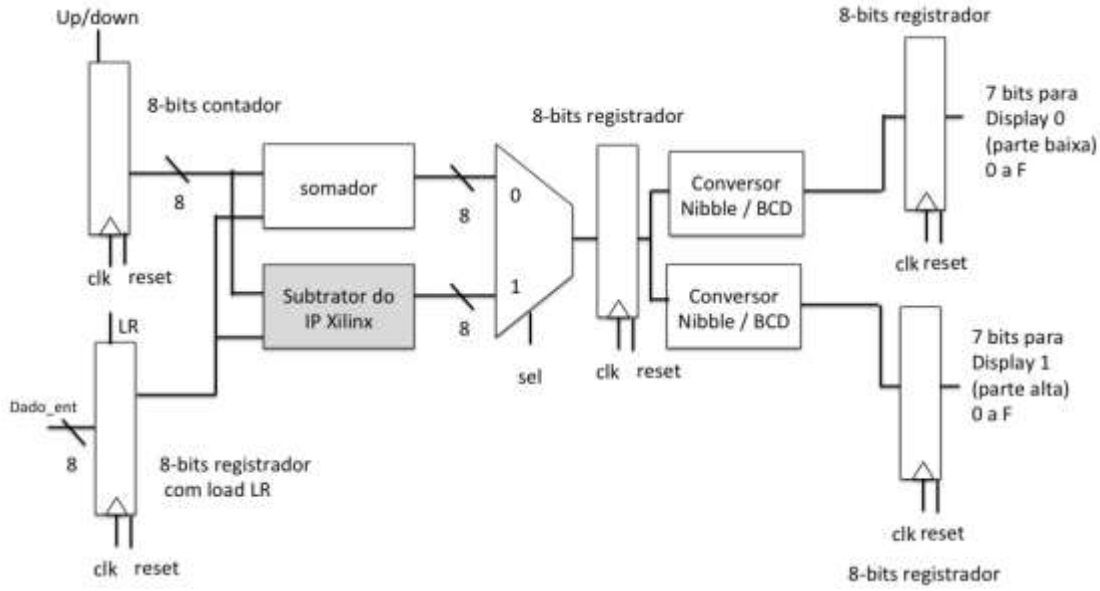
Trabalho 1 – Sistemas Digitais – 2020-2

Nome: Wellington Machado de Espindula
Matrícula:

1) Descreva o seguinte circuito digital em VHDL na ferramenta Vivado



No **testbench:**

- o dado de entrada (dado_ent) deve ser os **dois dígitos menos significativos** da matrícula descritos em binário (Exemplo: 0022134 seria 34 em binário).

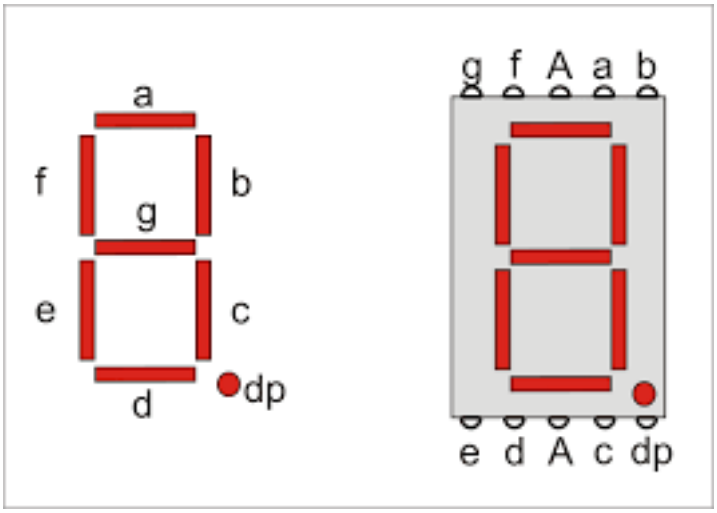
No **VHDL**:

- Registrador contador é inicializado com os **dois dígitos mais significativos** da matrícula descritos em binário (Exemplo: **00**22134 seria 00 em binário), quando o reset='1', e pode contar para cima e para baixo (Up/Down).

- Os demais registradores devem ser inicializados em ZERO quando (reset='1').

Conversor:

- Conversor de Nibble para 7seg é uma grande tabela verdade onde cada entrada de 4 bits define os leds do display 7 segmentos que deve acender para desenhar o número em hexadecimal correspondente. O led é ligado com o valor lógico 1.



Nibble					7seg							
#	N3	N2	N1	N0	#	a	b	c	d	e	f	g
0	0	0	0	0	7E	1	1	1	1	1	1	0
1	0	0	0	1	30	0	0	0	0	1	1	0
2	0	0	1	0	6D	1	1	0	1	1	0	1
3	0	0	1	1	79	1	1	1	1	0	0	1
4	0	1	0	0	33	0	1	1	0	0	1	1
5	0	1	0	1	5B	1	0	1	1	0	1	1
6	0	1	1	0	5F	1	1	0	1	1	1	1
7	0	1	1	1	70	1	1	1	0	0	0	0
8	1	0	0	0	7F	1	1	1	1	1	1	1
9	1	0	0	1	7B	1	1	1	1	0	1	1
A	1	0	1	0	77	1	1	1	0	1	1	1
b	1	0	1	1	1F	0	0	1	1	1	1	1
C	1	1	0	0	4E	1	0	0	1	1	1	0
d	1	1	0	1	3D	0	1	1	1	1	0	1
E	1	1	1	0	4F	1	0	0	1	1	1	1
F	1	1	1	1	47	1	0	0	0	1	1	1

Cole o VHDL completo aqui: **Nas próximas páginas constarão os códigos em VHDL de cada um dos componentes criados (reg, adder8bits, counter8bits). Os comentários indicam o nome do arquivo do componente. No circuito principal, existem saídas (cujos nomes iniciam por "dbg_") a mais do que constam no circuito descrito no enunciado, entretanto optei por mantê-los para facilitar no entendimento das waveforms e também na depuração.**

```vhdl
-- mainCircuit.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity mainCircuit is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           updown : in STD_LOGIC;
           dado_ent : in STD_LOGIC_VECTOR(7 downto 0);
           lr : in STD_LOGIC;
           sel : in STD_LOGIC;
           display0 : out STD_LOGIC_VECTOR (6 downto 0);
           display1 : out STD_LOGIC_VECTOR (6 downto 0);

           -- debug signals
           dbg_counter : out STD_LOGIC_VECTOR(7 downto 0);
           dbg_regin : out STD_LOGIC_VECTOR(7 downto 0);
           dbg_adder : out STD_LOGIC_VECTOR(7 downto 0);
           dbg_subtractor : out STD_LOGIC_VECTOR(7 downto 0);

           dbg_output_mux : out STD_LOGIC_VECTOR(7 downto 0);
           dbg_regmux_lsb : out STD_LOGIC_VECTOR(3 downto 0);
           dbg_regmux_msb : out STD_LOGIC_VECTOR(3 downto 0)
           );
end mainCircuit;

architecture Behavioral of mainCircuit is
component counter8bits is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           up_down : in STD_LOGIC;
           saida : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component reg is
    generic (n: integer);
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           lr : in STD_LOGIC;
```

```
            dado_ent : in STD_LOGIC_VECTOR (n-1 downto 0);
            saida : out STD_LOGIC_VECTOR (n-1 downto 0));
end component;

component adder8bits is
    Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           C : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component subtractor8bits
  PORT (
    A : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    B : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    S : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END component;

component bcd is
  Port (
    bin: in STD_LOGIC_VECTOR(3 downto 0);
    dec: out STD_LOGIC_VECTOR(6 downto 0)
  );
end component;


signal output_counter : STD_LOGIC_VECTOR(7 downto 0);
signal output_regin : STD_LOGIC_VECTOR(7 downto 0);

signal output_adder : STD_LOGIC_VECTOR(7 downto 0);
signal output_subtractor : STD_LOGIC_VECTOR(7 downto 0);

signal output_mux : STD_LOGIC_VECTOR(7 downto 0);

signal output_regmux : STD_LOGIC_VECTOR(7 downto 0);
signal output_regmux_msb : STD_LOGIC_VECTOR(3 downto 0);
signal output_regmux_lsb : STD_LOGIC_VECTOR(3 downto 0);

signal outuput_bcd_display0 : STD_LOGIC_VECTOR(6 downto 0);
signal outuput_bcd_display1 : STD_LOGIC_VECTOR(6 downto 0);

signal outuput_bcd_display0_8bits : STD_LOGIC_VECTOR(7 downto
```

```
0);
signal outuput_bcd_display1_8bits : STD_LOGIC_VECTOR(7 downto
0);

signal out_reg_display0 : STD_LOGIC_VECTOR(7 downto 0);
signal out_reg_display1 : STD_LOGIC_VECTOR(7 downto 0);

begin

counter : counter8bits
    PORT MAP (clk => clock, reset => reset, up_down =>
updown, saida => output_counter);

reg_in : reg
    GENERIC MAP (n => 8)
    PORT MAP (clk => clock, reset => reset, lr => lr,
dado_ent => dado_ent, saida => output_regin);

adder : adder8bits
    PORT MAP (A=>output_counter, B=>output_regin,
C=>output_adder);

subtractor : subtractor8bits
  PORT MAP (A => output_counter,B => output_regin,S
=>output_subtractor);

output_mux <= output_adder when sel = '0' else
              output_subtractor when sel = '1' else
              "00000000";

reg_mux : reg
    GENERIC MAP (n=>8)
    PORT MAP (clk => clock, reset => reset, lr => '1',
dado_ent => output_mux, saida => output_regmux);

output_regmux_lsb <= output_regmux(3 downto 0);
output_regmux_msb <= output_regmux(7 downto 4);

bcd_display0 : bcd
    PORT MAP (bin=>output_regmux_lsb,
dec=>outuput_bcd_display0);
```

```
bcd_display1 : bcd
    PORT MAP (bin=>output_regmux_msb,
dec=>outuput_bcd_display1);

outuput_bcd_display0_8bits <= '0' & outuput_bcd_display0;
outuput_bcd_display1_8bits <= '0' & outuput_bcd_display1;

reg_bcd_display0 : reg
    GENERIC MAP (n=>8)
    PORT MAP (clk => clock, reset => reset, lr => '1',
        dado_ent => outuput_bcd_display0_8bits, saida =>
out_reg_display0);

reg_bcd_display1 : reg
    GENERIC MAP (n=>8)
    PORT MAP (clk => clock, reset => reset, lr => '1',
        dado_ent => outuput_bcd_display1_8bits, saida =>
out_reg_display1);

display0 <= out_reg_display0(6 downto 0);
display1 <= out_reg_display1(6 downto 0);

-- Debugging signals
dbg_counter <= output_counter;
dbg_regin <= output_regin;

dbg_adder <= output_adder;
dbg_subtractor <= output_subtractor;

dbg_output_mux <= output_mux;
dbg_regmux_lsb <= output_regmux_lsb;
dbg_regmux_msb <= output_regmux_msb;

end Behavioral;
```

```vhdl
-- counter8bits.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity counter8bits is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           up_down : in STD_LOGIC;
           saida : out STD_LOGIC_VECTOR (7 downto 0));
end counter8bits;

architecture Behavioral of counter8bits is

signal count : std_logic_vector(7 downto 0);
constant init_const : std_logic_vector(7 downto 0) :=
"00000000"; -- 00 sao os 2 nums mais significativos da
matricula

begin

process(clk, reset) begin
if (reset='0') then
   count <= init_const;
elsif rising_edge(clk) then
    CASE up_down is
        when '0' => count <=
std_logic_vector(unsigned(count)-1);
        when '1' => count <=
std_logic_vector(unsigned(count)+1);
        when others => count <= count;
    end case;
end if;
end process;

saida <= count;
end Behavioral;
```

```vhdl
-- reg.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity reg is
    generic(n: natural);
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           lr : in STD_LOGIC;
           dado_ent : in STD_LOGIC_VECTOR (n-1 downto 0);
           saida : out STD_LOGIC_VECTOR (n-1 downto 0));
end reg;

architecture Behavioral of reg is
signal output: STD_LOGIC_VECTOR (n-1 downto 0);
begin
process(clk, reset, lr, dado_ent) begin
    if (reset = '0') then
        output <= (others=>'0');
    elsif rising_edge(clk) then
        if (lr='1') then
            output <= dado_ent;
        end if;
    end if;
end process;

saida <= output;
end Behavioral;
```

```
-- adder8bits.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity adder8bits is
    Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           C : out STD_LOGIC_VECTOR (7 downto 0));
end adder8bits;

architecture Behavioral of adder8bits is
begin
    C <= STD_LOGIC_VECTOR(SIGNED(A) + SIGNED(B));
end Behavioral;
```

```vhdl
-- bcd.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity bcd is
  Port (
    bin: in STD_LOGIC_VECTOR(3 downto 0);
    dec: out STD_LOGIC_VECTOR(6 downto 0)
  );
end bcd;

architecture Behavioral of bcd is
begin
    process(bin)
    begin
        case bin is
                                    -- abcdefg
            when "0000" => dec <= "1111110";   -- #0 off G
              when "0001" =>     dec <= "0000110";   -- #1 on
e,f
              when "0010" =>     dec <= "1101101";   -- #2
off c,f
              when "0011" =>     dec <= "1111001";   -- #3
off e,f
              when "0100" =>     dec <= "0110011";   -- #4
off a,d,e
              when "0101" =>     dec <= "1011011";   -- #5
off b,e
              when "0110" =>     dec <= "1011111";   -- #6
off b
              when "0111" =>     dec <= "1110000";   -- #7 on
a,b,c
              when "1000" =>     dec <= "1111111";   -- #8 on
all
              when "1001" =>     dec <= "1111011";   -- #9
off e
              when "1010" =>     dec <= "1110111";   --
#10(A) off d
              when "1011" =>     dec <= "0011111";   --
#11(b) off a,b
```

```
              when "1100" =>      dec <= "1001110";    --
#12(C) off b,c,g
              when "1101" =>      dec <= "0111101";    --
#13(D) off a,f
              when "1110" =>      dec <= "1001111";    --
#14(E) off b,c
              when "1111" =>      dec <= "1000111";    --
#15(F) off b,c,d
              when others =>      dec <= "0000000";    --
otherwise, turns everything off
      end case;
    end process;
end Behavioral;
```

2) Sintetize no VIVADO o VHDL circuito e informe a frequência e a utilização dos recursos

FPGA utilizado: **xc7a12ti** (Part: **xc7a12ticsg325-1L**)
Número de LUTs: **29**
Número de flip-flops: **38**

Quantos ciclos de relógio demora para aparecer o número correto na saída do circuito quando o contador é atualizado? **3**

3) Descreva um testbench que estimule alguns casos de operação do circuito, por exemplo que o contador conte para cima e para baixo e que haja somas e subtrações. A representação dos números é em complemento de 2.

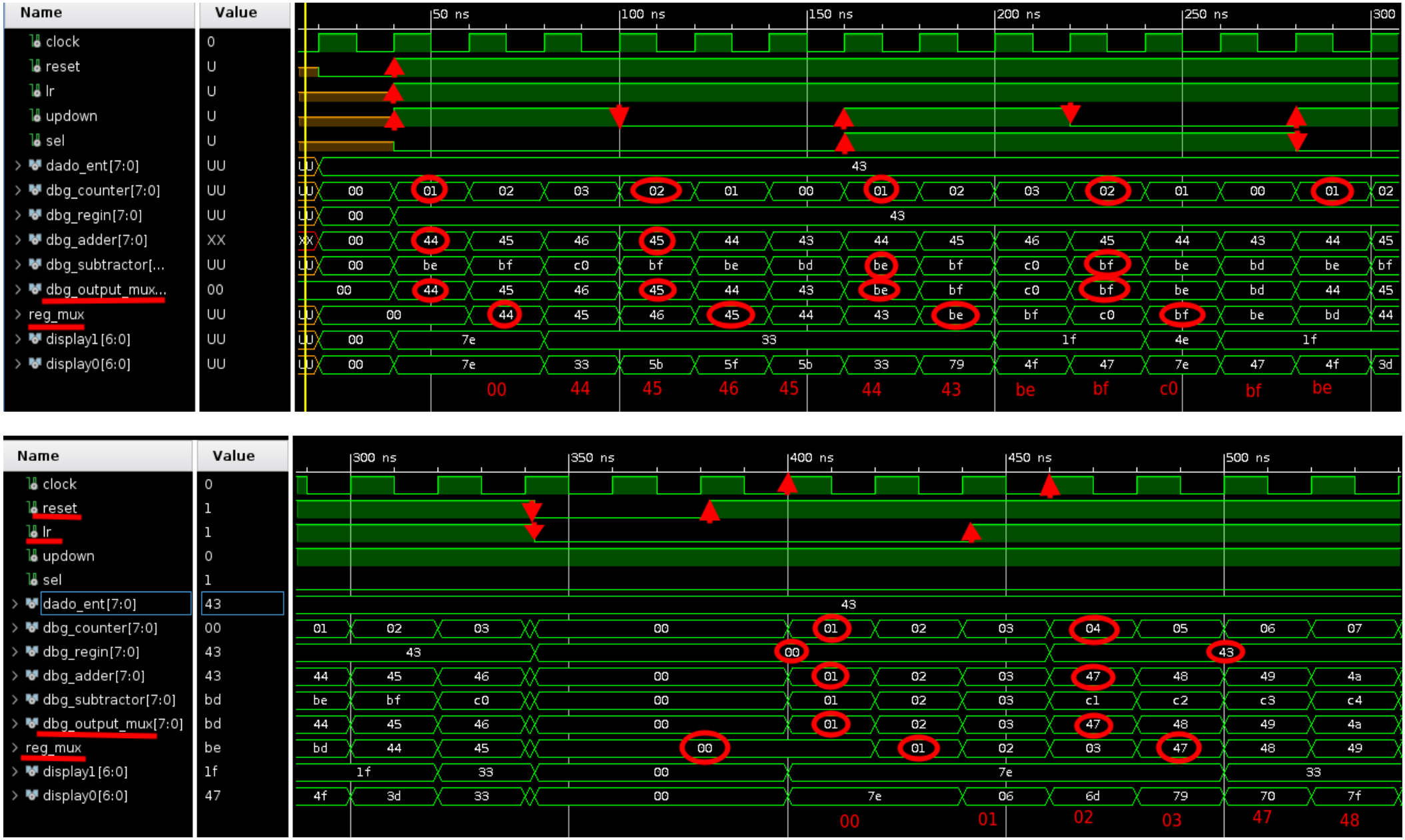
Copie e cole aqui o Testbench:

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

entity tb_mainCircuit is
--  Port ( );
end tb_mainCircuit;

architecture Behavioral of tb_mainCircuit is

component mainCircuit is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           updown : in STD_LOGIC;
           dado_ent : in STD_LOGIC_VECTOR(7 downto 0);
           lr : in STD_LOGIC;
           sel : in STD_LOGIC;
           display0 : out STD_LOGIC_VECTOR (6 downto 0);
           display1 : out STD_LOGIC_VECTOR (6 downto 0);

           -- debug signals
           dbg_counter : out STD_LOGIC_VECTOR(7 downto 0);
           dbg_regin : out STD_LOGIC_VECTOR(7 downto 0);
           dbg_adder : out STD_LOGIC_VECTOR(7 downto 0);
           dbg_subtractor : out STD_LOGIC_VECTOR(7 downto 0);

           dbg_output_mux : out STD_LOGIC_VECTOR(7 downto 0);
           dbg_regmux_lsb : out STD_LOGIC_VECTOR(3 downto 0);
           dbg_regmux_msb : out STD_LOGIC_VECTOR(3 downto 0)
          );
end component;

signal clock, reset, updown, lr, sel: STD_LOGIC;
signal dado_ent : STD_LOGIC_VECTOR(7 downto 0);
signal display0, display1: STD_LOGIC_VECTOR (6 downto 0);

signal dbg_counter, dbg_regin, dbg_adder, dbg_subtractor,
dbg_output_mux : STD_LOGIC_VECTOR(7 downto 0);
signal dbg_regmux_lsb, dbg_regmux_msb : STD_LOGIC_VECTOR(3
downto 0);

constant clk_period : time := 20ns;
```

```
-- num de matricula 00302367
-- 67 em decimal => 43 em hex, 01000011 em binario
constant lsn_matricula : std_logic_vector(7 downto 0) :=
"01000011";

begin

mainCircuit1 : mainCircuit
    PORT MAP (clock=>clock, reset=>reset, updown=>updown,
dado_ent=>dado_ent, lr=>lr,
        sel=>sel, display0=>display0, display1=>display1,
        dbg_counter=>dbg_counter,
        dbg_regin=>dbg_regin,
        dbg_adder=>dbg_adder,
        dbg_subtractor=>dbg_subtractor,
        dbg_output_mux=>dbg_output_mux,
        dbg_regmux_lsb=>dbg_regmux_lsb,
        dbg_regmux_msb=>dbg_regmux_msb);

process begin
    clock <= '1';
    wait for clk_period/2;
    clock <= '0';
    wait for clk_period/2;
end process;

process begin
    wait for clk_period;
    reset <= '0';
    dado_ent <= lsn_matricula;
    wait for clk_period;
    reset <= '1';
    lr <= '1';
    updown <= '1';
    sel <= '0';
    wait for 3*clk_period;
    updown <= '0';
    sel <= '0';
    wait for 3*clk_period;
    updown <= '1';
    sel <= '1';
```

```
    wait for 3*clk_period;
    updown <= '0';
    sel <= '1';
    wait for 3*clk_period;
    updown <= '1';
    sel <= '0';
    wait for 3.1*clk_period;
    reset <= '0';
    lr <= '0';
    wait for 2*clk_period;
    reset <= '1';
    wait for 3*clk_period;
    lr <= '1';
    updown <= '1';
    wait;
end process;

end Behavioral;
```

Copie e cole aqui o gráfico das simulações com ZOOM para poder observar os resultados.


Name
Value
clock
reset
lr
updown
sel
dado_ent[7:0]
dbg_counter[7:0]
dbg_regin[7:0]
dbg_adder[7:0]
dbg_subtractor[...
dbg_output_mux...
reg_mux
display1[6:0]
display0[6:0]
50 ns
100 ns
150 ns
200 ns
250 ns
300
00 44 45 46 45 44 43 be bf c0 bf be
Name
Value
clock
reset
lr
updown
sel
dado_ent[7:0]
dbg_counter[7:0]
dbg_regin[7:0]
dbg_adder[7:0]
dbg_subtractor[7:0]
dbg_output_mux[7:0]
reg_mux
display1[6:0]
display0[6:0]
300 ns
350 ns
400 ns
450 ns
500 ns
00 01 02 03 47 48

Comente os resultados:

No primeiro gráfico, tentei buscar verificar se as operações up-down, de adição e subtração e de seleção estavam funcionando. Como pode-se verificar, tudo está conforme previsto, levando três bordas de subida de relógio para que mudanças em up-down, seleção e nas entradas chegassem ao display. Dessa forma, testei as combinações de *up-down* 0 e 1 com a *sel* 0 e 1. Em vermelho abaixo, pus a "decodificação" dos sinais do *display* para hexadecimal. Todos os sinais estão usando o RADIX em hexadecimal. Vale ressaltar que utilizei up = 1 e down = 0.

No segundo, busquei verificar se o reset (assíncrono) e o load estavam funcionando corretamente. Como pode-se observar, assim que o reset é acionado - fora da subida de clock -, os registradores *counter* e de *regin* são zerados instantâneamente. Assim que o reset volta a 1, na próxima subida clock, o contador (*counter*) volta a contar. Depois, quando o load volta a 1, após a próxima subida de clock, o registrador de entrada de dados (*regin*) volta a armazenar e a adição e a subtração voltam a levar em consideração o valor deste. Dessa forma, pude concluir que os resultados estão em conformidade ao esperado.