);
    U1: entity work.debounce port map(clock => slow_clock, reset
       => reset, input => buttons, output => debounced_buttons,
       button => digit);
    pulse_input <= debounced_buttons(2) or debounced_buttons(1)
       or debounced_buttons(0);
    U2: entity work.clock_pulse port map(clock => slow_clock,
       reset => reset, input => pulse_input, pulse => clock_pulse
       );
    U3: entity work.doorlock port map(clock => clock_pulse, reset
        => reset, secret_code => secret_code, button => digit,
       pass => pass, fail => fail);
end Behavioral;
```