Diseño y Simulación de una Unidad Aritmética para la Suma, Resta y Comparación de dos Números Binarios de cuatro bits Utilizando un Lenguaje de Descripción de Hardware

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

1. Introducción

El desarrollo de Sistemas Digitales es un proceso largo y complicado que requiere de dedicación al momento de crear los distintos módulos que conformarán el sistema, si no se tiene cuidado esta tarea puede volverse un problema al momento de la implementación, ya que trabajar manipulando circuitos complejos en los cuales se desconocen las posibles fallas debidas a un mal diseño significa una perdida de tiempo y un gasto innecesario. Las pruebas reales en esta clase de sistemas también exige un gran gasto y esfuerzo en su construcción y acarrean el problema de no garantizar un correcto resultado sumado a las posibles fallas en la implementación debidas a malas conexiones o por elementos faltantes. Con el fin de reducir fallos, el proceso de diseño y comprobación de circuitos digitales ha implementado herramientas para facilitar el diseño como los PLD´s, CPLD´s o FPGAS por mencionar algunos, con las cuales podemos diseñar diferentes circuitos a partir de un lenguaje de descripción de hardware el cual como su nombre indica, podemos realizar el diseño de un circuito digital al describir su funcionamiento. Con el uso de estas herramientas el diseño y comprobación de un sistema se vuelve mucho más sencillo de analizar, corregir y simular. Para comprobar la funcionalidad de estas se describirá el proceso de diseño para una unidad aritmética usando el lenguaje de descripción de hardware VHDL y el entorno de desarrollo Vivado. Usando estas herramientas se diseñará una unidad aritmética empleando la FPGA(Arreglo de compuertas programables en campo por sus siglas en inglés) Artix 7 integrada en la tarjeta Basys 3 de Digilent.

2. Descripción del Método

Para el desarrollo de la unidad se usó el modelado de arriba abajo, Partiendo de una idea general, pasando por cada una de las funciones que componen al sistema. Una vez diseñado y comprobado cada módulo procederemos a unirlos en el conjunto total que desempeñará la tarea que se busca satisfacer.
En esta ocasión se busca diseñar un sistema aritmético para dos números binarios de cuatro bits en el que se incluyan las operaciones básicas de suma resta y comparación. por lo que se debe partir de estos módulos para comenzar el diseño. Además de incluir sus respectivos codificadores para la salida y multiplexores que realizaran la selección de las operaciones que se deseen visualizar. En la figura 1 se puede apreciar el diagrama detallado de la solución:
Partimos entonces por el diseño de las unidades aritméticas: En el caso del sumador se requieren dos vectores de entrada A Y B que se deben sumar. En VHDL es posible realizar la operación empleando los operadores aritméticos. Como es posible obtener salidas con carry se debe extender la cantidad de bits a la salida, por lo que se añaden tres ceros al resultado de salida usando el operador de concatenación: &. Esto nos entregará la suma de ambos números en la salida. Para evaluar que esta sea una salida BCD válida se debe primero evaluar que la salida no sea menor a nueve, en caso contrario se le debe sumar seis.
El módulo restador es muy parecido a un sumador pero con complementos, se debe primero obtener el complemento de B invirtiendo su salida y sumándole un uno[1]. Para evaluar si el resultado es un número negativo se debe evaluar el primer bit más significativo, si este es uno se debe obtener su complemento para hacer la corrección y obtener un valor BCD válido además de evaluar el signo. En la salida se incluye un código para determinar si la salida tiene o no signo.

Para el caso del comparador usamos una estructura secuencial y los operadores de comparación con los que podemos obtener los resultados rápidamente.

Una vez terminados los sistemas de aritméticos pasamos a crear los decodificadores con los que podremos desplegar los valores binarios en un display de siete segmentos en la salida. Para los decodificadores podemos usar asignación condicionada para obtener los valores exactos en su determinado caso. Para el decodificador del comparador se realiza el mismo proceso pero cuidando que en la salida se obtengan las salidas correspondientes para observar las letras según cada caso posible de la comparación(E en caso de ser iguales, G si A es mayor que B y L si A es menor). Una vez detallados los módulos podemos unirlos con multiplexores en los que se puede seleccionar la salida que se desee observar, con estos podemos intercambiar entre ver el los números ingresados, el resultado de la suma o resta y la comparación. Por último se añade un módulo con el que se despliegan los valores hacia el display. Dado que la FPGA solo permite mostrar un valor a la vez es necesario crear un sistema secuencial que muestre cada dígito en una fracción de tiempo para lograr que se observen todos los valores al mismo tiempo, para ello se usó un divisor de frecuencia y multiplexores que harán el trabajo descrito.

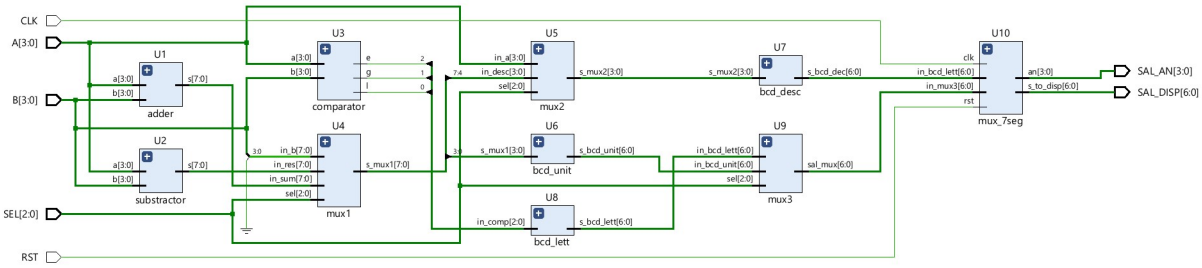


Figura 1: Diagrama general del sistema

3. Experimentos

Para la comprobación del sistema se trabajó y desarrolló el proyecto totalmente en la herramienta de Vivado la cual ofrece los elementos suficientes para trabajar con el lenguaje de VHDL además de incorporar su herramienta de simulación con la que se puede observar los diagramas de tiempo del sistema. Con esto es posible observar detalladamente el comportamiento del sistema, por lo que probamos la simulación en cada elemento antes de unirlos y solo hasta verificar que todos funcionaran adecuadamente es que se unieron. Las gráficas de las comprobaciones se encuentran en el apéndice C.

4. Conclusiones

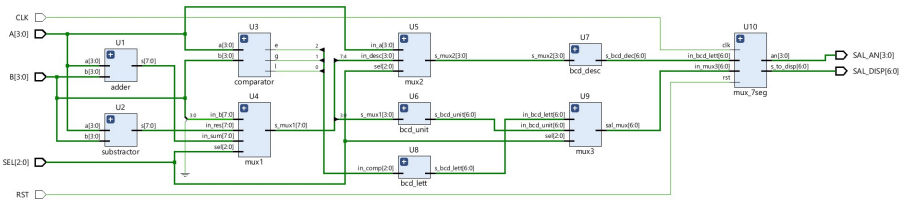
Las herramientas actuales facilitan de gran manera el proceso de diseño, con los problemas actuales crear elementos digitales se ha vuelto una tarea complicada que con el uso de esta clase de herramientas es más fácil el proceso de diseño. Con estas herramientas es posible diseñar y simular cualquier modulo de forma cómoda, por sobre la construcción del sistema en tabletas y utilizar circuitos físicos de difícil manipulación

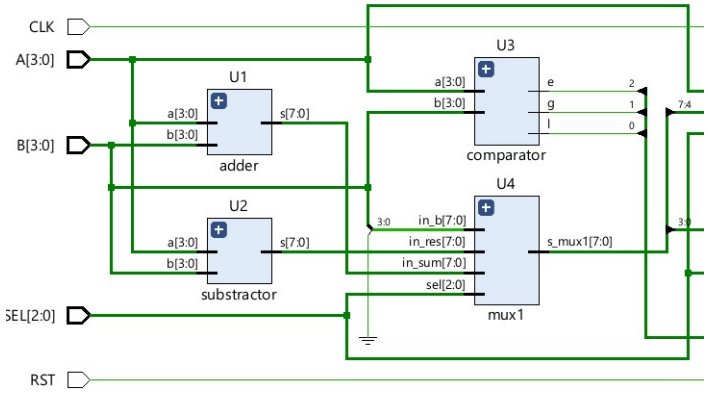
Referencias

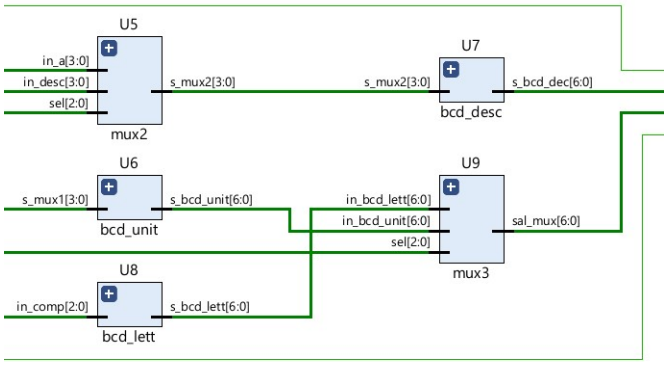
M. M. Mano, *Digital Design*, 2a ed. Englewood Cliffs, N.J: Prentice-Hall, 1991.

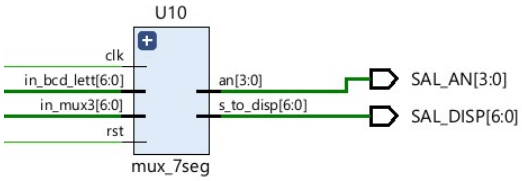
5. Apéndice A

Diagrama detallado de la solución









6. Apéndice B

Código en VHDL:
Módulo arithmetic_unit_top:

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

entity arithmetic_unit_top is
        port(   A: in std_logic_vector (3 downto 0);
                        B: in std_logic_vector (3 downto 0);
                        CLK: in std_logic;
                        RST : in std_logic;
                        SEL: in std_logic_vector (2 downto 0);
                        SAL_DISP: out std_logic_vector (6 downto 0);
                        SAL_AN: out std_logic_vector (3 downto 0));
end arithmetic_unit_top;

architecture Behavioral of arithmetic_unit_top is
-- Seales  para conexion interna:
signal s_sum : std_logic_vector (7 downto 0);
signal s_res : std_logic_vector (7 downto 0);
signal s_comp : std_logic_vector (2 downto 0);
signal fix_b : std_logic_vector (7 downto 0);
signal sal_mux1 : std_logic_vector (7 downto 0);
signal sal_mux2 : std_logic_vector (3 downto 0);
signal sal_bcd_unit : std_logic_vector (6 downto 0);
signal sal_bcd_dec : std_logic_vector (6 downto 0);
signal sal_bcd_lett : std_logic_vector (6 downto 0);
signal sal_mux3 : std_logic_vector (6 downto 0);


begin
fix_b <= "0000"&B;

U1: entity work.adder port map(
        a => A,
        b => B,
        s => s_sum
);

U2: entity work.substractor port map(
        a => A,
        b => B,
        s => s_res
);

U3: entity work.comparator port map(
        a => A,
        b => B,
        e => s_comp(2),
```

```
        g => s_comp(1),
        l => s_comp(0)
);

U4: entity work.mux1 port map(
        in_sum => s_sum,
        in_res => s_res,
        in_b => fix_b,
        sel => SEL,
        s_mux1 => sal_mux1
);

U5: entity work.mux2 port map(
        in_desc(0) => sal_mux1(4),
        in_desc(1) => sal_mux1(5),
        in_desc(2) => sal_mux1(6),
        in_desc(3) => sal_mux1(7),
        in_a => A,
        sel => SEL,
        s_mux2 => sal_mux2
);

U6: entity work.bcd_unit port map(
        s_mux1(0) => sal_mux1(0),
        s_mux1(1) => sal_mux1(1),
        s_mux1(2) => sal_mux1(2),
        s_mux1(3) => sal_mux1(3),
        s_bcd_unit => sal_bcd_unit
);

U7: entity work.bcd_desc port map(
        s_mux2 => sal_mux2,
        s_bcd_dec => sal_bcd_dec
);

U8: entity work.bcd_lett port map(
        in_comp => s_comp,
        s_bcd_lett => sal_bcd_lett
);

U9: entity work.mux3 port map(
        in_bcd_unit => sal_bcd_unit,
        in_bcd_lett => sal_bcd_lett,
        sel => SEL,
        sal_mux => sal_mux3
);

U10: entity work.mux_7seg port map(
        clk => CLK,
        rst => RST,
        in_mux3 => sal_mux3,
```

```
        in_bcd_lett => sal_bcd_dec,
        s_to_disp => SAL_DISP,
        an => SAL_AN
);
end Behavioral;
```

Módulo sumador:

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

entity adder is
        Port (  a: in std_logic_vector (3 downto 0);
                b: in std_logic_vector (3 downto 0);
                s: out std_logic_vector (7 downto 0));
end adder;

architecture Behavioral of adder is
signal temp: std_logic_vector(4 downto 0);
signal fix: std_logic;
signal complete: std_logic_vector (7 downto 0);

begin

temp <= ('0'&a)+b;
fix <= '1' when (temp > 9) or (temp(4) = '1') else '0';
complete <= ("000"&temp) when (fix = '0') else ("000"&(temp+6));
s <= complete;

end Behavioral;
```

Módulo Restador:

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

entity substractor is
        port(   a: in std_logic_vector (3 downto 0);
                b: in std_logic_vector (3 downto 0);
                s: out std_logic_vector (7 downto 0));
end substractor;

architecture Behavioral of substractor is
signal resta: std_logic_vector (4 downto 0);
signal comp_b: std_logic_vector (3 downto 0);
signal brw : std_logic;
signal temp : std_logic_vector (4 downto 0);
signal complete : std_logic_vector (7 downto 0);

begin
        comp_b <= (not b)+1;
        resta <= ('0'&a)+comp_b;
```

```
        brw <= '1' when (resta(4) = '0') else '0';
        temp <= ((not resta)+1) when brw = '1' else resta;
        complete <= '1'&brw&'1'&temp;
        s <= complete;

end Behavioral;
```

Módulo comparador:

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


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

architecture Behavioral of comparator is
begin
    process(a,b)
        begin
        if (a = b) then
            e <= '1';
                    g <= '0';
                    l <= '0';
        elsif (a > b) then
            e <= '0';
                    g <= '1';
                    l <= '0';
        else
            e <= '0';
                    g <= '0';
                    l <= '1';
        end if;
    end process;
end Behavioral;
```

Módulo Multiplexor hacia decodificador BCD unidades y descenas:

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

entity mux1 is
    port(   in_sum : in std_logic_vector(7 downto 0);
            in_res : in std_logic_vector (7 downto 0);
            in_b : in std_logic_vector (7 downto 0);
            sel : in std_logic_vector (2 downto 0);
```

```
            s_mux1 : out std_logic_vector (7 downto 0));
end mux1;

architecture Behavioral of mux1 is

begin
    with (sel) select
        s_mux1 <=   in_b when "000",
                    in_sum when "001",
                    in_res when "010",
                    "00000000" when others;
end Behavioral;
```

Módulo Multiplexor hacia decodificador decenas:

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

entity mux2 is
    port(   in_desc : in std_logic_vector (3 downto 0);
            in_a : in std_logic_vector (3 downto 0);
            sel : in std_logic_vector (2 downto 0);
            s_mux2 : out std_logic_vector(3 downto 0));
end mux2;

architecture Behavioral of mux2 is

begin
    with (sel) select
        s_mux2 <=   in_a when "000",
                    in_desc when "001",
                    in_desc when "010",
                    "0000" when others;
end Behavioral;
```

Módulo Decodificador BCD para las unidades:

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

entity bcd_unit is
    port(   s_mux1 : in std_logic_vector (3 downto 0);
            s_bcd_unit : out std_logic_vector (6 downto 0));
end bcd_unit;

architecture Behavioral of bcd_unit is

begin
    with s_mux1 select
        s_bcd_unit <=   "0000001" when "0000",
                        "1001111" when "0001",
                        "0010010" when "0010",
```

```
                        "0000110" when "0011",
                        "1001100" when "0100",
                        "0100100" when "0101",
                        "0100000" when "0110",
                        "0001111" when "0111",
                        "0000000" when "1000",
                        "0000100" when "1001",
                        "1111111" when others;

end Behavioral;
```

Módulo Decodificador BCD para las decenas:

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

entity bcd_desc is
    port(   s_mux2 : in std_logic_vector (3 downto 0);
            s_bcd_dec : out std_logic_vector (6 downto 0));
end bcd_desc;

architecture Behavioral of bcd_desc is

begin
    with s_mux2 select
        s_bcd_dec <=    "0000001" when "0000",
                        "1001111" when "0001",
                        "0010010" when "0010",
                        "0000110" when "0011",
                        "1001100" when "0100",
                        "0100100" when "0101",
                        "0100000" when "0110",
                        "0001111" when "0111",
                        "0000000" when "1000",
                        "0000100" when "1001",
                        "0000001" when "1011",
                        "1111110" when "1111",
                        "1111111" when others;

end Behavioral;
```

Módulo Decodificador de letras:

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

entity bcd_lett is
    port(   in_comp : in std_logic_vector (2 downto 0);
            s_bcd_lett : out std_logic_vector (6 downto 0));
end bcd_lett;

architecture Behavioral of bcd_lett is
```

```
begin
    with (in_comp) select
        s_bcd_lett <=   "0110000" when "100",
                        "0100000" when "010",
                        "1110001" when "001",
                        "1111111" when others;
end Behavioral;
```

Módulo Multiplexor para números y letras:

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

entity mux3 is
    port(   in_bcd_unit : in std_logic_vector (6 downto 0);
            in_bcd_lett : in std_logic_vector (6 downto 0);
            sel : in std_logic_vector (2 downto 0);
            sal_mux : out std_logic_vector (6 downto 0));
end mux3;

architecture Behavioral of mux3 is

begin
    with (sel) select
        sal_mux <=  in_bcd_unit when "000",
                    in_bcd_unit when "001",
                    in_bcd_unit when "010",
                    in_bcd_lett when "100",
                    "1111111" when others;
end Behavioral;
```

Módulo Multiplexor para el display de siete segmentos:

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

entity mux_7seg is
    port(   clk : in std_logic;
            rst : in std_logic;
            in_mux3 : in std_logic_vector (6 downto 0);
            in_bcd_lett : in std_logic_vector (6 downto 0);
            s_to_disp : out std_logic_vector (6 downto 0);
            an : out std_logic_vector (3 downto 0));
end mux_7seg;

architecture Behavioral of mux_7seg is
signal clk190 : std_logic := '0';
signal counter1 : std_logic_vector (17 downto 0);

begin
    --Divisor de frecuencia
```

```
    clock: process(clk,rst)
        begin
        if (rst = '1') then
            counter1 <= (others => '0');
        elsif (clk'event and clk = '1') then
            counter1 <= counter1+1;
            if (counter1(17) = '1') then
                clk190 <= not clk190;
                counter1 <= (others => '0');
            end if;
        end if;
    end process clock;

    with (clk190) select
        s_to_disp <=    in_mux3 when '0',
                        in_bcd_lett when '1',
                        "1111111" when others;

    with (clk190) select
        an <=   "1110" when '0',
                "1101" when '1',
                "1111" when others;

end Behavioral;
```

7. Apéndice C

Resultados de la simulación

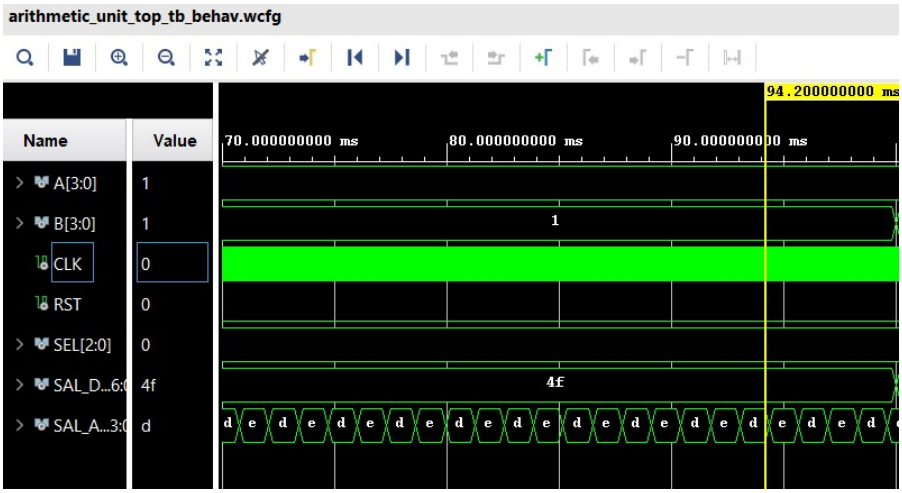


Figura 2: Despliegue de los números ingresados

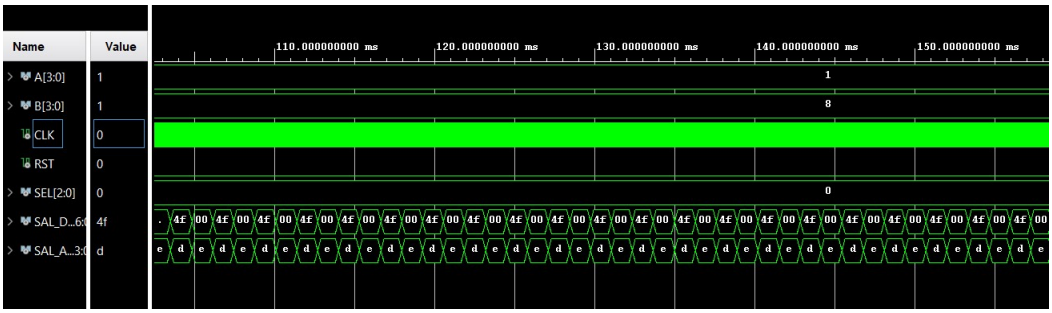


Figura 3: Despliegue de los números ingresados, ambos diferentes

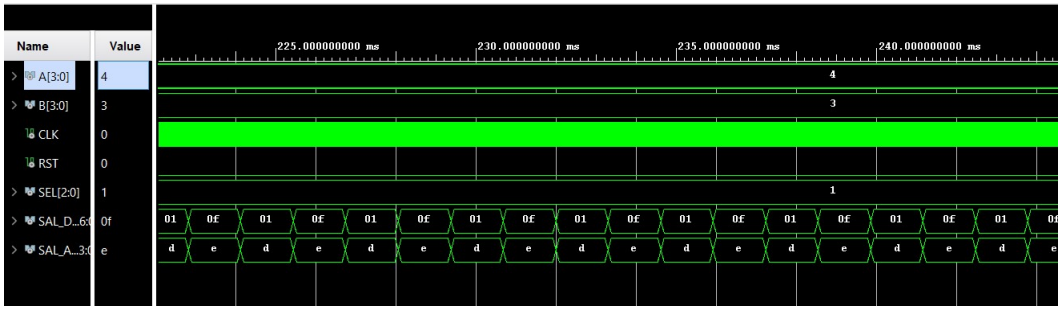


Figura 4: Resultado de la suma

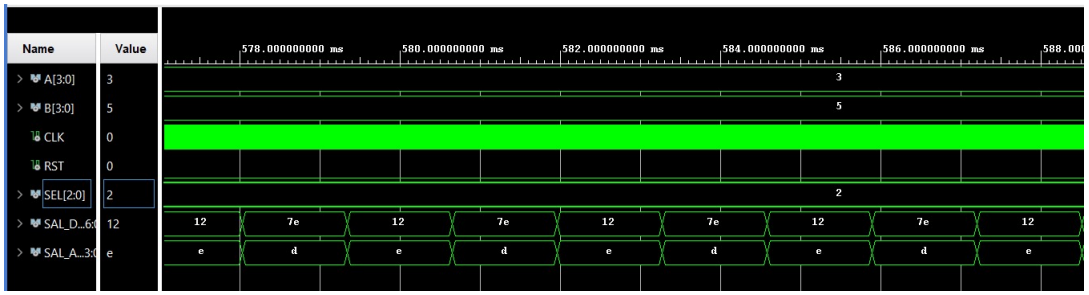


Figura 5: Resultado de la resta

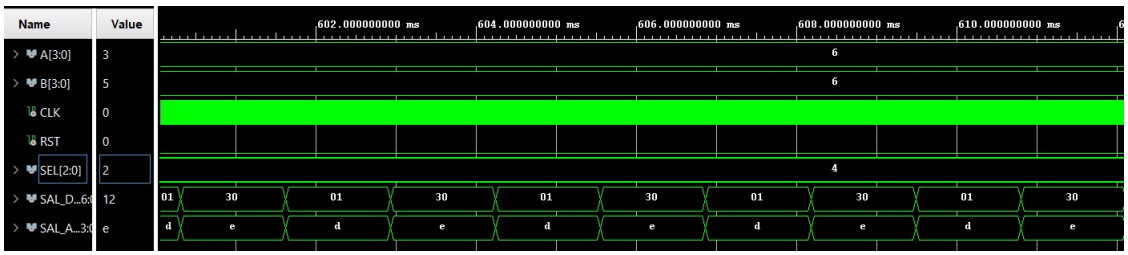


Figura 6: Resultado de la comparación con dos números iguales

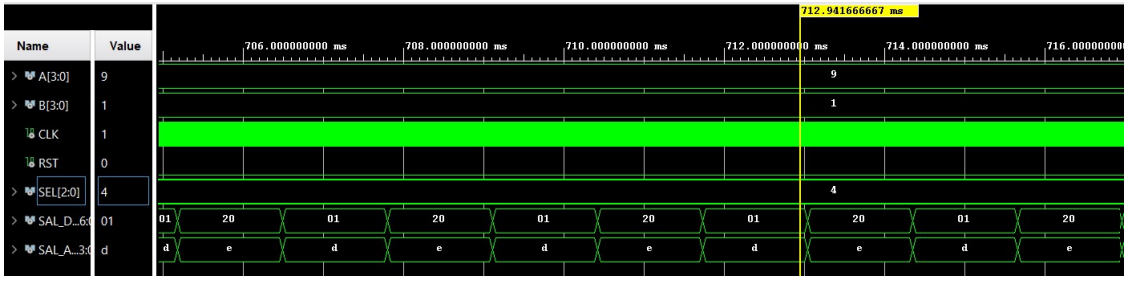


Figura 7: Resultado de la comparación con A mayor que B

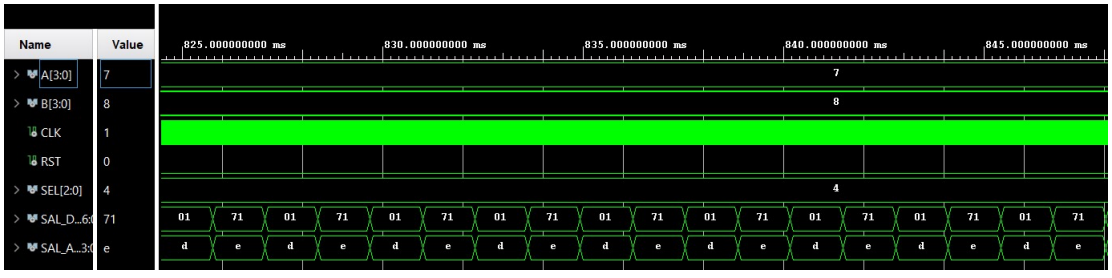


Figura 8: Resultado de la comparación con A menor que B