Sistema en VHDL con suma, resta y comparación de 2 palabras a 4 bits

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

17 de septiembre de 2023

1. Introducción.

En el campo de la ingeniería y la informática, las operaciones aritméticas fundamentales de suma, resta y comparación desempeñan un papel esencial en una amplia gama de aplicaciones. En este proyecto, presentaremos el diseño y la implementación de un sistema digital en VHDL que realiza estas operaciones con palabras de 4 bits. Tanto el sumador como el restador se basan en un diseño tradicional de acarreo completo, mientras que el comparador utiliza un enfoque de comparación de magnitud de 4 bits.A comparación de la simulación en este sistema tiene la capacidad para manejar números en su totalidad, incluyendo valores negativos. A diferencia del diseños pasado que restringen las operaciones solo a números positivos, nuestro sistema en VHDL permite operar con números positivos y negativos, también el comparador tiene la tarea de determinar si dos números son iguales, si uno es mayor que el otro o si uno es menor que el otro. A lo largo de este informe, profundizaremos en el diseño y el funcionamiento de estas operaciones cruciales en el contexto de sistemas digitales basados en VHDL.

2. Descripción del método.

La solución propuesta para el sistema consiste en una entrada binaria a un sumador completo de 4 bits, el cual produce una salida en formato *BCD*. Esto es necesario ya que se requiere pasarlo a un formato BCD para mostrarlo en tres displays de 7 segmentos. Se condiciona la salida de este sumador de tal manera que si es menor que nueve, se le agregan ceros para tener 8 bits en la salida (4 bits para cada display de 7 segmentos). En cualquier otro caso, se verifica que la salida no sea mayor que nueve para asegurar una salida *BCD válida*. Si no se cumple esta condición, se le suman 6 en binario hasta obtener un BCD válido. Una vez que esto se cumple, se pueden segmentar las combinaciones necesarias en 4 bits para su representación en los displays de 7 segmentos. La resta presenta el problema de que si se requiere representar números negativos de 2 dígitos decimales, se necesitaría un display adicional para representar el signo, lo cual se contempla en este sistema. Entonces, si $x > y$, se realiza la resta y se representa de la misma manera que en la suma, en caso contrario se realiza el complemento a 2 para obtener el valor correspondiente a la resta verificando que el resultado sea menor a 10, en este caso el tercer

display mostrará el signo para el resultado. Para el comparador, se necesita un cambio ligero en la lógica, ya que debe mostrar *'G'* si $x > y$, *'E'* si $x = y$ y *'L' six* $< y$. Por lo tanto, para cada caso se debe proporcionar una representación diferente a la suma y resta. Esto implica que en el decodificador de 7 segmentos se deben restringir tres combinaciones para realizar esta representación. Esto se abordará dejando las letras necesarias utilizando diferentes codificadores. La elección de la operación necesaria se realiza mediante un multiplexor. Si configuramos el multiplexor en 01, se realizará la suma de las entradas; si configuramos en 10, obtendremos la resta. Por último, el bit más significativo es vital para la comparación. Si configuramos el multiplexor en $x1$, se generará la comparación, teniendo como resultado la figura 1.

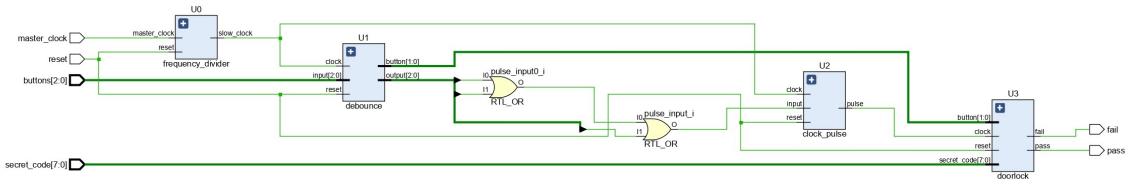

Figura 1: Esquematico del sistema

3. Experimentos

Las salidas del diseño se dejan en el apéndice A, donde se pueden analizar las variables que se muestran del archivo top en la figura 2, donde se aprecian distintas señales pero solo no interesan las correspondientes a los display, ya que estos valores son las representaciones de la salida, o los valores que deberíamos tener, debido a que el selector no está dentro de ninguno de los casos donde se mande la salida de ninguna de las operaciones podemos parecia en la figura 3. que todos los segmentos están en alto, es decir apagados, además podemos apreciar cuales son las valores de las 2 entradas correspondientes que son $a = 10$ y $b = 5$, si generamos la conversión de los valores de los display, correspondientes a 12 para *display0* y 79 para el *display1* podemos verificar que tenemos 1 y 5 dando como resultado la suma de las dos entradas en codigo BCD, ademas podemos notar que el *display2* está apagado, de esta manera puedes verificar que la salida del sistema es correcta, podemos jugar con el archivo test bench para realizar toda las pruebas con las combinaciones generando las salidas correctas en los todos los display.

4. Conclusión

El proceso de crear un diseño digital utilizando VHDL implica un aprendizaje significativo. Se requiere comprender la sintaxis de VHDL, los conceptos de diseño digital y cómo implementarlos en un entorno como Vivado, creando un sistema que realiza múltiples operaciones (suma, resta y comparación), destacando la importancia del diseño modular en VHDL. Poder dividir el sistema en módulos o entidades más pequeñas facilita el diseño, la depuración y la reutilización de código y debido a que no tenemos acceso a la tarjeta la simulación es una parte esencial del proceso de diseño en VHDL, aunque también se debe tomar en cuenta que antes de implementar el diseño en hardware, es crucial realizar una simulación exhaustiva para verificar su funcionalidad y detectar posibles errores, además la implementación en hardware a través de herramientas como Vivado implica la síntesis y optimización del diseño. Esto se traduce en la generación de un circuito físico a partir del código VHDL con la optimización de recursos como área y velocidad, por último la transición de un diseño físico a un diseño digital implica un cambio

de paradigma importante, que nos permite una mayor flexibilidad y la posibilidad de realizar cambios más rápidamente en el diseño sin modificar hardware físico.

Apéndice A: Imagenes de la simulación

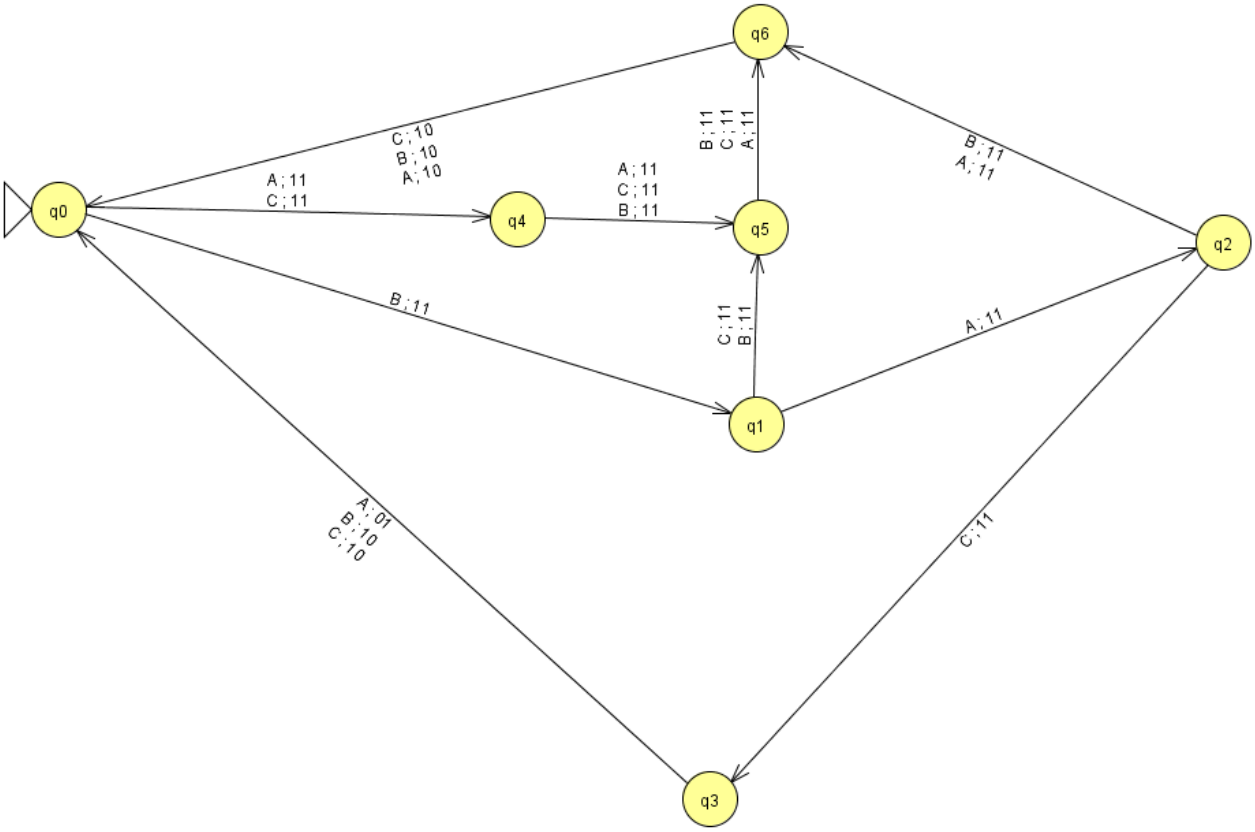

Figura 2: Variables y señales del archivo top

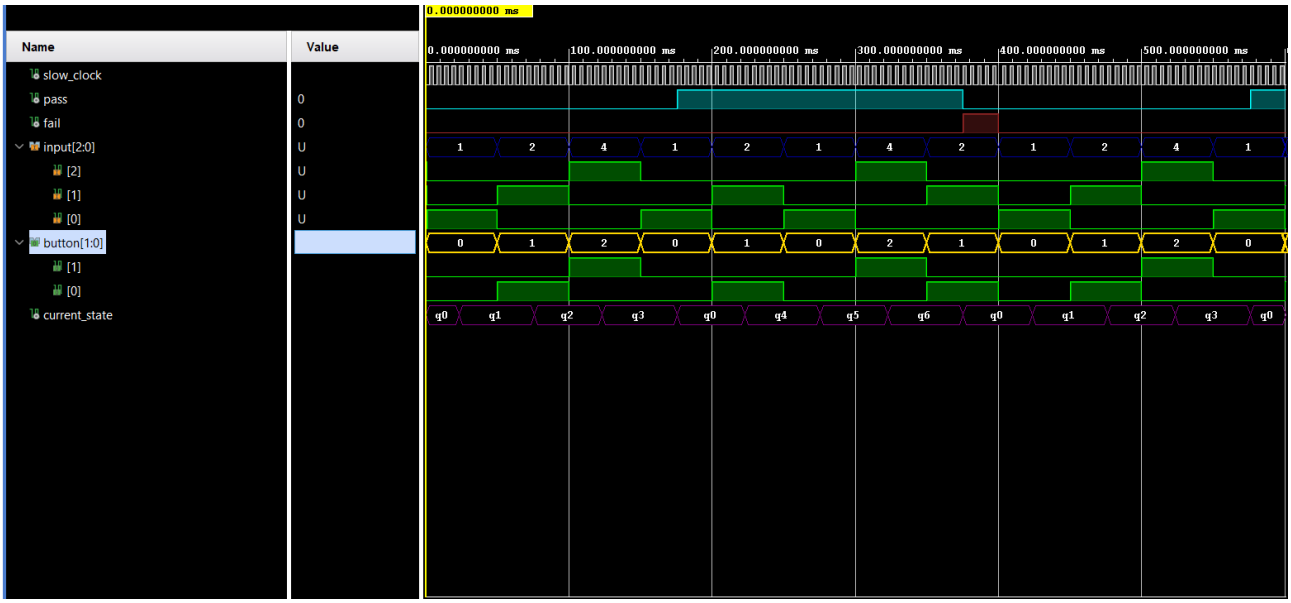

Figura 3: Señales de salida

Apéndice B: Codigo VHDL

Código del módulo con pulsos de reloj verificados

```
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;
```