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.

6 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.

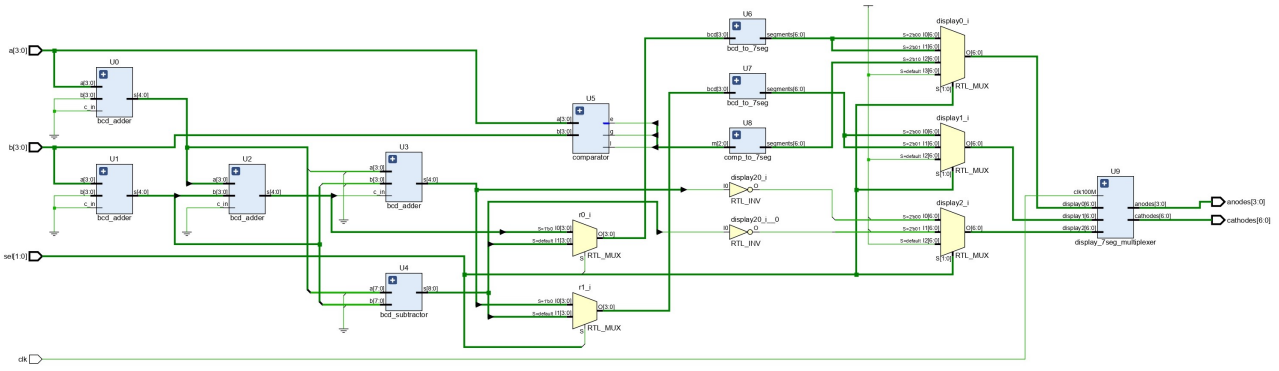


Figura 1: Esquematico del sistema

3. Experimentos

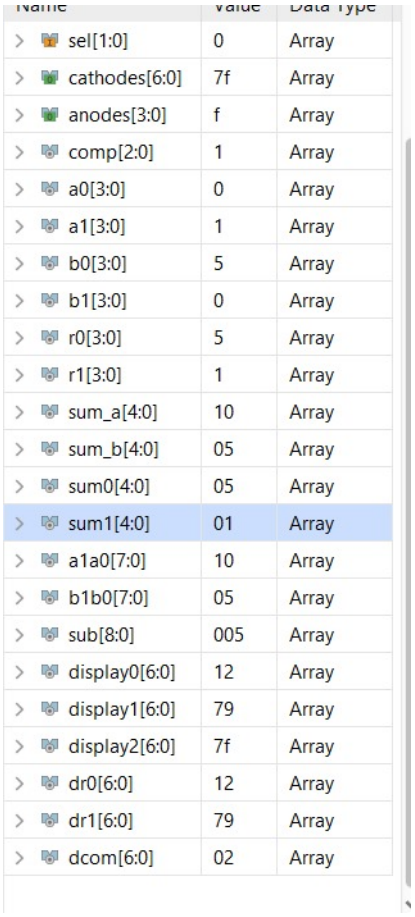
Las salidas del diseño se dejan en el apéndice A, donde se pueden analizar las variables que se muestran del archivo top en la figura 2, donde se aprecian distintas señales pero solo no interesan las correspondientes a los display, ya que estos valores son las representaciones de la salida, o los valores 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



Name	Value	Data Type
sel[1:0]	0	Array
cathodes[6:0]	7f	Array
anodes[3:0]	f	Array
comp[2:0]	1	Array
a0[3:0]	0	Array
a1[3:0]	1	Array
b0[3:0]	5	Array
b1[3:0]	0	Array
r0[3:0]	5	Array
r1[3:0]	1	Array
sum_a[4:0]	10	Array
sum_b[4:0]	05	Array
sum0[4:0]	05	Array
sum1[4:0]	01	Array
a1a0[7:0]	10	Array
b1b0[7:0]	05	Array
sub[8:0]	005	Array
display0[6:0]	12	Array
display1[6:0]	79	Array
display2[6:0]	7f	Array
dr0[6:0]	12	Array
dr1[6:0]	79	Array
dcom[6:0]	02	Array

Figura 2: Variables y señales del archivo top

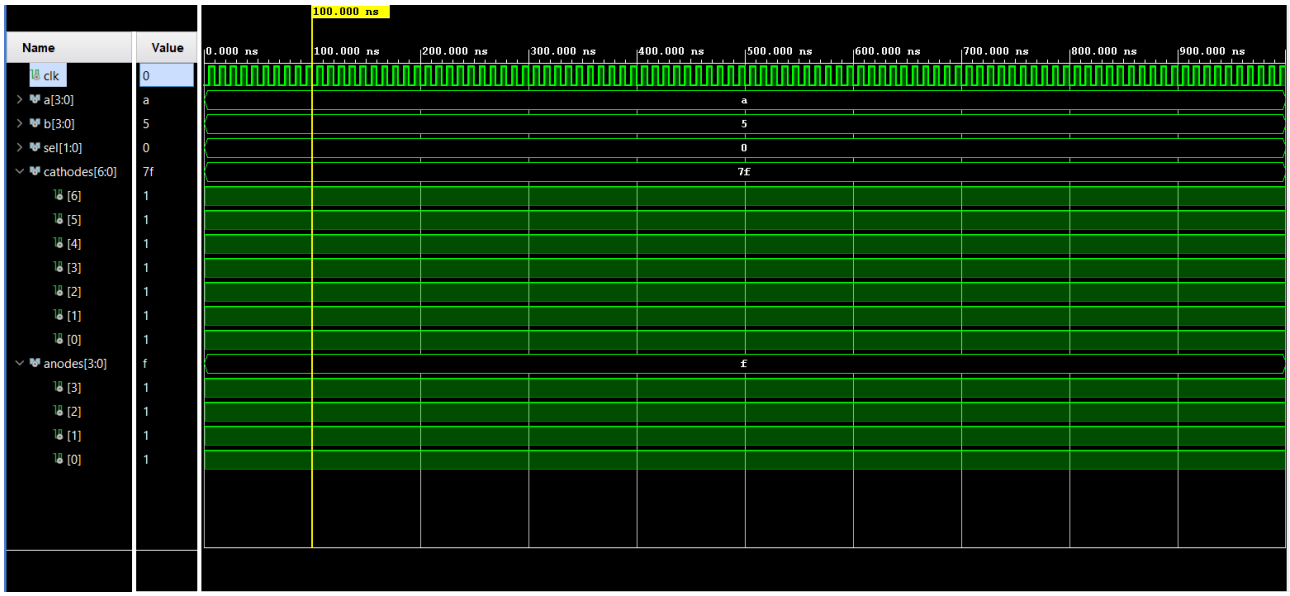


Figura 3: Señales de salida

Apéndice B: Codigo VHDL

Código del módulo sumador

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

entity bcd_adder is
    port(
        a, b : in std_logic_vector (3 downto 0);
        c_in : in std_logic;
        s : out std_logic_vector (4 downto 0)
    );
end bcd_adder;

architecture Behavioral of bcd_adder is
    signal sum : std_logic_vector (4 downto 0);
begin
    sum <= ('0' & a) + ('0' & b) + ("0000" & c_in);
    s <= sum when sum < 10 else sum + 6;
end Behavioral;
```

Código del módulo restador

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

entity bcd_subtractor is
    port(
```

```
            a, b : in std_logic_vector(7 downto 0);
            s : out std_logic_vector (8 downto 0)
        );
    end bcd_subtractor;

    architecture Behavioral of bcd_subtractor is
        signal sub, sub_aux: std_logic_vector(8 downto 0);
        signal s0, s1 : std_logic_vector(3 downto 0);
    begin
        sub <= ('0' & a) - ('0' & b);
        sub_aux <= sub when sub(8) = '0' else ('0' & not(sub
           (7 downto 0))) + 1;
        s0 <= sub_aux(3 downto 0);
        s1 <= sub_aux(7 downto 4);
        s(8) <= sub(8);
        s(3 downto 0) <= s0 when s0 < 10 else s0 + 10;
        s(7 downto 4) <= s1 when s1 < 10 else s1 + 10;
    end Behavioral;
```

Código del módulo comparador

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

    entity comparator is
        port(
            a, b : in std_logic_vector (3 downto 0);
            l, e, g : out std_logic
        );
    end comparator;

    architecture Behavioral of comparator is

    begin
        l <= '1' when a < b else '0';
        e <= '1' when a = b else '0';
        g <= '1' when a > b else '0';
    end Behavioral;
```

Código del módulo codificador BCD a 7 segmentos

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

    entity bcd_to_7seg is
        port(
            bcd : in std_logic_vector (3 downto 0);
            segments : out std_logic_vector (6 downto 0) -- G
               F E D C B A
```

```
        );
    end bcd_to_7seg;

    architecture Behavioral of bcd_to_7seg is

    begin
        with bcd select
            segments <= "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",
                        "0010000" when "1001",
                        "1111111" when others;
    end Behavioral;
```

Código del módulo codificador del comparador a 7 segmentos

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

    entity comp_to_7seg is
        port(
            m : in std_logic_vector (2 downto 0);          -- L
                E G
            segments : out std_logic_vector (6 downto 0) -- G
                F E D C B A
        );
    end comp_to_7seg;

    architecture Behavioral of comp_to_7seg is

    begin
        with m select
        segments <= "1000111" when "100",
                    "0000110" when "010",
                    "0000010" when "001",
                    "1111111" when others;
    end Behavioral;
```

Código del módulo para desplegar en el 7 segmentos

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

```
4
5           entity display_7seg_multiplexer is
6               port(
7                   clk100M : in std_logic;
8                   display0, display1, display2 : in
                       std_logic_vector (6 downto 0);
9                   anodes : out std_logic_vector (3 downto 0);
10                  cathodes : out std_logic_vector (6 downto 0)
11              );
12          end display_7seg_multiplexer;
13
14          architecture Behavioral of display_7seg_multiplexer is
15              signal counter : integer range 0 to 500000;
16              signal selector : std_logic_vector (1 downto 0) := "
                   00";
17          begin
18          clk50Hz:
19              process(clk100M)
20              begin
21                  if (rising_edge(clk100M)) then
22                      if (counter < 500000) then
23                          counter <= counter + 1;
24                      else
25                          selector <= selector + 1;
26                          counter <= 0;
27                      end if;
28                  end if;
29              end process;
30
31              with selector select
32                  anodes <= "1011" when "01",
33                            "1101" when "10",
34                            "1110" when "11",
35                            "1111" when others;
36              with selector select
37                  cathodes <= display2  when "01",
38                              display1  when "10",
39                              display0  when "11",
40                              "1111111" when others;
41          end Behavioral;
```

Código top conjuntador de módulos

```
1           library IEEE;
2           use IEEE.STD_LOGIC_1164.ALL;
3
4           entity arithmetic_system is
5               port(
6                   clk : in std_logic;
```

```
7             a, b : in std_logic_vector (3 downto 0);
8             sel : in std_logic_vector (1 downto 0);
9             cathodes : out std_logic_vector (6 downto 0);
10            anodes : out std_logic_vector (3 downto 0)
11        );
12    end arithmetic_system;
13
14    architecture Behavioral of arithmetic_system is
15        signal comp : std_logic_vector (2 downto 0);
16        signal a0, a1, b0, b1, r0, r1: std_logic_vector (3
              downto 0);
17        signal sum_a, sum_b, sum0, sum1: std_logic_vector (4
              downto 0);
18        signal a1a0, b1b0 : std_logic_vector (7 downto 0);
19        signal sub : std_logic_vector (8 downto 0);
20        signal display0, display1, display2, dr0, dr1, dcom:
              std_logic_vector (6 downto 0);
21    begin
22    U0: entity work.bcd_adder port map(a => a, b => "0000",
          c_in => '0', s => sum_a);
23    U1: entity work.bcd_adder port map(a => b, b => "0000",
          c_in => '0', s => sum_b);
24        a0 <= sum_a(3 downto 0);
25        a1 <= "000" & sum_a(4);
26        b0 <= sum_b(3 downto 0);
27        b1 <= "000" & sum_b(4);
28    U2: entity work.bcd_adder port map(a => a0, b => b0, c_in
          => '0', s => sum0);
29    U3: entity work.bcd_adder port map(a => a1, b => b1, c_in
          => sum0(4), s => sum1);
30        a1a0 <= a1 & a0;
31        b1b0 <= b1 & b0;
32    U4: entity work.bcd_subtractor port map(a => a1a0, b =>
          b1b0, s => sub);
33    U5: entity work.comparator port map(a => a, b => b, l =>
          comp(2), e => comp(1), g => comp(0));
34        r0 <= sum0(3 downto 0) when sel(0) = '0' else sub(3
              downto 0);
35        r1 <= sum1(3 downto 0) when sel(0) = '0' else sub(7
              downto 4);
36    U6: entity work.bcd_to_7seg port map(bcd => r0, segments
          => dr0);
37    U7: entity work.bcd_to_7seg port map(bcd => r1, segments
          => dr1);
38    U8: entity work.comp_to_7seg port map(m => comp, segments
          => dcom);
39        with sel select
40            display0 <= dr0 when "00",
```

```
41                              dr0 when "01",
42                              dcom when "10",
43                              "1111111" when others;
44              with sel select
45                  display1 <= dr1 when "00",
46                              dr1 when "01",
47                              "1111111" when others;
48              with sel select
49                  display2 <= not(sum1(4)) & "111111" when "00",
50                              not(sub(8)) & "111111" when "01",
51                              "1111111" when others;
52          U9: entity work.display_7seg_multiplexer port map(clk100M
                => clk, display0 => display0, display1 => display1,
53              display2 => display2, anodes => anodes, cathodes =>
                   cathodes);
54          end Behavioral;
```