TP N°2: Voltimetro digital con salida VGA

Vázquez, Matías - 91523
mfvazquez@gmail.com

12 de diciembre de 2015

En el presente Trabajo Práctico se implementará en FPGA un sistema digital para un voltímetro digital con salida VGA.

1. Especificaciones

Se implementó en lenguaje descriptor de hardware VHDL un voltímetro digital con salida VGA. La tensión máxima que podrá mostrar el voltímetro es 3,3 V.

Se utilizó el kit de desarrollo "Spartan-3 Starter Board" de la empresa digilent. Utilizando una frecuencia de clock de 50 MHz.

2. Diseño

2.1. Diagrama en bloques general

A continuación, en la figura 1 se presenta el diagrama en bloques general del sistema. La tensión a ser mostrada por salida VGA es V_{in} que es conectada a la entrada no inversora del amplificador operacional. El amplificador operacional generará a través del Flip-Flop D un PWM de forma que el ciclo de trabajo generado sea equivalente a V_{in}. Por ejemplo: si $V_{in} = 3{,}3\,\text{V}$ el ciclo de trabajo será del 100 %; si $V_{in} = 1{,}65\,\text{V}$ el ciclo de trabajo será del 50 %;

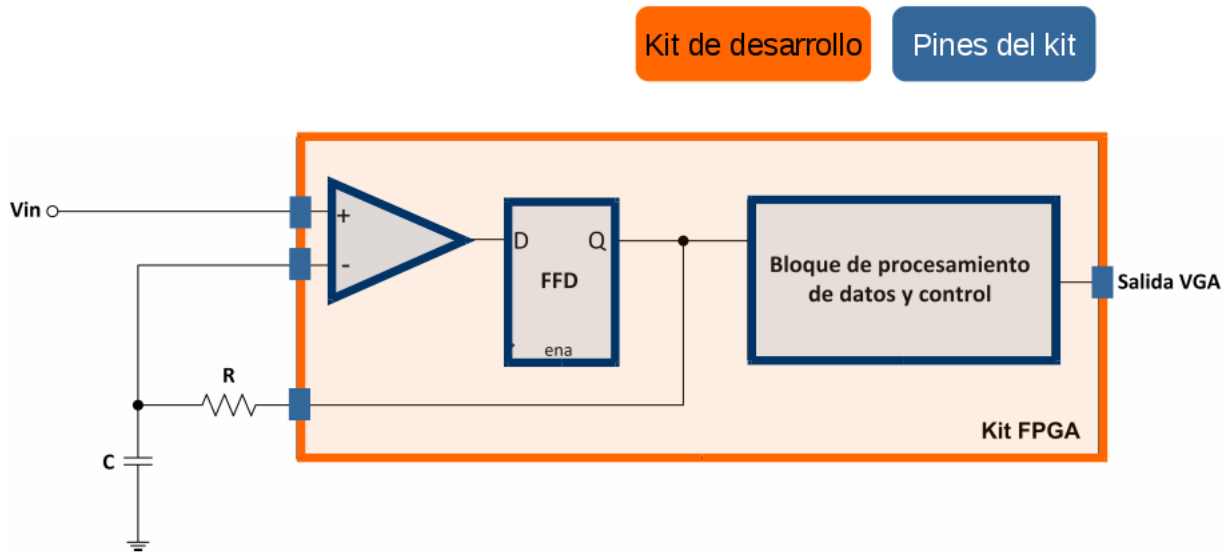

Figura 1: Diagrama en bloques general

2.2. Diagrama en bloques del procesamiento y control

En la figura 2 se presenta el diagrama en bloques del procesamiento y control del voltímetro. Este bloque es el encargado de obtener en base al PWM recibido la tensión V_{in} y codificar dicho valor en un contador BCD de 3 digitos, y mostrar el resultado por salida VGA.

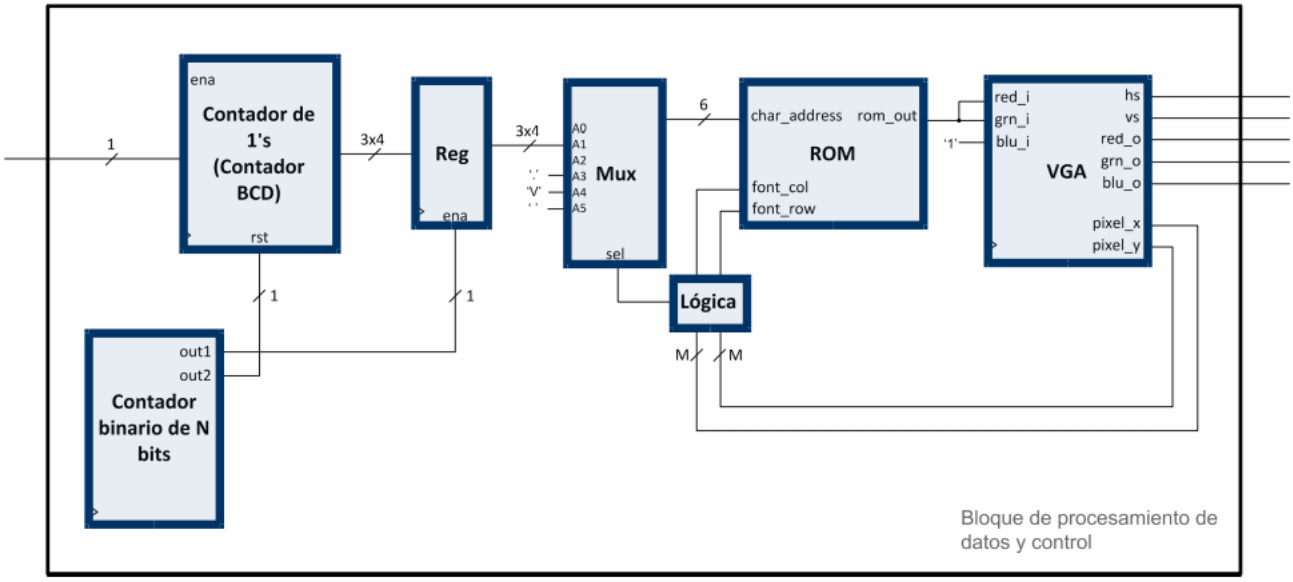

Figura 2: Diagrama en bloques del procesamiento y control

2.3. Contador BCD

Es un contador BCD de 3 dígitos, cuenta los ciclos de reloj en que la entrada recibida se encuentra en 1. Cuenta con una entrada `rst` para ser reiniciado.

2.4. Contador binario de N bits

Siendo N = 9 bits, este contador cuenta 330 ciclos de reloj, para luego guardar la salida del contador BCD en el registro y finalmente reiniciar el contador BCD. Notar que si la entrada del contador BCD es un PWM del 100 % la salida del contador BCD será 330 (en decimal) en el momento en el que el contador binario de N bits haya contado los 330 ciclos de reloj. Si el PWM es del 50 % el contador BCD solo contará 165 ciclos para cuando se guarde su valor en el registro.

2.5. Multiplexor

El multiplexor es utilizado por el bloque `Lógica` para seleccionar el caracter que se necesite mostrar por pantalla.

2.6. ROM

En la memoria ROM serán guardadas las estructuras de los caracteres necesarios para mostrar el resultado por pantalla. Estos son los números del '0' al '9', ' ', ',' y 'V'.

2.7. Lógica

Este bloque recibe la fila y columna del pixel actual del bloque "VGAz en base a este dato deberá seleccionar el carácter que deba ser mostrado. En caso de no estar en un pixel correspondiente a la medición del voltímetro seleccionará el carácter ' ' utilizando el multiplexor.

3. Simulaciones

A continuación se muestran algunas simulaciones realizadas mediante pruebas.

3.1. Contador BCD

En esta simulación se comprobó que el contador BCD cuenta como un número decimal de 3 dígitos, cada dígito esta representado por 4 bits.

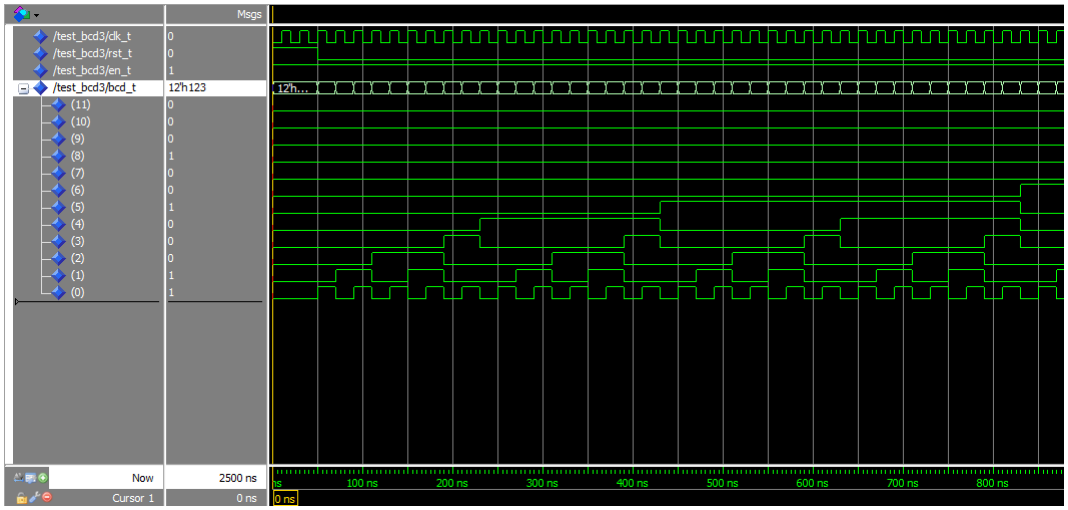

Figura 3: Simulación del contador BCD

3.2. Contador binario de N bits

Se verificó que los flags de salida de este bloque se ejecuten en el orden necesario. Como se verifica en la figura 4 primero se habilita el flag `out_1` que corresponde al registro, guardando así el valor de la salida del contador BCD en ese momento. Paso siguiente reinicia el contador BCD.

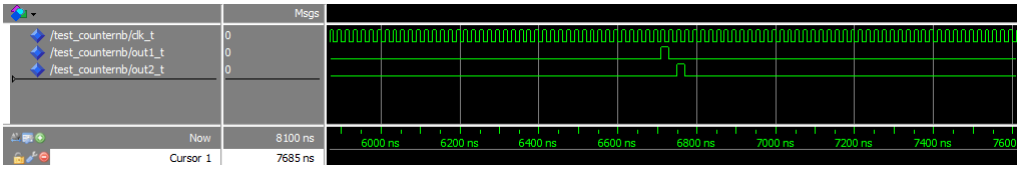

Figura 4: Simulación del contador binario de N bits

3.3. Multiplexor

Las entradas del multiplexor eran: “0000”, “0000”, “1000”, “0100”, “0010” y “0001”. se fue incrementando en 1 los bits de selección desde 0 hasta 3. Se verificó que la salida del multiplexor correspondía a la siguiente secuencia: “0001”, “0010”, “0100” y “1000”.

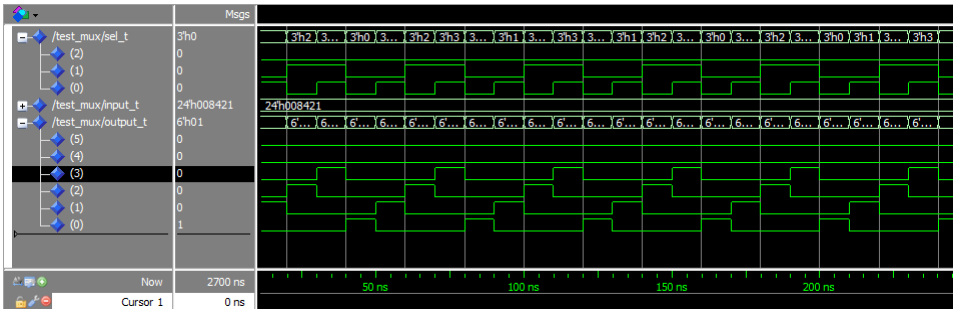

Figura 5: Simulación del multiplexor

3.4. Registro

Se verificó que la salida del registro es igual a la entrada cuando el flag `en_t` valía 1. Cuando valía 0 guardaba el último dato mostrado.

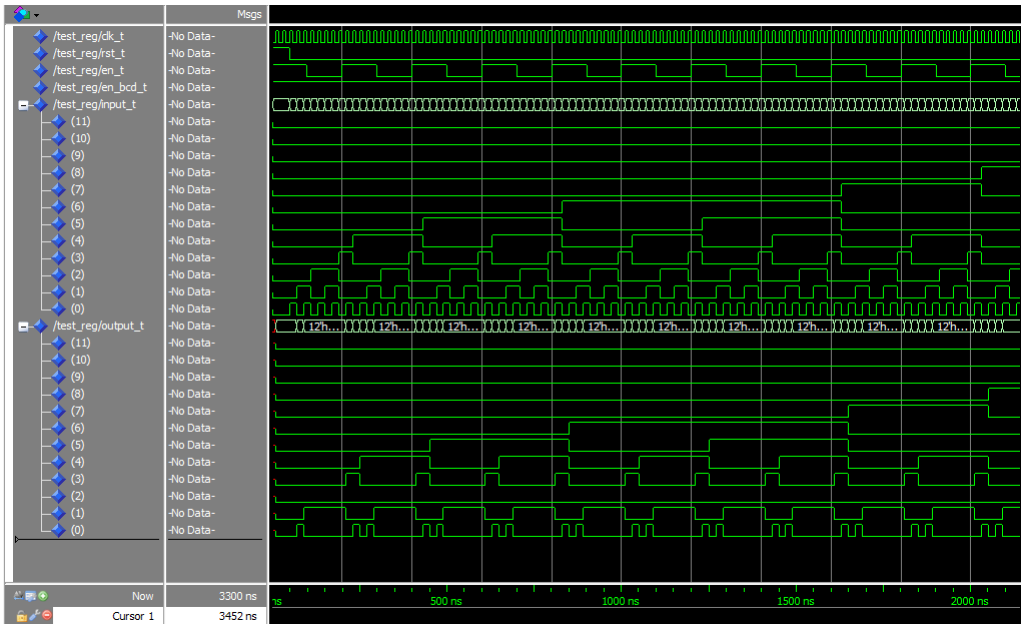

Figura 6: Simulación del registro

3.5. Sampler

Este bloque esta compuesto por el contador BCD, el contador binario de N bits y el registro. Se ingresó un PWM de 80 % y se verificó que la salida era 265, valor que corresponde a 2,65 V. Se puede comprobar que si un PWM del 100 % corresponde a 3,30 V entonces un PWM del 80 % corresponde a 2,64 V

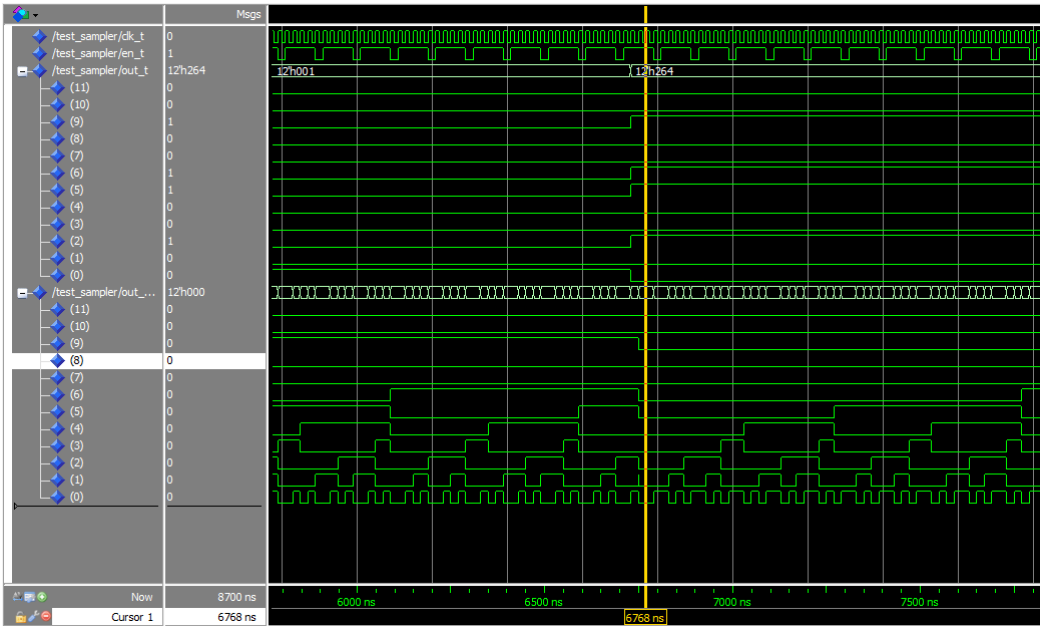

Figura 7: Simulación del sampler

4. Resumen de síntesis

Utilización Lógica	Usados	Utilización
Slices	116	2 %
Flip-Flops	65	0 %
LUTs	213	2 %
GCLK	1	4 %

5. Código fuente VHDL

5.1. BCD.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity BCD is
	port(
		clk: in std_logic;
		rst: in std_logic;
		en: in std_logic;
		cuenta: out std_logic_vector(3 downto 0);
		flag: out std_logic
	);
end;

architecture BCD_arq of BCD is
	signal flag_aux: std_logic;
begin
	process(clk, rst, en)
		variable aux: unsigned(3 downto 0);
	begin
		if rst = '1' then
			aux := "0000";
			flag_aux <= '0';
		elsif rising_edge(clk) then
			if en = '1' then
				aux := aux + 1;
				if aux = "1001" then
					flag_aux <= '1';
				elsif aux = "1010" then
					aux := "0000";
					flag_aux <= '0';
				end if;
			end if;
		end if;
		cuenta <= std_logic_vector(aux);
	end process;
	flag <= flag_aux and en;
end;
```

5.2. BCD3.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity BCD3 is
        port(
                clk: in std_logic;
                rst: in std_logic;
                en: in std_logic;
                cuenta: out std_logic_vector(11 downto 0)
        );
end;

architecture Beh of BCD3 is

        signal clk_aux: std_logic;
        signal rst_aux: std_logic;
        signal en_aux: std_logic_vector(3 downto 0);
        signal bcd_aux: std_logic_vector(11 downto 0);

        component BCD is
                port(
                        clk: in std_logic;
                        rst: in std_logic;
                        en: in std_logic;
                        cuenta: out std_logic_vector(3 downto 0);
                        flag: out std_logic
                );
        end component;

begin

        en_aux(0) <= en;
        clk_aux <= clk;
        rst_aux <= rst;
        cuenta <= bcd_aux;

        bcds_3:
        for i in 0 to 2 generate

        inst_BCD: BCD
                port map(
                        clk => clk_aux,
                        rst => rst_aux,
                        en => en_aux(i),
                        cuenta => bcd_aux(i*4+3 downto i*4),
                        flag => en_aux(i+1)
                );

    end generate;

end;
```

5.3. BCD3_test.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity test_BCD3 is
end;

architecture Beh of test_BCD3 is
        signal clk_t: std_logic := '0';
        signal rst_t: std_logic := '1';
        signal en_t: std_logic := '1';
        signal bcd_t: std_logic_vector(11 downto 0);


        component BCD3 is
                port(
                        clk: in std_logic;
                        rst: in std_logic;
                        en: in std_logic;
                        cuenta: out std_logic_vector(11 downto 0)
                );
        end component;

begin

        inst_BCD3: BCD3
        port map(
                clk => clk_t,
                rst => rst_t,
                en => en_t,
                cuenta => bcd_t
        );

        clk_t <= not clk_t after 10 ns;
        rst_t <= '0' after 50 ns;
end;
```

5.4. Char_ROM.vhd

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

entity Char_ROM is
        generic(
                N: integer:= 6;
                M: integer:= 3;
                W: integer:= 8
        );
        port(
                char_address: in std_logic_vector(5 downto 0);
                font_row, font_col: in std_logic_vector(M-1 downto 0);
                rom_out: out std_logic
        );
end;

architecture p of Char_ROM is
        subtype tipoLinea is std_logic_vector(0 to W-1);

        type char is array(0 to W-1) of tipoLinea;
        constant Cero: char:= (
                                                        "00011100",
                                                        "00110110",
                                                        "01100011",
                                                        "01101011",
                                                        "01100011",
                                                        "00110110",
                                                        "00011100",
                                                        "00000000"
                                                );
        constant Uno: char:= (
                                                        "00001100",
                                                        "00111100",
                                                        "00001100",
                                                        "00001100",
                                                        "00001100",
                                                        "00001100",
                                                        "00111111",
                                                        "00000000"
                                                );
        constant Dos: char:= (
                                                        "00111110",
                                                        "01100011",
                                                        "00000110",
                                                        "00001100",
                                                        "00011000",
                                                        "00110000",
                                                        "01111111",
                                                        "00000000"
                                                );

        constant Tres: char:= (
                                                        "01111111",
                                                        "00000110",
                                                        "00001100",
                                                        "00000110",
```

```
                                                    "00000011",
                                                    "01100110",
                                                    "00111100",
                                                    "00000000"
                                                );
    constant Cuatro: char:= (
                                                    "00000110",
                                                    "00001110",
                                                    "00010110",
                                                    "00100110",
                                                    "01111111",
                                                    "00000110",
                                                    "00000110",
                                                    "00000000"
                                                );
    constant Cinco: char:= (
                                                    "01111111",
                                                    "01100000",
                                                    "01111110",
                                                    "00000011",
                                                    "00000011",
                                                    "01100110",
                                                    "00111100",
                                                    "00000000"
                                                );
    constant Seis: char:= (
                                                    "00000110",
                                                    "00001100",
                                                    "00110000",
                                                    "01100000",
                                                    "01111110",
                                                    "01000011",
                                                    "00111100",
                                                    "00000000"
                                                );
    constant Siete: char:= (
                                                    "01111111",
                                                    "00000110",
                                                    "00001100",
                                                    "00011000",
                                                    "00110000",
                                                    "01100000",
                                                    "01100000",
                                                    "00000000"
                                                );
    constant Ocho: char:= (
                                                    "00111110",
                                                    "01100011",
                                                    "01100011",
                                                    "01111111",
                                                    "01100011",
                                                    "01100011",
                                                    "00111110",
                                                    "00000000"
                                                );
    constant Nueve: char:= (
                                                    "00111110",
                                                    "01100011",
                                                    "00111110",
                                                    "00000110",
```

```
                                                    "00000110",
                                                    "00011000",
                                                    "01100000",
                                                    "00000000"
                                                );
    constant Punto: char:= (
                                                    "00000000",
                                                    "00000000",
                                                    "00000000",
                                                    "00000000",
                                                    "00000000",
                                                    "00011000",
                                                    "00011000",
                                                    "00000000"
                                                );
    constant V: char:= (
                                                    "10000001",
                                                    "10000001",
                                                    "01000010",
                                                    "01000010",
                                                    "00100100",
                                                    "00100100",
                                                    "00011000",
                                                    "00000000"
                                                );
    constant Espacio: char:= (
                                                    "00000000",
                                                    "00000000",
                                                    "00000000",
                                                    "00000000",
                                                    "00000000",
                                                    "00000000",
                                                    "00000000",
                                                    "00000000"
                                                );

    type memo is array(0 to 255) of tipoLinea;
    signal RAM: memo:= (
            0 => Cero(0), 1 => Cero(1), 2 => Cero(2), 3 => Cero(3),
            4 => Cero(4), 5 => Cero(5), 6 => Cero(6), 7 => Cero(7),

            8 => Uno(0), 9 => Uno(1), 10 => Uno(2), 11 => Uno(3),
            12 => Uno(4), 13 => Uno(5), 14 => Uno(6), 15 => Uno(7),

            16 => Dos(0), 17 => Dos(1), 18 => Dos(2), 19 => Dos(3),
            20 => Dos(4), 21 => Dos(5), 22 => Dos(6), 23 => Dos(7),

            24 => Tres(0), 25 => Tres(1), 26 => Tres(2), 27 => Tres(3),
            28 => Tres(4), 29 => Tres(5), 30 => Tres(6), 31 => Tres(7),

            32 => Cuatro(0), 33 => Cuatro(1), 34 => Cuatro(2), 35 => Cuatro(3),
            36 => Cuatro(4), 37 => Cuatro(5), 38 => Cuatro(6), 39 => Cuatro(7),

            40 => Cinco(0), 41 => Cinco(1), 42 => Cinco(2), 43 => Cinco(3),
            44 => Cinco(4), 45 => Cinco(5), 46 => Cinco(6), 47 => Cinco(7),

            48 => Seis(0), 49 => Seis(1), 50 => Seis(2), 51 => Seis(3),
            52 => Seis(4), 53 => Seis(5), 54 => Seis(6), 55 => Seis(7),

            56 => Siete(0), 57 => Siete(1), 58 => Siete(2), 59 => Siete(3),
```

```
                60 => Siete(4), 61 => Siete(5), 62 => Siete(6), 63 => Siete(7),

                64 => Ocho(0), 65 => Ocho(1), 66 => Ocho(2), 67 => Ocho(3),
                68 => Ocho(4), 69 => Ocho(5), 70 => Ocho(6), 71 => Ocho(7),

                72 => Nueve(0), 73 => Nueve(1), 74 => Nueve(2), 75 => Nueve(3),
                76 => Nueve(4), 77 => Nueve(5), 78 => Nueve(6), 79 => Nueve(7),

                80 => Punto(0), 81 => Punto(1), 82 => Punto(2), 83 => Punto(3),
                84 => Punto(4), 85 => Punto(5), 86 => Punto(6), 87 => Punto(7),

                88 => V(0), 89 => V(1), 90 => V(2), 91 => V(3), 92 => V(4),
                93 => V(5), 94 => V(6), 95 => V(7),

                96 => Espacio(0), 97 => Espacio(1), 98 => Espacio(2),
                99 => Espacio(3), 100 => Espacio(4), 101 => Espacio(5),
                102 => Espacio(6), 103 => Espacio(7),

                104 to 255 => "00000000"
        );

        signal char_addr_aux: std_logic_vector(8 downto 0);

begin

        char_addr_aux <= char_address & font_row;
        rom_out <= RAM(conv_integer(char_addr_aux))(conv_integer(font_col));

end;
```

5.5. CounterNb.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity CounterNb is
        port(
                clk: in std_logic;
                out1: out std_logic;
                out2: out std_logic := '0'
        );
end;

architecture Beh of CounterNb is
begin
        process(clk)
                variable rst: std_logic := '1';
                variable out_rst: unsigned(1 downto 0) := "10";
                variable cuenta: unsigned(8 downto 0) := (others => '0');

        begin
                if rising_edge(clk) then
                        if out_rst = "01" then
                                out1 <= '0';
                                out_rst := "10";
                        elsif out_rst = "10" then
                                out2 <= '1';
                                out_rst := "11";
                        elsif out_rst = "11" then
                                out2 <= '0';
                                out_rst := "00";
                        else
                                cuenta := cuenta + 1;
                                if rst = '1' then
                                        cuenta := (others => '0');
                                        rst := '0';
                                        out_rst := "01";
                                        out1 <= '1';
                                elsif cuenta = "101001001" then
                                        rst := '1';
                                end if;
                        end if;
                end if;
        end process;
end;
```

5.6. CounterNb_test.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity test_CounterNb is
end;

architecture beh of test_CounterNb is
        signal clk_t: std_logic := '0';
        signal out1_t: std_logic;
        signal out2_t: std_logic;

        component CounterNb is
                port(
                        clk: in std_logic;
                        out1: out std_logic := '0';
                        out2: out std_logic := '0'
                );
        end component;

begin
        inst_CounterNb: CounterNb
                port map(
                        clk => clk_t,
                        out1 => out1_t,
                        out2 => out2_t
                );
        clk_t <= not clk_t after 10 ns;
end;
```

5.7. FFD.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FFD is
        port(
                clk: in std_logic;
                rst: in std_logic;
                ena: in std_logic;
                D: in std_logic;
                Q: out std_logic
        );
end FFD;

architecture beh of ffd is
begin
        process(clk, rst)
        begin
                if rst = '1' then
                        Q <= '0';
                elsif rising_edge(clk) then
                        if ena = '1' then
                                Q <= D;
                        end if;
                end if;
        end process;
end beh;
```

5.8. Logic.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Logic is
	port(
		pixel_row, pixel_col: in std_logic_vector(9 downto 0);
		font_row, font_col: out std_logic_vector(2 downto 0);
		sel: out std_logic_vector(2 downto 0)
	);
end;

architecture Beh of Logic is
begin
	process(pixel_row, pixel_col)
		variable col, col_aux, fila, fila_aux, char_col, char_fila: unsigned(9 d
		variable pixel_col_aux, pixel_fila_aux: unsigned(2 downto 0);
	begin
		col := unsigned(pixel_col);
		fila := unsigned(pixel_row);
		col_aux := col mod 8;
		fila_aux := fila mod 8;
		char_col := col/8;
		char_fila := fila/8;

		pixel_col_aux := col_aux(2 downto 0);
		pixel_fila_aux := fila_aux(2 downto 0);
		font_row <= std_logic_vector(pixel_fila_aux);
		font_col <= std_logic_vector(pixel_col_aux);

		if char_fila = 30 then
			if char_col = 38 then
				sel <= "000";   -- (0) digito mas significativo
			elsif char_col = 39 then
				sel <= "011";   -- (3) punto
			elsif char_col = 40 then
				sel <= "001";   -- (1) primer decimal
			elsif char_col = 41 then
				sel <= "010";   -- (2) segundo decimal
			elsif char_col = 42 then
				sel <= "100";   -- (4) V
			else
				sel <= "101";   -- (5) espacio
			end if;
		else
			sel <= "101";           -- (5) espacio
		end if;
	end process;
end;
```

5.9. Mux.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Mux is
	port(
		input: in std_logic_vector(23 downto 0);
		sel: in std_logic_vector(2 downto 0);
		output: out std_logic_vector(5 downto 0)
	);

end;

architecture Beh of Mux is

	signal sel_aux: std_logic_vector(2 downto 0);
	signal input_aux: std_logic_vector(23 downto 0);
	signal output_aux: std_logic_vector(5 downto 0);

	component Mux_gen is
		generic(
			inputs: natural := 16;
			data_bus: natural := 4;
			sel_length: natural := 2
		);
		port(
			input: in std_logic_vector(inputs-1 downto 0);
			sel: in std_logic_vector(sel_length-1 downto 0);
			output: out std_logic_vector(data_bus-1 downto 0)
		);
	end component;

begin

	inst_Mux_gen: Mux_gen
		generic map(
			inputs => 24,
			data_bus => 4,
			sel_length => 3
		)
		port map(
			input => input_aux,
			sel => sel_aux,
			output => output_aux(3 downto 0)
		);

	output_aux(5 downto 4) <= "00";
	output <= output_aux;
	input_aux <= input;
	sel_aux <= sel;

end;
```

5.10. Mux_gen.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Mux_gen is
        generic(
                inputs: natural := 16;
                data_bus: natural := 4;
                sel_length: natural := 2
        );
        port(
                input: in std_logic_vector(inputs-1 downto 0);
                sel: in std_logic_vector(sel_length-1 downto 0);
                output: out std_logic_vector(data_bus-1 downto 0)
        );

end;

architecture Beh of Mux_gen is
begin
        process(sel, input)
        variable sel_aux: unsigned(sel_length-1 downto 0);
        variable fin, inicio: natural;
        begin
                sel_aux := unsigned(sel);
                inicio := to_integer(sel_aux);
                inicio := inicio * data_bus;
                fin := inicio + data_bus - 1;

                output <= input(fin downto inicio);
        end process;
end;
```

5.11. Mux_test.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity test_Mux is
end;

architecture Beh of test_Mux is
        signal  sel_t: std_logic_vector(2 downto 0) := "000";
        signal  input_t: std_logic_vector(23 downto 0) := "000000001000010000100001";
        signal  output_t: std_logic_vector(5 downto 0);

        component Mux is
                port(
                        input: in std_logic_vector(23 downto 0);
                        sel: in std_logic_vector(2 downto 0);
                        output: out std_logic_vector(5 downto 0)
                );
        end component;

begin

        inst_Mux: Mux
                port map(
                        input => input_t,
                        sel => sel_t,
                        output => output_t
                );
        sel_t(0) <= not sel_t(0) after 10 ns;
        sel_t(1) <= not sel_t(1) after 20 ns;
end;
```

5.12. Process_and_controll.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Process_and_controll is
        port(
                en: in std_logic;
                clk: in std_logic;
                hs: out std_logic;
                vs: out std_logic;
                red_o: out std_logic_vector(2 downto 0);
                grn_o: out std_logic_vector(2 downto 0);
                blu_o: out std_logic_vector(1 downto 0)
        );
end;

architecture Beh of Process_and_controll is

        signal en_aux: std_logic;
        signal clk_aux: std_logic;
        signal output_aux: std_logic_vector(11 downto 0);
        signal input_aux: std_logic_vector(23 downto 0);
        signal hs_aux: std_logic;
        signal vs_aux: std_logic;
        signal red_o_aux: std_logic_vector(2 downto 0);
        signal grn_o_aux: std_logic_vector(2 downto 0);
        signal blu_o_aux: std_logic_vector(1 downto 0);

        component Sampler is
                port(
                        en: in std_logic;
                        clk: in std_logic;
                        output: out std_logic_vector(11 downto 0)
                );
        end component;

        component VGA_volt is
                port(
                        mclk: in std_logic;
                        input: in std_logic_vector(23 downto 0);
                        hs: out std_logic;
                        vs: out std_logic;
                        red_o: out std_logic_vector(2 downto 0);
                        grn_o: out std_logic_vector(2 downto 0);
                        blu_o: out std_logic_vector(1 downto 0)
                );
        end component;

begin

        en_aux <= en;
        clk_aux <= clk;
        hs <= hs_aux;
        vs <= vs_aux;
        red_o <= red_o_aux;
        grn_o <= grn_o_aux;
        blu_o <= blu_o_aux;
```

```
        input_aux(11 downto 8) <= output_aux(3 downto 0);
        input_aux(7 downto 4) <= output_aux(7 downto 4);
        input_aux(3 downto 0) <= output_aux(11 downto 8);

        input_aux(15 downto 12) <= "1010"; -- direccion en memoria del caracter: '.'
        input_aux(19 downto 16) <= "1011"; -- direccion en memoria del caracter: 'V'
        input_aux(23 downto 20) <= "1100"; -- direccion en memoria del caracter: ' '

        inst_Sampler: Sampler
                port map(
                        en => en_aux,
                        clk => clk_aux,
                        output => output_aux
                );

        inst_VGA_volt: VGA_volt
                port map(
                        mclk => clk_aux,
                        input => input_aux,
                        hs => hs_aux,
                        vs => vs_aux,
                        red_o => red_o_aux,
                        grn_o => grn_o_aux,
                        blu_o => blu_o_aux
                );

end;
```

5.13. Reg.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity Reg is
        port(
                clk: in std_logic;
                en: in std_logic;
                input: in std_logic_vector(11 downto 0);
                output: out std_logic_vector(11 downto 0)
        );
end;

architecture Beh of Reg is

begin

        process(clk, en)
        begin
                if rising_edge(clk) and en = '1' then
                        output <= input;
                end if;
        end process;

end;
```

5.14. Reg_test.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity test_Reg is
end;

architecture Beh of test_Reg is
        signal clk_t: std_logic := '0';
        signal rst_t: std_logic := '1';
        signal en_t: std_logic := '1';
        signal en_bcd_t: std_logic := '1';
        signal input_t: std_logic_vector(11 downto 0);
        signal output_t: std_logic_vector(11 downto 0);

        component Reg is
                port(
                        clk: in std_logic;
                        en: in std_logic;
                        input: in std_logic_vector(11 downto 0);
                        output: out std_logic_vector(11 downto 0)
                );
        end component;



        component BCD3 is
                port(
                        clk: in std_logic;
                        rst: in std_logic;
                        en: in std_logic;
                        cuenta: out std_logic_vector(11 downto 0)
                );
        end component;

begin

        inst_Reg: Reg
        port map(
                        clk => clk_t,
                        en => en_t,
                        input => input_t,
                        output => output_t
        );


        inst_BCD3: BCD3
        port map(
                clk => clk_t,
                rst => rst_t,
                en => en_bcd_t,
                cuenta => input_t
        );

        clk_t <= not clk_t after 10 ns;
        rst_t <= '0' after 50 ns;
        en_t <= not en_t after 100 ns;
```

```
end;
```

5.15. Sampler.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Sampler is
        port(
                en: in std_logic;
                clk: in std_logic;
                out_aux: out std_logic_vector(11 downto 0);
                output: out std_logic_vector(11 downto 0)
        );
end;

architecture Beh of Sampler is
        signal out1_aux: std_logic;
        signal out2_aux: std_logic;
        signal clk_aux: std_logic;
        signal en_aux: std_logic;
        signal input_aux: std_logic_vector(11 downto 0);
        signal output_aux: std_logic_vector(11 downto 0);

        component BCD3 is
                port(
                        clk: in std_logic;
                        rst: in std_logic;
                        en: in std_logic;
                        cuenta: out std_logic_vector(11 downto 0)
                );
        end component;

        component Reg is
                port(
                        clk: in std_logic;
                        en: in std_logic;
                        input: in std_logic_vector(11 downto 0);
                        output: out std_logic_vector(11 downto 0)
                );
        end component;

        component CounterNb is
        port(
                clk: in std_logic;
                out1: out std_logic;
                out2: out std_logic
        );
        end component;

begin
        clk_aux <= clk;
        en_aux <= en;
        output <= output_aux;
        out_aux <= input_aux;

        inst_CounterNb: CounterNb
                port map(
                        clk => clk_aux,
                        out1 => out1_aux,
                        out2 => out2_aux
```

```
        );

        inst_BCD3: BCD3
        port map(
                clk => clk_aux,
                rst => out2_aux,
                en => en_aux,
                cuenta => input_aux
        );

        inst_Reg: Reg
        port map(
                        clk => clk_aux,
                        en => out1_aux,
                        input => input_aux,
                        output => output_aux
        );

end;
```

5.16. Sampler_test.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity test_Sampler is
end;

architecture beh of test_Sampler is
        signal clk_t: std_logic := '0';
        signal en_t: std_logic;
        signal out_t: std_logic_vector(11 downto 0);
        signal out_aux_t: std_logic_vector(11 downto 0);

        component Sampler is
                port(
                        en: in std_logic;
                        clk: in std_logic;
                        out_aux: out std_logic_vector(11 downto 0);
                        output: out std_logic_vector(11 downto 0)
                );
        end component;

        component GenEna
                generic(
                        N: natural := 2;
                        M: natural := 1
                );
                port(
                        clk: in std_logic;
                        en: out std_logic
                );
        end component;


begin
        inst_Sampler: Sampler
                port map(
                        en => en_t,
                        clk => clk_t,
                        out_aux => out_aux_t,
                        output => out_t
                );

        inst_genEna: GenEna
                generic map(
                        N => 2,
                        M => 1
                )
                port map(
                        clk => clk_t,
                        en => en_t
                );

        clk_t <= not clk_t after 10 ns;
end;
```

5.17. VGActrl.vhd

```
-- Modulo: Controlador VGA
-- Descripcion:
-- Autor: Sistemas Digitales (66.17)
--        Universidad de Buenos Aires - Facultad de Ingenieria
--        www.campus.fi.uba.ar
-- Fecha: 16/04/13
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity vga_ctrl is
    port (
                mclk: in std_logic;
                red_i: in std_logic;
                grn_i: in std_logic;
                blu_i: in std_logic;
                hs: out std_logic;
                vs: out std_logic;
                red_o: out std_logic_vector(2 downto 0);
                grn_o: out std_logic_vector(2 downto 0);
                blu_o: out std_logic_vector(1 downto 0);
                pixel_row: out std_logic_vector(9 downto 0);
                pixel_col: out std_logic_vector(9 downto 0)
        );

end vga_ctrl;

architecture vga_ctrl_arq of vga_ctrl is

        -- Numero de pixeles en una linea horizontal (800)
        constant hpixels: unsigned(9 downto 0) := "1100100000";
        -- Numero de lineas horizontales en el display (521)
        constant vlines: unsigned(9 downto 0) := "1000001001";

        constant hbp: unsigned(9 downto 0) := "0010010000";
        -- Back porch horizontal (144)
        constant hfp: unsigned(9 downto 0) := "1100010000";
        -- Front porch horizontal (784)
        constant vbp: unsigned(9 downto 0) := "0000011111";
        -- Back porch vertical (31)
        constant vfp: unsigned(9 downto 0) := "0111111111";
        -- Front porch vertical (511)

        -- Contadores (horizontal y vertical)
        signal hc, vc: unsigned(9 downto 0);
        -- Flag para obtener una habilitacion cada dos ciclos de clock
        signal clkdiv_flag: std_logic;
        -- Senal para habilitar la visualizacion de datos
        signal vidon: std_logic;
        -- Senal para habilitar el contador vertical
        signal vsenable: std_logic;


begin
```

```
-- Division de la frecuencia del reloj
process(mclk)
begin
    if rising_edge(mclk) then
        clkdiv_flag <= not clkdiv_flag;
    end if;
end process;

-- Contador horizontal
process(mclk)
begin
    if rising_edge(mclk) then
        if clkdiv_flag = '1' then
            if hc = hpixels then
                hc <= (others => '0');
                vsenable <= '1';
                -- Habilitacion del cont vert
            else
                hc <= hc + 1;
                -- Incremento del cont horiz
                vsenable <= '0';
                -- El cont vert se mantiene deshabilitado
            end if;
        end if;
    end if;
end process;

-- Contador vertical
process(mclk)
begin
    if rising_edge(mclk) then
        if clkdiv_flag = '1' then
            if vsenable = '1' then
                if vc = vlines then
                    vc <= (others => '0');
                else
                    vc <= vc + 1;
                end if;
            end if;
        end if;
    end if;
end process;

hs <= '1' when (hc < "0001100001") else '0';
vs <= '1' when (vc < "0000000011") else '0';

pixel_col <= std_logic_vector(hc - 144) when (vidon = '1')
            else std_logic_vector(hc);
pixel_row <= std_logic_vector(vc - 31) when (vidon = '1')
            else std_logic_vector(vc);

vidon <= '1' when (((hc < hfp) and (hc > hbp)) and ((vc < vfp) and (vc > vbp)))
            else '0';


    red_o <= (others => '1') when (red_i = '1' and vidon = '1')
            else (others => '0');
    grn_o <= (others => '1') when (grn_i = '1' and vidon = '1')
            else (others => '0');
    blu_o <= (others => '1') when (blu_i = '1' and vidon = '1')
```

```
                else (others => '0');

end vga_ctrl_arq;
```

5.18. VGA_volt.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity VGA_volt is
        port(
                mclk: in std_logic;
                input: in std_logic_vector(23 downto 0);
                hs: out std_logic;
                vs: out std_logic;
                red_o: out std_logic_vector(2 downto 0);
                grn_o: out std_logic_vector(2 downto 0);
                blu_o: out std_logic_vector(1 downto 0)
        );
end;

architecture Beh of VGA_volt is

                signal mclk_aux: std_logic;
                signal hs_aux: std_logic;
                signal vs_aux: std_logic;
                signal red_o_aux: std_logic_vector(2 downto 0);
                signal grn_o_aux: std_logic_vector(2 downto 0);
                signal blu_o_aux: std_logic_vector(1 downto 0);
                signal pixel_row_aux: std_logic_vector(9 downto 0);
                signal pixel_col_aux: std_logic_vector(9 downto 0);

                signal char_address_aux: std_logic_vector(5 downto 0);
                signal font_row_aux, font_col_aux: std_logic_vector(2 downto 0);
                signal rom_out_aux: std_logic;

                signal input_aux: std_logic_vector(23 downto 0);
                signal sel_aux: std_logic_vector(2 downto 0);


        component vga_ctrl is
                port (
                        mclk: in std_logic;
                        red_i: in std_logic;
                        grn_i: in std_logic;
                        blu_i: in std_logic;
                        hs: out std_logic;
                        vs: out std_logic;
                        red_o: out std_logic_vector(2 downto 0);
                        grn_o: out std_logic_vector(2 downto 0);
                        blu_o: out std_logic_vector(1 downto 0);
                        pixel_row: out std_logic_vector(9 downto 0);
                        pixel_col: out std_logic_vector(9 downto 0)
                );
        end component;

        component Mux is
                port(
                        input: in std_logic_vector(23 downto 0);
                        sel: in std_logic_vector(2 downto 0);
                        output: out std_logic_vector(5 downto 0)
                );
        end component;
```

```
        component Logic is
                port(
                        pixel_row, pixel_col: in std_logic_vector(9 downto 0);
                        font_row, font_col: out std_logic_vector(2 downto 0);
                        sel: out std_logic_vector(2 downto 0)
                );
        end component;


        component Char_ROM is
        generic(
                N: integer:= 6;
                M: integer:= 3;
                W: integer:= 8
        );
        port(
                char_address: in std_logic_vector(5 downto 0);
                font_row, font_col: in std_logic_vector(M-1 downto 0);
                rom_out: out std_logic
        );
        end component;

begin

        input_aux <= input;
        hs <= hs_aux;
        vs <= vs_aux;
        red_o <= red_o_aux;
        grn_o <= grn_o_aux;
        blu_o <= blu_o_aux;
        mclk_aux <= mclk;

        inst_vga_ctrl: vga_ctrl
                port map (
                        mclk => mclk_aux,
                        red_i => rom_out_aux,
                        grn_i => rom_out_aux,
                        blu_i => '1',
                        hs => hs_aux,
                        vs => vs_aux,
                        red_o => red_o_aux,
                        grn_o => grn_o_aux,
                        blu_o => blu_o_aux,
                        pixel_row => pixel_row_aux,
                        pixel_col => pixel_col_aux
                );

        inst_Mux: Mux
                port map(
                        input => input_aux,
                        sel => sel_aux,
                        output => char_address_aux
                );

        inst_Logic: Logic
                port map(
                        pixel_row => pixel_row_aux,
                        pixel_col => pixel_col_aux,
                        font_row => font_row_aux,
```

```
                        font_col => font_col_aux,
                        sel => sel_aux
                );

        inst_Char_ROM: Char_ROM
        port map(
                char_address => char_address_aux,
                font_row => font_row_aux,
                font_col => font_col_aux,
                rom_out => rom_out_aux
        );

end;
```

5.19. Volt.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity VOLT is
	port(
		clk50: in std_logic;
		data_volt_in_p,data_volt_in_n: in std_logic;
		data_volt_out: out std_logic;
		hs: out std_logic;
		vs: out std_logic;
		red_o: out std_logic_vector(2 downto 0);
		grn_o: out std_logic_vector(2 downto 0);
		blu_o: out std_logic_vector(1 downto 0)
	);

	attribute loc: string;
	attribute slew: string;
	attribute drive: string;
	attribute iostandard: string;
	attribute loc of clk50: signal is "B8";

	attribute iostandard of data_volt_in_p: signal is "LVDS_25";
	attribute loc of data_volt_in_p: signal is "N18";
	attribute iostandard of data_volt_in_n: signal is "LVDS_25";
	attribute loc of data_volt_in_n: signal is "M18";

	attribute loc of data_volt_out: signal is "P18";
	attribute slew of data_volt_out: signal is "FAST";
	attribute drive of data_volt_out: signal is "8";
	attribute iostandard of data_volt_out: signal is "LVCMOS25";

	attribute loc of hs: signal is "T4";
	attribute loc of vs: signal is "U3";
	attribute loc of red_o: signal is "R8 T8 R9";
	attribute loc of grn_o: signal is "P6 P8 N8";
	attribute loc of blu_o: signal is "U4 U5";

end VOLT;

architecture beh of VOLT is

	component IBUFDS
		port(
			I : in std_logic;
			IB : in std_logic;
			O : out std_logic
		);
	end component;


	component FFD is
		port(
			clk: in std_logic;
			rst: in std_logic;
			ena: in std_logic;
			D: in std_logic;
			Q: out std_logic
```

```
        );
    end component;

    component Process_and_controll is
        port (
            en: in std_logic;
            clk: in std_logic;
            hs: out std_logic;
            vs: out std_logic;
            red_o: out std_logic_vector(2 downto 0);
            grn_o: out std_logic_vector(2 downto 0);
            blu_o: out std_logic_vector(1 downto 0)
        );
    end component;

    signal Diff_Input: std_logic;
    signal process_in: std_logic;

begin
    ibuf0: IBUFDS port map(
        I => data_volt_in_p,
        IB => data_volt_in_n,
        O => Diff_Input
    );

    inst_flop: FFD
        port map(
            clk => clk50,
            rst =>'0',
            ena => '1',
            D => Diff_Input,
            Q => process_in
        );

    inst_proces: Process_and_controll
        port map(
            process_in,
            clk50,
            hs,
            vs,
            red_o,
            grn_o,
            blu_o
        );

    data_volt_out <= process_in;

end;
```