LABORATÓRIO 01 REVISÃO LÓGICA SEQUENCIAL

Arthur Faria Campos 16/0024242

Programa de Engenharia Eletrônica
Faculdade Gama - Universidade de Brasília
St. Leste Projeção A - Gama Leste, Brasília - DF, 72444-240
email: arthur-fc@hotmail.com

Felipe Lima Alcântara 16/0027918

Programa de Engenharia Eletrônica
Faculdade Gama - Universidade de Brasília
St. Leste Projeção A - Gama Leste, Brasília - DF, 72444-240
email: lipelima0327@gmail.com

RESUMO

O documento apresenta o relatório técnico do primeiro experimento, da matéria Prática de Eletrônica Digital 2. Este experimento denominado "Revisão Lógica Sequencial" busca revisar, conceitos já vistos na matéria anterior e suas implementações em VHDL.
Como um contador Up /Down e também implementar um modulador de largura de pulso PWM.

1. INTRODUÇÃO

A atividade 1 buscou por meio de 2 exercícios e 1 desafio, revisar a utilização de lógica sequencial em códigos VHDL.

O primeiro exercício se baseia na implementação do CI 74LS193 na FPGA, este CI nada mais é do que um contador binário de quatro bits. Este CI tem opção de contagem progressiva e regressiva.

Quando a contagem progressiva ultrapassa o valor "1111", a saída CARRY do CI vai para nível lógico alto, da mesma forma quando a contagem regressiva vai para números negativos, após "0000", a saída BORROW vai para nível lógico alto.

O segundo exercício é a implementação de um modulador de largura de pulso, PWM, onde a largura deveria ser escolhida por quatro swicthes, a saída do PWM foi mapeada por um LED.

O PWM tem muita importância por se tratar de um regulador de pulso, sendo utilizado para controle de varios aparelhos eletrônicos. Como por exemplo controle de velocidade de motores.

2. EXPERIMENTO

2.1. Exercício 01

A implementação do CI 74LS193 foi feita em VHDL, com o sinal de clock de 1 Hz, sendo assim necessário a criação de um divisor de clock.

O código da Figura 3 descreve o funcionamento do CI, o contador é feito com flip-flops, porém a saída do componente, receberá a variável que está recebendo o implemento de contagem. Além disso o contador contará de "0000" a "1111", invertendo o início e o fim na descida. A saída do contador será ligada diretamente ao leds, eles serão acesos caso a recebam nível lógico alto.

A forma de Contagem do CI pode ser representada pelo Fluxograma da Figura 1.

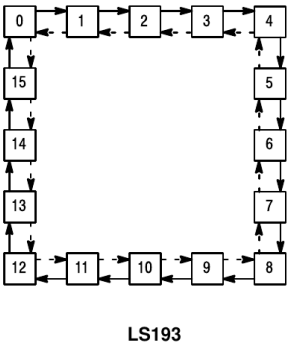

Figura 1: Contagem

O código da Figura 2 mostra o Top Module do CI onde são declarados, os componentes e as conexões de entrada e saídas que serão ligadas a FPGA Basys 3.

COUNT UP ———
COUNT DOWN - - - - - -

Foi inserida dois botões, para as entradas que controlariam a contagem de subida e outra que controla a descida, não sendo admitido que as duas entradas sejam ‘1’ simultaneamente. Isso pode ser verificado no Process Butons (Linha 54), na qual da prioridade para um dos botões caso os dois sejam pressionados ao mesmo tempo.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_arith.ALL;
4  use IEEE.STD_LOGIC_unsigned.ALL;
5
6
7  entity Top_Module_74LS193 is
8      Port (  Data : in STD_LOGIC_VECTOR(3 downto 0);
9              Clear : in STD_LOGIC;
10             Load : in STD_LOGIC;
11             Count_Up : in STD_LOGIC;
12             Count_Down : in STD_LOGIC;
13             Clk : in STD_LOGIC;
14             led : out STD_LOGIC_VECTOR(3 downto 0); -- Outputs
15             Borrow : out STD_LOGIC;
16             Carry : out STD_LOGIC);
17 end Top_Module_74LS193;
18
19 architecture Behavioral of Top_Module_74LS193 is
20
21 --Divisor de Clock ----------------------------------------------
22 Component Clock_Divider is
23     Generic(Preset : INTEGER);
24     Port (  Clk_in : in STD_LOGIC;
25             Reset : in STD_LOGIC;
26             Clk_out : out STD_LOGIC);
27 end Component;
28
29 -- Sinais Utilizados no Divisor de Clock
30 signal sClk_1Hz: STD_LOGIC := '0';
31 ----------------------------------------------------------------
32 --Contador Up / Down --------------------------------------------
33 Component Count_Up_Down is
34     Port (  Reset : in STD_LOGIC;
35             Data : in STD_LOGIC_VECTOR(3 downto 0);
36             Load : in STD_LOGIC;
37             Up_Down : in STD_LOGIC;
38             Clk : in STD_LOGIC;
39             Borrow: out STD_LOGIC;
40             Carry: out STD_LOGIC;
41             Num : out STD_LOGIC_VECTOR(3 downto 0));
42 end Component;
43
44 -- Sinais Utilizados no Contador
45 signal sSel: STD_LOGIC := '0';
46 ----------------------------------------------------------------
47 begin
48
49     Clock_1Hz: Clock_Divider generic map (Preset => 10)
50                              port map ( Clk_in => Clk,
51                                         Reset => Clear,
52                                         Clk_out => sClk_1Hz);
53
54     Butons: process (Count_Up,Count_Down)
55         begin
56             if(Count_up = '1') then
57                 sSel <= '0';
58             elsif(Count_Down = '1') then
59                 sSel <= '1';
60             end if;
61         end process;
62
63     Count: Count_Up_Down port map ( Reset => Clear,
64                                     Data => Data,
65                                     Load => Load,
66                                     Up_Down => sSel,
67                                     Clk => sClk_1Hz,
68                                     Borrow => Borrow,
69                                     Carry => Carry,
70                                     Num => led);
71 end Behavioral;
```

Figura 2: Top Module CI 74LS193

Na figura 3 temos o Codigo do Funcionamento do CI, sendo um contador Up/Down de 4 bits com Carry e Borrow.
Além disso foi necessário inserir um botão de reset, para resetar a contagem e um de load para caregar a entrada inserida nos Switches(Data), a contagem será demonstrada por leds e o carry e borrow também.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_arith.ALL;
4  use IEEE.STD_LOGIC_unsigned.ALL;
5
6
7  entity Count_Up_Down is
8      Port (  Reset : in STD_LOGIC;
9              Data : in STD_LOGIC_VECTOR(3 downto 0);
10             Load : in STD_LOGIC;
11             Up_Down : in STD_LOGIC;
12             Clk : in STD_LOGIC;
13             Borrow: out STD_LOGIC;
14             Carry: out STD_LOGIC;
15             Num : out STD_LOGIC_VECTOR(3 downto 0));
16 end Count_Up_Down;
17
18 architecture Behavioral of Count_Up_Down is
19
20     signal sCount : std_logic_vector(3 downto 0) := "0000";
21     signal sBorrow: STD_LOGIC:='0';
22     signal sCarry: STD_LOGIC:='0';
23
24
25 begin
26     Num <= sCount;
27     Borrow <= sBorrow;
28     Carry <= sCarry;
29     process(Clk,Reset,Load)
30         begin
31             if rising_edge(Clk) then
32                 sCarry <= '0';
33                 sBorrow <= '0';
34                 if reset = '1' then
35                     sCount<="0000";
36                 elsif Load = '1' then
37                     sCount <= Data;
38                 elsif Up_Down='0' then
39                     if(sCount = "1111") then
40                         sCarry <= '1';
41                     end if;
42                     sCount <= sCount + '1';
43                 elsif Up_Down='1' then
44                     if(sCount = "0000") then
45                         sBorrow <= '1';
46                     end if;
47                     sCount<=sCount - '1';
48                 end if;
49             end if;
50         end process;
51 end Behavioral;
```

Figura 3: Contador Up / Down

O divisor de clock utiliza flip-flops, quanto mais maior será a divisão da frequência. Para a implementação em VHDL é utilizada a equação (1), onde o resultado será o número de parada da contagem que é realizada em um Process, resetado quando chega ao número estabelecido pela equação 1, e invertendo o sinal de saida.

$$Num = \frac{\frac{Fplaca}{Fdesejada}}{2} - 1 \qquad (1)$$

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

entity Clock_Divider is
    Generic (Preset : INTEGER);
    Port (  Clk_in : in STD_LOGIC;
            Reset : in STD_LOGIC;
            Clk_out : out STD_LOGIC);
end Clock_Divider;

architecture Behavioral of Clock_Divider is

    signal sClk : STD_LOGIC := '0';
    signal sCount : INTEGER := 0;

begin

    Clk_out <= sClk;

    Clock_Divider: process(Clk_in,Reset)
        begin
            if rising_edge(Clk_in) then
                if Reset = '1' then
                    sCount <= 0;
                    sClk <= '0';
                elsif sCount = Preset then
                    sClk <= not sClk;
                    sCount <= 0;
                else
                    sCount <= sCount + 1;
                end if;
            end if;
        end process;
end Behavioral;
```

Figura 4: Divisor de Clock

2.2. Exercício 02

Para realizar a implementação do PWM, foi necessário inserir quatro chaves de entrada, para a escolha da largura de pulso e um LED para mapear a saída.

O PWM utiliza-se do Duty Cycle, que nada mais é do que a porcentagem da onda que estará ativa. Ele receberá um valor a partir do que for inserido nas chaves, cada valor representará uma porcentagem, descrita na tabela (1).

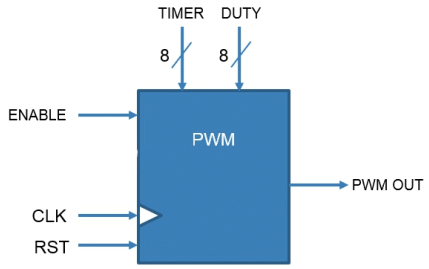

Figura 5: Component PWM

Tabela 1: Duty Cycle

DUTY	$\frac{n^o\ Pulsos}{n^o\ Total\ de\ Pulsos}$	DUTY%
"0000"	1/16	6.25%
"0001"	2/16	12.50%
"0010"	3/16	18.75%
"0011"	4/16	25.00%
"0100"	5/16	31.25%
"0101"	6/16	37.50%
"0110"	7/16	43.75%
"0111"	8/16	50.00%
"1000"	9/16	56.25%
"1001"	10/16	62.50%
"1010"	11/16	68.75%
"1011"	12/16	75.00%
"1100"	13/16	81.25%
"1101"	14/16	87.50%
"1110"	15/16	93.75%
"1111"	16/16	100%

No Top Module, Figura 7, o clock utilizado foi o da placa Basys3, este de 100MHz, pois com este clock a visualização do PWM se torna mais fácil, pois só será visto um aumento ou perda da luminosidade do led.

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



entity Top_Module_PWM is
    Port (  Clk : in STD_LOGIC;
            swTimer : in STD_LOGIC_VECTOR(3 downto 0);
            swDuty : in STD_LOGIC_VECTOR(3 downto 0);
            swEnable : in STD_LOGIC;
            swReset : in STD_LOGIC;
            Led : out STD_LOGIC);
end Top_Module_PWM;

architecture Behavioral of Top_Module_PWM is

--Component de PWM ----------------------------------
Component PWM is
    Port (  Timer : in STD_LOGIC_VECTOR(3 downto 0);
            Duty : in STD_LOGIC_VECTOR(3 downto 0);
            Enable : in STD_LOGIC;
            Clk : in STD_LOGIC;
            Reset : in STD_LOGIC;
            PWM_out : out STD_LOGIC);
end Component;
-----------------------------------------------------
begin
    PWM_Module : PWM port map ( Timer => swTimer,
                                Duty => swDuty,
                                Enable => swEnable,
                                Clk => Clk,
                                Reset => swReset,
                                PWM_out => Led);

end Behavioral;
```

Figura 6: Top Module PWM

Para efetuar o controle da largura, foi realizado condicionais if comparando o Duty Cycle com o valor total do clock, que é uma entrada chamada TIMER que tem um funcionamento igual ao do Duty, se os dois valores forem iguais a Led permanecerá ligada.

Foi inserido um sinal denominado sCycle_off, o intuito deste sinal é saber até quando o contador deverá manter a saída em nível lógico baixo, quando o contador for igual ao valor de sCycle_off a onda irá para nível lógico alto.

E por fim se o contador for igual ao valor do TIMER, a onda irá para nível lógico baixo.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_arith.ALL;
4  use IEEE.STD_LOGIC_unsigned.ALL;
5
6
7  entity PWM is
8      Port (  Timer : in STD_LOGIC_VECTOR(3 downto 0);
9              Duty : in STD_LOGIC_VECTOR(3 downto 0);
10             Enable : in STD_LOGIC;
11             Clk : in STD_LOGIC;
12             Reset : in STD_LOGIC;
13             PWM_out : out STD_LOGIC);
14 end PWM;
15
16 architecture Behavioral of PWM is
17
18     signal sCycle_off : STD_LOGIC_VECTOR(3 downto 0);
19     signal sCount : STD_LOGIC_VECTOR(3 downto 0) := "0000";
20     signal sPWM_out : STD_LOGIC;
21
22 begin
23
24     sCycle_off <= Timer - Duty - '1';
25
26 -- Caso Especial
27
28     with Timer select
29         PWM_out <= Clk and Enable and (not Reset) when "0000",
30         sPWM_out when others;
31
32     PWM: process (Clk,Reset)
33         begin
34             if(Reset = '1') then
35                 sPWM_out <= '0';
36                 sCount <= "0000";
37             elsif rising_edge(Clk) then
38                 if (Enable = '1') then
39                     sCount <= sCount + '1';
40                     if (Duty >= Timer) then
41                         sPWM_out <= '1';
42                     elsif ( sCount = sCycle_off) then
43                         sPWM_out <= '1';
44                     elsif ( sCount = Timer) then
45                         sPWM_out <= '0';
46                         sCount <= "0000";
47                     end if;
48                 else
49                     sPWM_out <= '0';
50                     sCount <= "0000";
51                 end if;
52             end if;
53     end process;
54
55 end Behavioral;
```

Figura 7: Component PWM

3. RESULTADOS

Antes e Implementarmos na FPGA simulamos os códigos por meio de Test Benchs.

3.1. Simulações exercício 01

Na simulação abaixo (Figura 8) simulamos a contagem positiva do CI, sendo que quando alcançado o máximo do contador “1111” o sinal de Carry é ativado até a próxima contagem ou Reset do Contador.

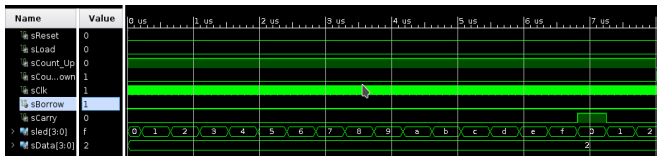

Figura 8: Simulação Contador Up

Ja na Figura 9 está a simulação da contagem negativa do CI, sendo que quando alcançado o mínimo do contador “0000” o sinal de Borrow é ativado até a próxima contagem ou Reset do Contador.

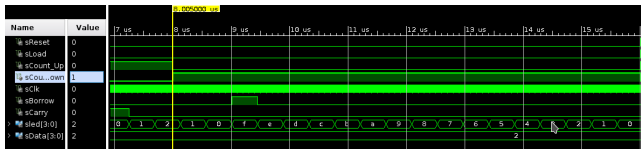

Figura 9: Simulação Contador Down

Na Figura 10 está a simulação do botão Load, que seta a contador para o número que estava Carregado no Data, continuando a contagem conforme a direção do Up/Down.

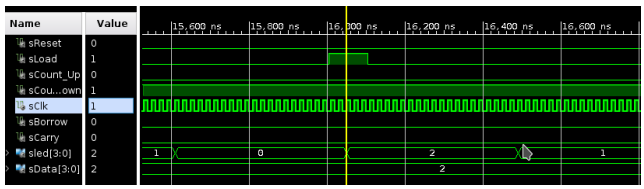

Figura 10: Simulação Load

Resource	Utilization	Available	Utilization %
LUT	57	20800	0.27
FF	38	41600	0.09
IO	12	106	11.32
BUFG	1	32	3.13

Tabela 2: Recursos Utilizados da FPGA no CI

3.2. Simulações Exercício 02

As simulações do exercício 02 apresentam a comprovação dos resultados esperados, no qual os ciclos vao aumentando de acordo com o valor no sinal do Duty, como podemos ver pela discrepância entre os sinais da Figura 11 e os sinais da Figura 14.

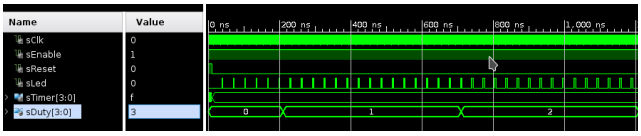

Figura 11: Simulação PWM (1/4)

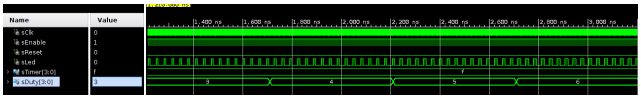

Figura 12: Simulação PWM (2/4)

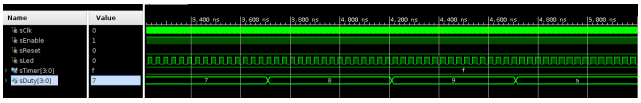

Figura 13: Simulação PWM (3/4)

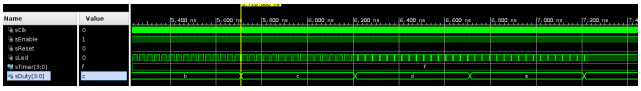

Figura 14: Simulação PWM (4/4)

A tabela abaixo mostra a utilizaçao dos componentes da Basys 3.

Resource	Utilization	Available	Utilization %
LUT	16	20800	0.08
FF	5	41600	0.01
IO	12	106	11.32
BUFG	1	32	3.13

Tabela 3: Componentes utilizados no PWM

4. DISCUSSÃO E CONCLUSÕES

Os experimentos foram bem-sucedidos e apresentaram os resultados esperados pela teoria, sendo comprovados tanto pelas simulações e pela implementação na FPGA Basys 3.

No experimento 01 observamos que quanto maior a frequência de clock utilizada, mais difícil se torna a visualização da contagem por isso é necessário reduzir o clock de entrada, para analisar visualmente na FPGA. Contudo para outras aplicações pode-se utilizar frequências mais altas.

Já no exercício 2, no início mantivemos uma frequência muito baixa, portanto o efeito que se estava esperando não era perceptível, ao deixar o clock com o valor original da placa 100MHz, foi possível verificar o efeito do PWM, alterando assim a intensidade do Led.

Também pode-se perceber que em nenhuma das duas atividades chegamos a utilizar uma quantidade considerável de componentes da Basys 3, como mostrado pelas tabelas 2 e 3.

Assim, concluímos o laboratório obtendo sucesso na revisão de conceitos já vistos na matéria anterior e suas implementações em VHDL.

5. REFERÊNCIAS

[1] Villanova, G. (2016). Uma arquitetura PWM em VHDL. [online] Embarcados. Disponivel em: https://www.embarcados.com.br/uma-arquitetura-pwm-em-vhdl/ [Acessado 7 Sep. 2017].

[2] Wakerly, John F., Digital Design: Principles and Practices, 4th ed., Prentice Hall, 2005.

[3] Chu, Pong P., FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version, Wiley, 2011. [EBRARY]

[4] D'Amore, R., VHDL: Descrição e Síntese de Circuitos Digitais, 2 a . edição, LTC, 2012