Diseño y simulación de una unidad de control para el cálculo de la función factorial de cualquier número en el intervalo de cero a siete utilizando un lenguaje de descripción.

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

1. Introducción

Los sistemas digitales se componen de dos ramas principales en las que se tienen el uso de circuitos combinacionales los cuales entregan un resultado a partir de una entrada dada y realizando una serie de combinaciones para obtener el resultado esperado, por otra parte se encuentran los sistemas secuenciales que cuentan con la capacidad de almacenar datos y entregar diferentes resultados según las entradas que reciba a través del tiempo. De cada grupo se pueden crear sistemas únicos usando elementos puramente combinacionales o secuenciales, existe también la posibilidad de combinar ambos para obtener sistemas más potentes compensando las deficiencias de cada grupo. Una unidad de control(Control Unit) es un sistema secuencial que coordina el flujo de datos a través de la unidad, generalmente está acompañado de elementos combinacionales para realizar operaciones, a este conjunto se le conoce como ruta de datos(Data Path). Con el fin de comprobar la funcionalidad de las unidades de control es que se a elaborado el diseño de un sistema que en función de una entrada n y un indicador se obtendrá el valor de la función factorial del valor dado. A continuación se presenta el diseño y los pasos para la elaboración del sistema.

2. Descripción del método

Para la solución se propone seguir la metodología de arriba hacia abajo, de modo que podemos partir de una idea general para posteriormente diseñar cada una de las partes que componen al sistema. El factorial de un número es una función recursiva que consiste en realizar la operación $n! = n((n-1)!)$ en términos simples una función recursiva no es más que un proceso iterativo por lo que es necesario utilizar un registro acumulador para guardar el producto de los términos, por lo que se necesitarán registros para acumular el resultado y las iteraciones necesarias además de sistemas aritméticos de resta y multiplicación necesarios para realizar las operaciones. Una vez descrito el problema podemos modelar el diagrama de la solución:

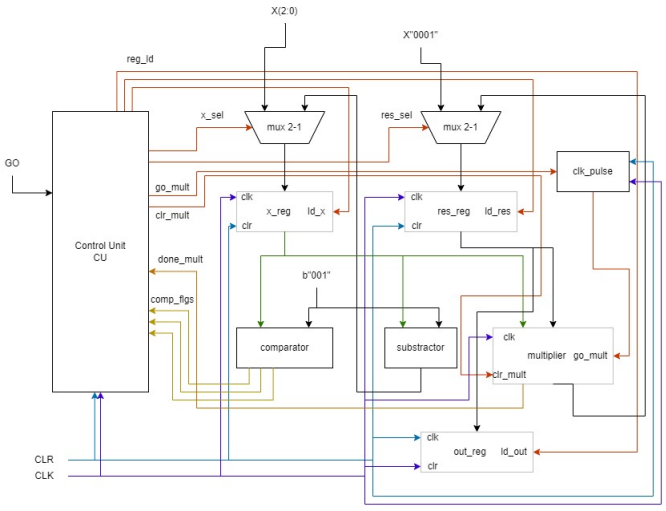


Figura 1: Diagrama general de la solución

Por medio de los multiplexores seleccionaremos la entrada hacia los registros donde se guardaran los valores iniciales en primera instancia, en el registro X se almacena el número del que se desea obtener el factorial, además de funcionar como el registro que guiará la cantidad de iteraciones necesarias para cumplir la tarea, la unidad de comparador determinará el momento en que finalice el procedimiento mientras que el módulo restador irá restando una unidad al registro X de modo que se pueda continuar con el procedimiento hasta que se cumpla la condición $X <= 1$. El registro resultado funciona como acumulador donde se irán guardando los resultados de las operaciones $n(n-1)$ que será la salida del módulo de multiplicación, en cuanto termine la operación se enviará la señal hacia el registro de salida (out_reg) el cual será el que entregue el resultado final. Para guiar la operación es que se utiliza la unidad de control la cual recibe señales generadas por la ruta de datos y envía algunas para indicar en que momento se deben realizar cada uno de los pasos a la ruta de datos. La unidad de control es en sí un autómata el cual según el estado en que se encuentre entrega una determinada salida, para el diseño de la unidad de control se siguió el diagrama siguiente:

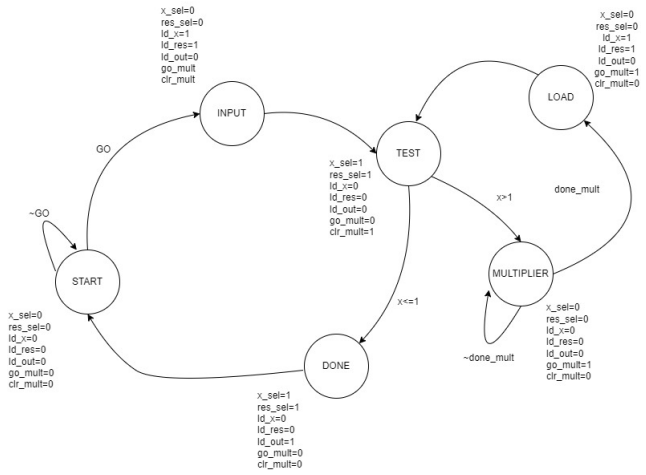


Figura 2: Diagrama de estados para el diseño de la unidad de control

3. Experimentos

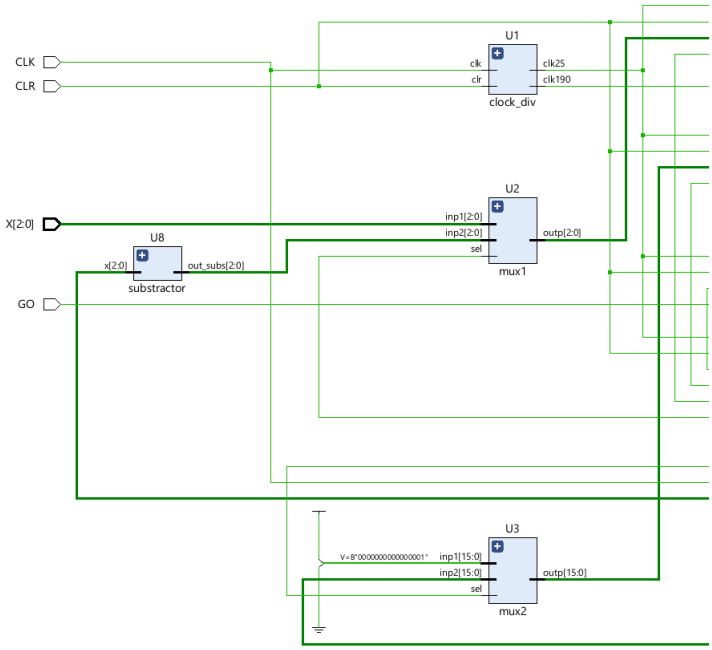
Para comprobar la funcionalidad de cada uno de los módulos se realizó una simulación de su comportamiento antes de integrarlos en el sistema final, cada módulo fue probado por separado además de realizarse la comprobación de la unión de todos los sistemas. La comprobación del funcionamiento del sistema se encuentra en el apéndice C de este trabajo.

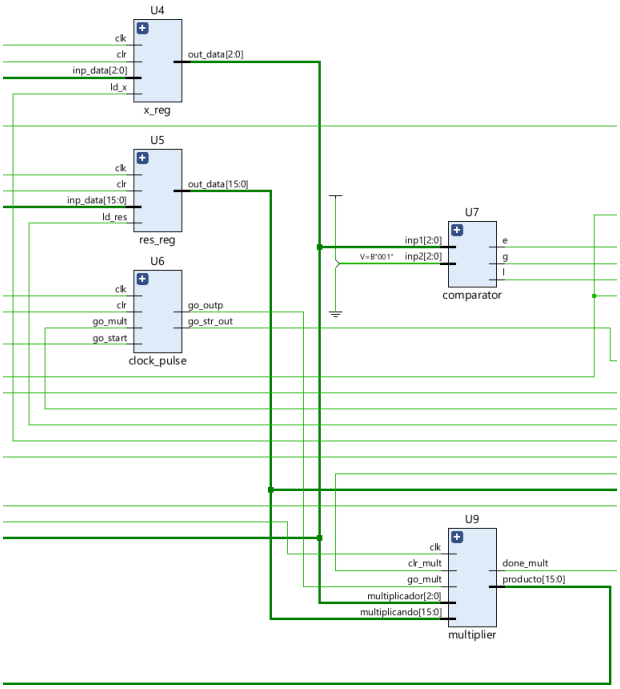
4. Conclusiones

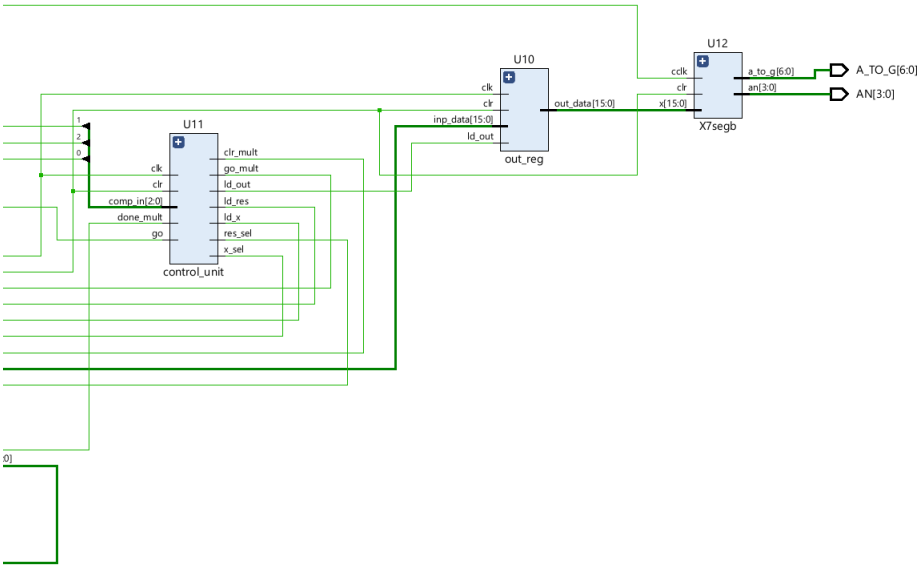
Una unidad de control es una de las partes fundamentales que componen a una unidad de procesamiento, la implementación de este tipo de sistemas permiten mantener un control en las operaciones de sistemas combinacionales por lo que se pueden realizar combinaciones complejas con la ayuda de un control que coordine la operación evitando errores en el proceso.

5. Apéndice A

Diagrama detallado de la solución:







6. Apéndice B

Código en VHDL:

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

entity factorial_algorithm_top is
    port(   CLK : in STD_LOGIC;
            CLR : in STD_LOGIC;
            GO : in std_logic;
            X : in std_logic_vector (2 downto 0);
            A_TO_G : out std_logic_vector (6 downto 0);
            AN : out std_logic_vector (3 downto 0)
    );
end factorial_algorithm_top;

architecture Behavioral of factorial_algorithm_top is
signal clk_25 : std_logic;
signal clk_190 : std_logic;
signal bin_one : std_logic_vector (2 downto 0) := "001";
signal hex_one : std_logic_vector (15 downto 0) := X"0001";
signal out_mux1 : std_logic_vector (2 downto 0);
signal out_mux2 : std_logic_vector (15 downto 0);
signal out_x_reg : std_logic_vector (2 downto 0);
signal out_result_reg : std_logic_vector (15 downto 0);
signal go_start_output : std_logic;
signal go_output : std_logic;
signal clr_multiplier_output : std_logic;
signal out_comparator : std_logic_vector (2 downto 0);
signal out_substractor : std_logic_vector (2 downto 0);
```

```
signal done_multiplier : std_logic;
signal product_multiplier : std_logic_vector (15 downto 0);
signal load_out : std_logic;
signal output_register : std_logic_vector (15 downto 0);
signal x_selector : std_logic;
signal res_selector : std_logic;
signal load_x : std_logic;
signal load_result : std_logic;
signal go_multiplier : std_logic;
signal clr_multiplier : std_logic;

begin
U1: entity work.clock_div port map(
        clk => CLK,
        clr => CLR,
        clk25 => clk_25,
        clk190 => clk_190
);

U2: entity work.mux1 port map(
        inp1 => X,
        inp2 => out_substractor,
        sel => x_selector,
        outp => out_mux1
);

U3: entity work.mux2 port map(
        inp1 => hex_one,
        inp2 => product_multiplier,
        sel => res_selector,
        outp => out_mux2
);

U4: entity work.x_reg port map(
        inp_data => out_mux1,
        clk => clk_25,
        clr => CLR,
        ld_x => load_x,
        out_data => out_x_reg
);

U5: entity work.res_reg port map(
        inp_data => out_mux2,
        clk => clk_25,
        clr => CLR,
        ld_res => load_result,
        out_data => out_result_reg
);

U6: entity work.clock_pulse port map(
        go_start => GO,
```

```
        go_mult => go_multiplier,
--        clr_mult => clr_multiplier,
        clk => clk_25,
        clr => CLR,
        go_str_out => go_start_output,
        go_outp => go_output
--        clr_m_outp => clr_multiplier_output

);

U7: entity work.comparator port map(
        inp1 => out_x_reg,
        inp2 => bin_one,
        g => out_comparator(2),
        e => out_comparator(1),
        l => out_comparator(0)
);

U8: entity work.substractor port map(
        x => out_x_reg,
        out_subs => out_substractor
);

U9: entity work.multiplier port map(
        clk => CLK,
        clr_mult => clr_multiplier,
        go_mult => go_output,
        multiplicando => out_result_reg,
        multiplicador => out_x_reg,
        done_mult => done_multiplier,
        producto => product_multiplier
);

U10: entity work.out_reg port map(
        inp_data => out_result_reg,
        clk => clk_25,
        clr => CLR,
        ld_out => load_out,
        out_data => output_register
);

U11: entity work.control_unit port map(
        go => go_start_output,
        clk => clk_25,
        clr => CLR,
        comp_in => out_comparator,
        done_mult => done_multiplier,
        x_sel => x_selector,
        res_sel => res_selector,
        ld_x => load_x,
        ld_res => load_result,
```

```
        ld_out => load_out,
        go_mult => go_multiplier,
        clr_mult => clr_multiplier
);

U12: entity work.X7segb port map(
        x => output_register,
        cclk => clk_190,
        clr => CLR,
        a_to_g => A_TO_G,
        an => AN
);

end Behavioral;
```

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

entity clock_div is
    port(   clk : in std_logic;
            clr : in std_logic;
            clk25 : out std_logic;
            clk190 : out std_logic
        );
end clock_div;

architecture Behavioral of clock_div is
signal counter : std_logic_vector (1 downto 0);
signal counter1 : std_logic_vector (17 downto 0);
signal tmp_clk : std_logic := '1';
signal tmp_clk1 : std_logic := '1';

begin
    clk_25: process (clk,clr)
    begin
        if (clr = '1') then
            counter <= "00";
            clk25 <= not tmp_clk;
        elsif (rising_edge(clk)) then
            counter <= counter+1;
            if (counter(1) = '1') then
                tmp_clk <= not tmp_clk;
                clk25 <= tmp_clk;
                counter <= "00";
            end if;
        end if;
    end process clk_25;

    clk_190: process (clk,clr)
    begin
        if (clr = '1') then
```

```
            counter1 <= (others => '0');
            clk190 <= not tmp_clk1;
        elsif (rising_edge(clk)) then
            counter1 <= counter1+1;
            if (counter1(17) = '1') then
                tmp_clk1 <= not tmp_clk1;
                clk190 <= tmp_clk1;
                counter1 <= (others => '0');
            end if;
        end if;
    end process clk_190;
end Behavioral;
```

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

entity mux1 is
    port(   inp1,inp2 : in std_logic_vector(2 downto 0);
            sel : in std_logic;
            outp : out std_logic_vector(2 downto 0));
end mux1;

architecture Behavioral of mux1 is
begin
    with (sel) select
        outp <= inp1 when '0',
                inp2 when '1',
                "000" when others;

end Behavioral;
```

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

entity mux2 is
    port(   inp1,inp2 : in std_logic_vector(15 downto 0);
            sel : in std_logic;
            outp : out std_logic_vector(15 downto 0));
end mux2;

architecture Behavioral of mux2 is
begin
    with (sel) select
        outp <= inp1 when '0',
                inp2 when '1',
                X"0000" when others;

end Behavioral;
```

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

```
entity x_reg is
    port(   inp_data : in std_logic_vector (2 downto 0);
            clk,clr,ld_x : in std_logic;
            out_data : out std_logic_vector (2 downto 0));
end x_reg;

architecture Behavioral of x_reg is

begin
    process(clk,clr,ld_x)
    begin
        if (clr = '1') then
            out_data <= (others => '0');
        elsif (rising_edge(clk)) then
            if(ld_x = '1') then
                out_data <= inp_data;
            end if;
        end if;
    end process;

end Behavioral;
```

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

entity res_reg is
    port(   inp_data : in std_logic_vector (15 downto 0);
            clk,clr,ld_res : in std_logic;
            out_data : out std_logic_vector (15 downto 0));
end res_reg;

architecture Behavioral of res_reg is

begin
    process(clk,clr,ld_res)
    begin
        if(clr = '1') then
            out_data <= (others => '0');
        elsif (rising_edge(clk)) then
            if (ld_res = '1') then
                out_data <= inp_data;
            end if;
        end if;
    end process;
end Behavioral;
```

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

entity clock_pulse is
    port(   go_start : in std_logic;
```

```
            go_mult : in std_logic;
--            clr_mult : in std_logic;
            clk : in std_logic;
            clr : in std_logic;
            go_str_out: out std_logic;
            go_outp : out std_logic
--            clr_m_outp : out std_logic
            );
end clock_pulse;

architecture Behavioral of clock_pulse is
signal delay1,delay2,delay3,delay4,delay5,delay6 : std_logic;
signal delay7,delay8,delay9 : std_logic;

begin
    go_mult_pulse: process (clk,clr)
    begin
        if (clr = '1') then
            delay1 <= '0';
            delay2 <= '0';
            delay3 <= '0';
        elsif (rising_edge(clk)) then
            delay1 <= go_mult;
            delay2 <= delay1;
            delay3 <= delay2;
        end if;
    end process go_mult_pulse;
    go_outp <= delay1 and delay2 and (not delay3);

--    clr_mult_pulse: process(clk,clr)
--    begin
--        if (clr = '1') then
--            delay4 <= '0';
--            delay5 <= '0';
--            delay6 <= '0';
--        elsif (rising_edge(clk)) then
--            delay4 <= clr_mult;
--            delay5 <= delay4;
--            delay6 <= delay5;
--        end if;
--    end process;
--    clr_m_outp <= delay4 and delay5 and (not delay6);

    start_pulse: process (clk,clr)
    begin
        if (clr = '1') then
            delay7 <= '0';
            delay8 <= '0';
            delay9 <= '0';
        elsif (rising_edge(clk)) then
            delay7 <= go_start;
```

```
            delay8 <= delay7;
            delay9 <= delay8;
        end if;
    end process;
    go_str_out <= delay7 and delay8 and (not delay9);
end Behavioral;
```

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

entity comparator is
    port(   inp1,inp2 : in std_logic_vector (2 downto 0);
            g,e,l : out std_logic);
end comparator;

architecture Behavioral of comparator is

begin
    process (inp1,inp2)
    begin
        if (inp1 > inp2) then
            g <= '1';
            e <= '0';
            l <= '0';
        elsif (inp1 = inp2) then
            g <= '0';
            e <= '1';
            l <= '0';
        else
            g <= '0';
            e <= '0';
            l <= '1';
        end if;
    end process;

end Behavioral;
```

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

entity substractor is
    port(   x : in std_logic_vector (2 downto 0);
            out_subs : out std_logic_vector (2 downto 0));
end substractor;

architecture Behavioral of substractor is

begin
    out_subs <= x-"001";

end Behavioral;
```

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

entity multiplier is
    port(   clk : in std_logic;
            clr_mult : in std_logic;
            go_mult : in std_logic;
            multiplicando: in std_logic_vector (15 downto 0);
            multiplicador: in std_logic_vector (2 downto 0);
            done_mult : out std_logic;
            producto : out std_logic_vector (15 downto 0)
        );
end multiplier;

architecture Behavioral of multiplier is
signal x,res : std_logic_vector (15 downto 0);
signal y : std_logic_vector (2 downto 0);

begin
    multiplication: process(clk,clr_mult)
    variable calc,donev : std_logic;
    begin
        if (clr_mult = '1') then
            x <= (others => '0');
            y <= (others => '0');
            res <= (others => '0');
            calc := '0';
            donev := '0';
        elsif (rising_edge(clk)) then
            donev := '0';
            if (go_mult = '1') then
                x <= multiplicando;
                y <= multiplicador;
                calc := '1';
            elsif (calc = '1') then
                if (y > "000") then
                    res <= res+x;
                    y <= y-"001";
                elsif (y = "000") then
                    producto <= res;
                    donev := '1';
                    calc := '1';
                else
                    producto <= (others => '0');
                end if;
            end if;
        end if;
        done_mult <= donev;
    end process multiplication;
```

```
end Behavioral;
```

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

entity out_reg is
    port(   inp_data : in std_logic_vector (15 downto 0);
            clk,clr,ld_out : in std_logic;
            out_data : out std_logic_vector (15 downto 0));
end out_reg;

architecture Behavioral of out_reg is

begin
    process (clk,clr,ld_out)
    begin
        if (clr = '1') then
            out_data <= (others => '0');
        elsif (rising_edge(clk)) then
            if (ld_out = '1') then
                out_data <= inp_data;
            end if;
        end if;
    end process;
end Behavioral;
```

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

entity control_unit is
    port(   go : in std_logic;
            clk : in std_logic;
            clr : in std_logic;
            comp_in : in std_logic_vector (2 downto 0);
            done_mult : in std_logic;
            x_sel : out std_logic;
            res_sel : out std_logic;
            ld_x : out std_logic;
            ld_res : out std_logic;
            ld_out : out std_logic;
            go_mult : out std_logic;
            clr_mult : out std_logic
);
end control_unit;

architecture Behavioral of control_unit is
type state_type is (q0, q1, q2, q3, q4,q5);
signal present_state, next_state : state_type;

begin
```

```
--Registro de los 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,go,comp_in,done_mult)
begin
    case(present_state) is
        when q0 =>
            if (go <= '0') then
                next_state <= q0;
            else
                next_state <= q1;
            end if;
        when q1 =>
                next_state <= q2;
        when q2 =>
            if (comp_in = "001") then
                next_state <= q5;
            elsif (comp_in = "010") then
                next_state <= q5;
            else
                next_state <= q3;
            end if;
        when q3 =>
            if (done_mult = '1') then
                next_state <= q4;
            else
                next_state <= q3;
            end if;
        when q4 =>
            next_state <= q2;
        when q5 =>
            next_state <= q0;
        when others =>
            null;
    end case;
end process C1;

--Registro de las salidas
C2: process(present_state)
begin
    if (present_state = q0) then
        x_sel <= '0';
        res_sel <= '0';
```

```
            ld_x <= '0';
            ld_res <= '0';
            ld_out <= '0';
            go_mult <= '0';
            clr_mult <= '0';
        elsif (present_state = q1) then
            x_sel <= '0';
            res_sel <= '0';
            ld_x <= '1';
            ld_res <= '1';
            ld_out <= '0';
            go_mult <= '0';
            clr_mult <= '0';
        elsif (present_state = q2) then
            x_sel <= '1';
            res_sel <= '1';
            ld_x <= '0';
            ld_res <= '0';
            ld_out <= '0';
            go_mult <= '0';
            clr_mult <= '1';
        elsif (present_state = q3) then
            x_sel <= '1';
            res_sel <= '1';
            ld_x <= '0';
            ld_res <= '0';
            ld_out <= '0';
            go_mult <= '1';
            clr_mult <= '0';
        elsif (present_state = q4) then
            x_sel <= '1';
            res_sel <= '1';
            ld_x <= '1';
            ld_res <= '1';
            ld_out <= '0';
            go_mult <= '0';
            clr_mult <= '0';
        elsif (present_state = q5) then
            x_sel <= '0';
            res_sel <= '0';
            ld_x <= '0';
            ld_res <= '0';
            ld_out <= '1';
            go_mult <= '0';
            clr_mult <= '1';
        end if;
    end process;

end Behavioral;
```

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

```
use ieee.std_logic_unsigned.all;


entity X7segb is
    Port ( x : in STD_LOGIC_VECTOR (15 downto 0);
           cclk,clr : in STD_LOGIC;
           a_to_g : out STD_LOGIC_VECTOR (6 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0));
end X7segb;
architecture Behavioral of X7segb is
    --se?ales intermedias para conexi?n interna
    signal digit : std_logic_vector (3 downto 0);
    signal count : std_logic_vector (1 downto 0);
    signal aen : std_logic_vector (3 downto 0);
    --se?ales para el divisor de frecuencia
    signal count_1 : std_logic_vector(17 downto 0);
    signal clk190 : std_logic;
    ----------------------------
begin
    --Divisor de frecuencia
    count_1 <= count_1+1 when rising_edge(cclk);
    clk190 <= count_1(17);
    --control de display en blanco
    aen(3) <= (x(15) or x(14) or x(13) or x(12)); --Activa 3er d?gito
    --si hay datos a la entrada
    aen(2) <= (x(15) or x(14) or x(13) or x(12) or --Activa 3er d?gito
                x(11) or x(10) or x(9) or x(8));--si hay datos a la entrada
    --o si hay datos para d?gito 4
    aen(1) <= (x(15) or x(14) or x(13) or x(12) or --Activa 2o d?gito
               x(11) or x(10) or x(9) or x(8)) or --si hay datos a la entrada
              x(7) or x(6) or x(5) or x(4); --o si hay datos para d?gito 3 ? 4
    aen(0) <= '1'; --Primer d?gito (de derecha a izquierda) siempre encendido
    --contador de 2 bits
    cont_2bit: process (clk190,clr)
    begin
        if (clr = '1') then
            count <= "00";
        elsif (rising_edge(clk190)) then
            count <= count + 1;
        end if;
    end process cont_2bit;
    --Multiplexor de 4x1
    --Selecciona (con la se?al count) 1 de 4 entradas (de 4 bits cada una)
    with count select
 digit <= x(3 downto 0) when "00",
        x(7 downto 4) when "01",
        x(11 downto 8) when "10",
        x(15 downto 12) when others;
    --seg7dec
    --Convierte un n?mero binario de 4 bits a 7 segmentos
    with digit select
```

```
    a_to_g <= "1001111" when "0001", --1
              "0010010" when "0010", --2
              "0000110" when "0011", --3
              "1001100" when "0100", --4
              "0100100" when "0101", --5
              "0100000" when "0110", --6
              "0001111" when "0111", --7
              "0000000" when "1000", --8
              "0000100" when "1001", --9
              "0001000" when "1010", --A
              "1100000" when "1011", --B
              "0110001" when "1100", --C
              "1000010" when "1101", --D
              "0110000" when "1110", --E
              "0111000" when "1111", --F
              "0000001" when others; --0
    --selecci?n del d?gito
    ancode: process (count)
    begin
        if (aen(conv_integer(count)) = '1') then --convierte el valor de count(1
            an <= (others => '1'); --asigna '1s' a toda la se?al an(3:0) = "1111
            an(conv_integer(count)) <= '0';
        else
            an <= "1111";
        end if;
    end process ancode;
end Behavioral;
```

---

7. Apéndice C

Resultados de la simulación

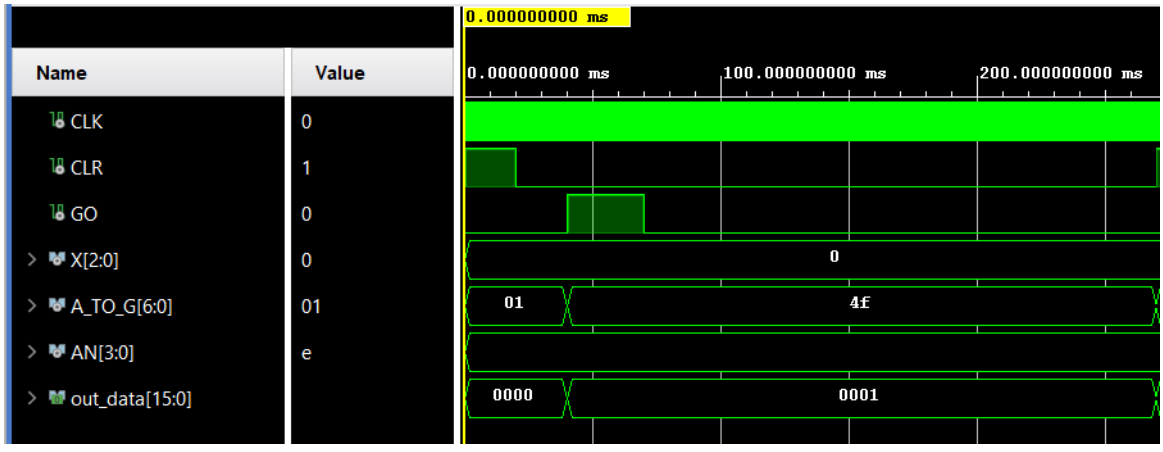


Figura 3: Simulación para n=0

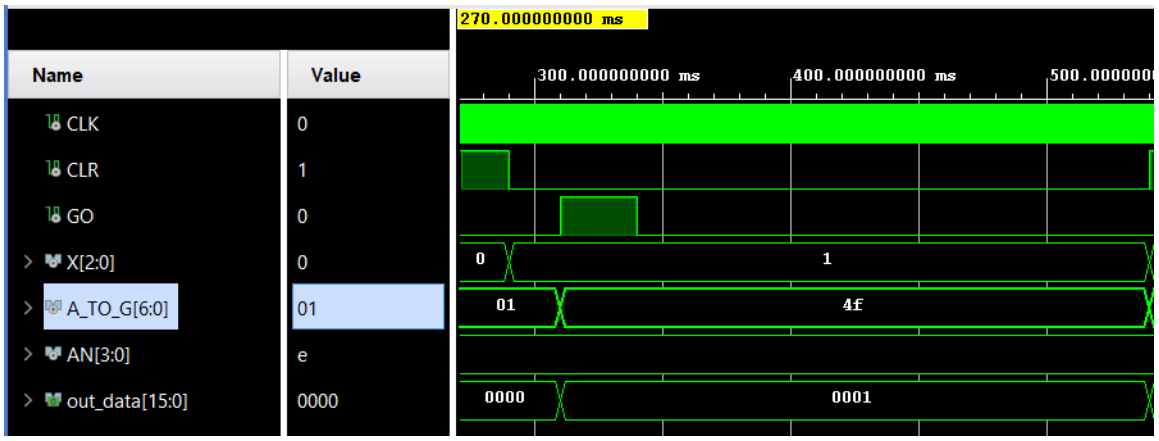


Figura 4: Simulación para n=1

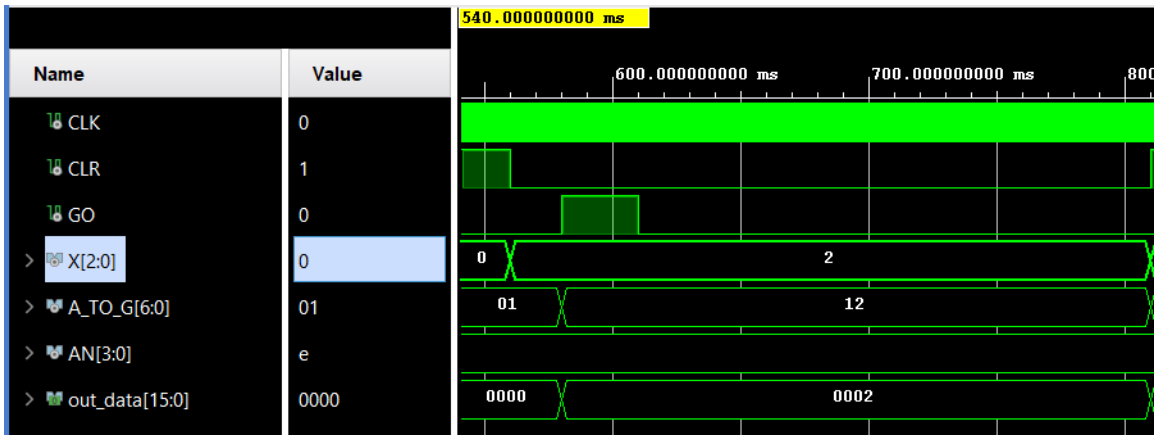


Figura 5: Simulación para n=2

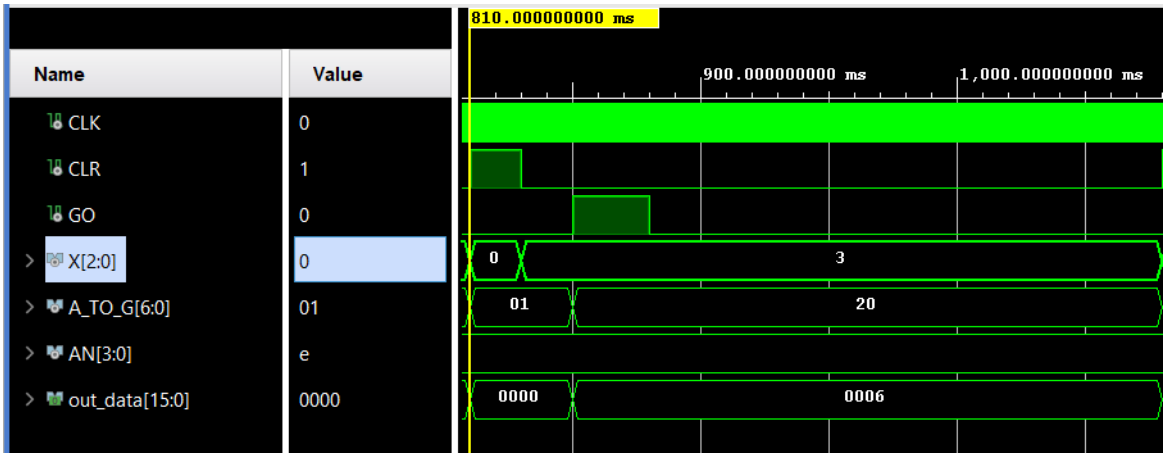


Figura 6: Simulación para n=3

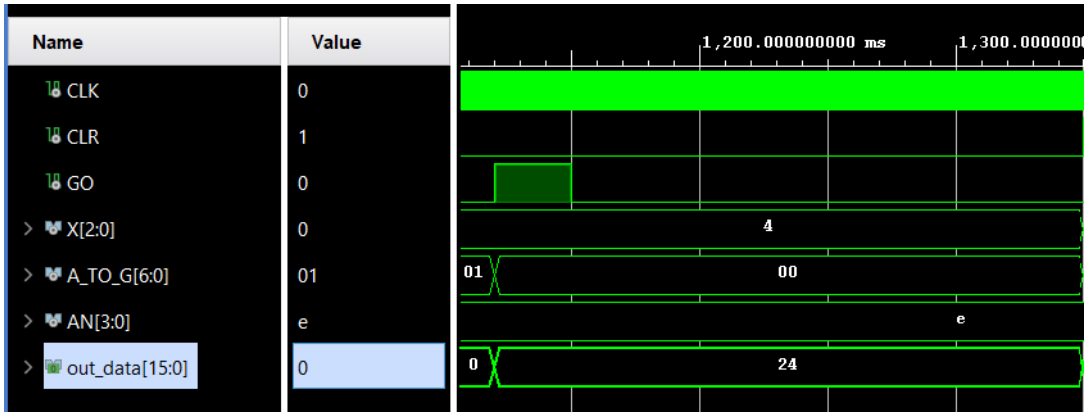


Figura 7: Simulación para n=4