



**UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ**  
**INSTITUTO DE GEOCIÊNCIAS E ENGENHARIAS**  
**FACULDADE DE COMPUTAÇÃO E ENGENHARIA ELÉTRICA**  
**CURSO DE ENGENHARIA DA COMPUTAÇÃO**

**Relatório de Sistemas Embarcados**

**201840601017 – Iago Costa das Flores**

**201740601025 – Leyrisvan da Costa Nascimento**

**201840601031 – Warley Rabelo Galvão**

**Marabá – PA**

**2021**



**UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ**  
**INSTITUTO DE GEOCIÊNCIAS E ENGENHARIAS**  
**FACULDADE DE COMPUTAÇÃO E ENGENHARIA ELÉTRICA**  
**CURSO DE ENGENHARIA DA COMPUTAÇÃO**

**IAGO COSTA DAS FLORES**  
**LEYRISVAN DA COSTA NASCIMENTO**  
**WARLEY RABELO GALVÃO**

**TRABALHO FINAL 01**

Este relatório é um critério de atividade avaliativa da disciplina SISTEMAS EMBARCADOS ministrada pelo professor Dr. José Carlos da Silva.

**Marabá – PA**

**2021**

## Sumário

Objetivo.....	4
Atividade 01 .....	4
Atividade 02 .....	6
Atividade 03 .....	7
Atividade 04 .....	9
Atividade 05 .....	10
Conclusões.....	12
Referências .....	12

## Objetivo

- ✓ Realizar as atividades utilizando o simulador Quartus II, e assim espera-se:
- ✓ Implementar o código na linguagem de programação em VHDL;
- ✓ Absolver as funções do código fonte;
- ✓ Compilar com sucesso e simular os resultados obtidos.

## Atividade 01

Implementar o mux de 4 entradas.

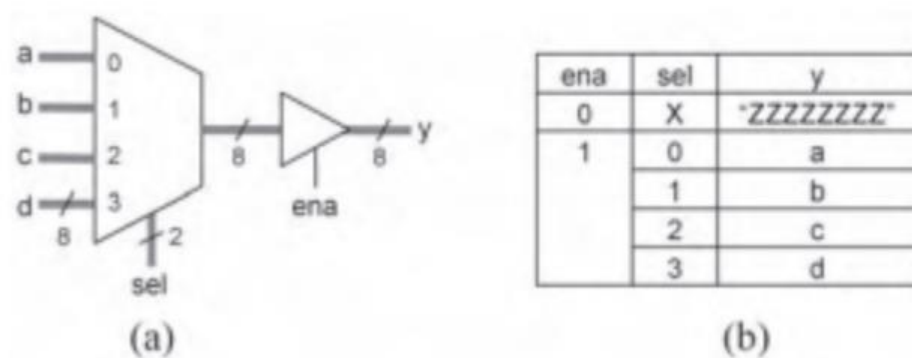


Figura (a): Representando a entidade e Figura (b) Representando a arquitetura.

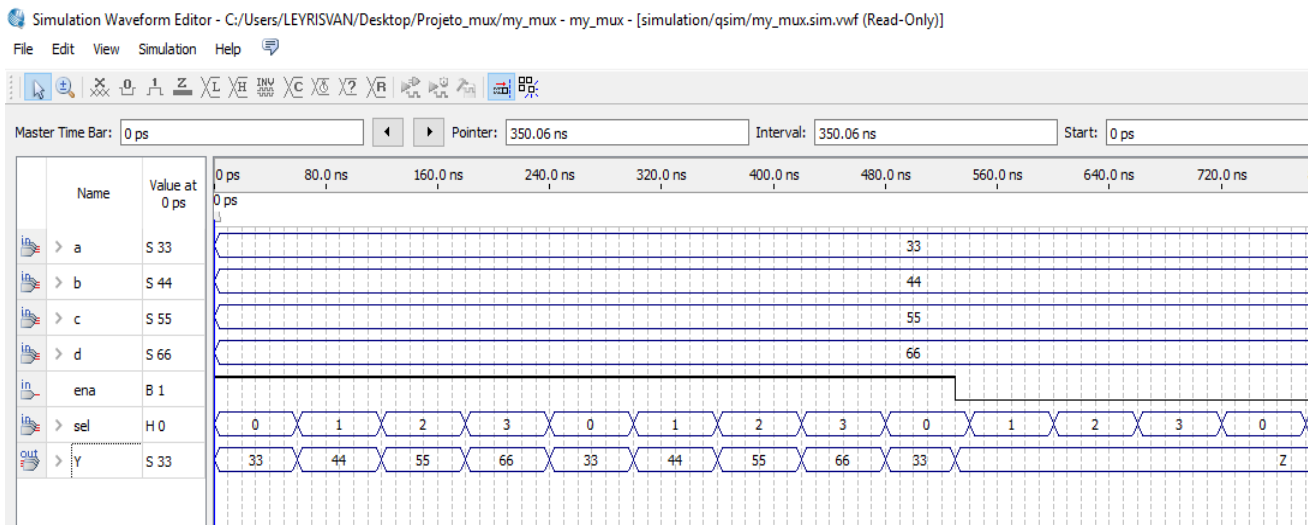
```
1  LIBRARY ieee; ----Faz a importação das bibliotecas ieee
2  USE ieee.std_logic_1164.all; --importa as funções de uso
3
4  ENTITY my_mux IS -- Declaração da minha entidade
5  port ( -- chamada da função port pra criação de portas
6    a,b,c,d: IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- declaração das variáveis de entrada
7    sel: IN NATURAL RANGE 0 TO 3; -- variavel de seleção de tamanho 5 (0 a 3)
8    ena: IN STD_LOGIC; ---variavel pra ativar o estado logico
9    Y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0) -- variavel de saída
10 ); --parentese de fechamento da função
11 END my_mux; --fim de my_mux da minha entidade
12
13 ARCHITECTURE HARDWARE OF my_mux IS -- arquitetura do hardware para meu mux
14 SIGNAL X: STD_LOGIC_VECTOR (7 DOWNTO 0); -- sinal que receberá o estado logico do vetor
15 BEGIN -- início de logica da arquitetura
16 X<=a WHEN sel=0 ELSE -- variavel sel=0 entao ele envia o valor de a para X
17   b WHEN sel=1 ELSE -- variavel sel=1 entao ele envia o valor de b para X
18   c WHEN sel=2 ELSE -- variavel sel=2 entao ele envia o valor de c para X
19   d ; -- senão sel3=3
20
21 Y<=X WHEN ena='1' ELSE -- condição da variavel de saída
22   (others => 'Z'); -- caso contrario entra z como impedancia
23 END HARDWARE ; -- finaliza hardware
```

Messages

Type	ID	Message
>		Running Quartus II 64-Bit EDA Netlist Writer
>		Command: quartus_eda --read_settings_files=off --write_settings_files=off my_mux -c my_mux
>	204019	Generated file my_mux.vo in folder "C:/Users/LEYRISVAN/Desktop/Projeto_mux/simulation/qsim/" for EDA simulation
>		Quartus II 64-Bit EDA Netlist Writer was successful. 0 errors, 0 warnings
>	293000	Quartus II Full Compilation was successful. 0 errors, 9 warnings

System (17) / Processing (117)

Figura1: Código fonte comentado de my\_mux



**Figura 2:** Simulação do código fonte do my\_mux.

**Resultado:** Foi implementado o código correspondente ao mux obedecendo os padrões de programação em VHDL no Quartus II como mostra a **Figura 1**: Pacotes de Biblioteca LIBRARY ieee e use.ieee.std\_logic\_1164.all onde serão armazenadas as informações compiladas, seguindo isso temos a entidade **ENTITY** com seu nome **my\_mux** e seus pinos de entrada e saída com suas devidas tamanhos de vetores indicado pelo **std\_logic\_vector** valor NATURE RANGE pra variável **sel** que irá receber valores pra chave de seleção de cada variável; terminado a entidade finaliza-se essa entidade **END my\_mux**..

Para sua arquitetura **ARCHITECTURE** com seu devido nome **HARDWARE** e seguido do nome **my\_mux** que foi atribuído a **ENTITY** especificou-se o funcionamento do circuito com suas declarações e comando tais como a condição que a variável **X** que é intermediária pra ativar valores com suas condições de seleção que as variáveis de entrada **a,b,c,d** irão receber e a medida que a chave esteja ativada no estado logico **1** a variável de saída irá receber valores determinado por uma determinada seleção de clock com os valores que forem atribuídos para a variáveis de entrada e caso o ativador seja desativado com estado logico **0** o valor de impedância será a letra **z** e assim finalizando a arquitetura com **END HARDWARE**.

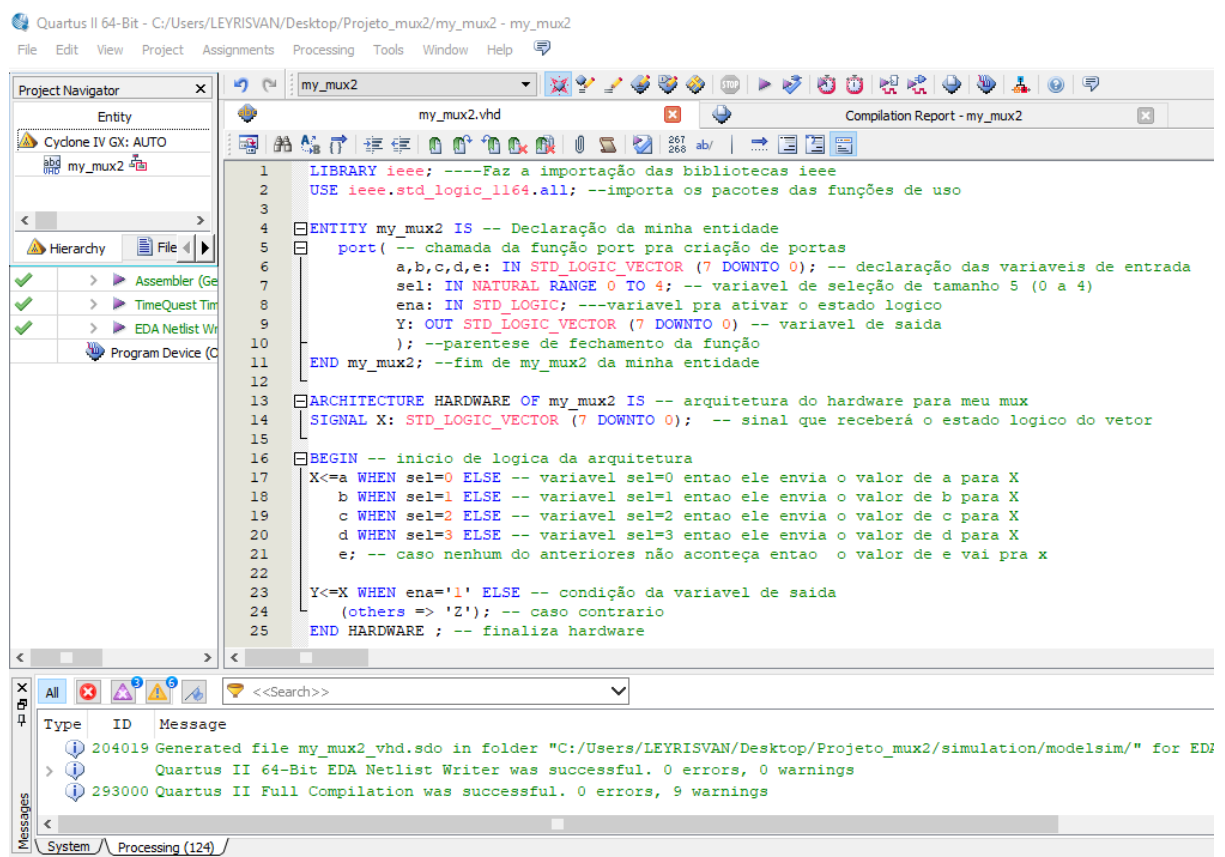
Após compilado foi feito uma simulação atribuindo valores decimais para as variáveis de entrada e selecionando um ponto no estado logico 1 para variável de ativação **ena**

e um intervalo de clock para variável **sel** para repetir os valores de entrada para ser representado a ainda **y** como mostra a **Figura 2**.

Com sucesso obtivemos resposta que corresponde com o código e assim o resultado teve êxito.

## Atividade 02

Implementar o código da atividade 01 acrescentando uma entrada “e”.

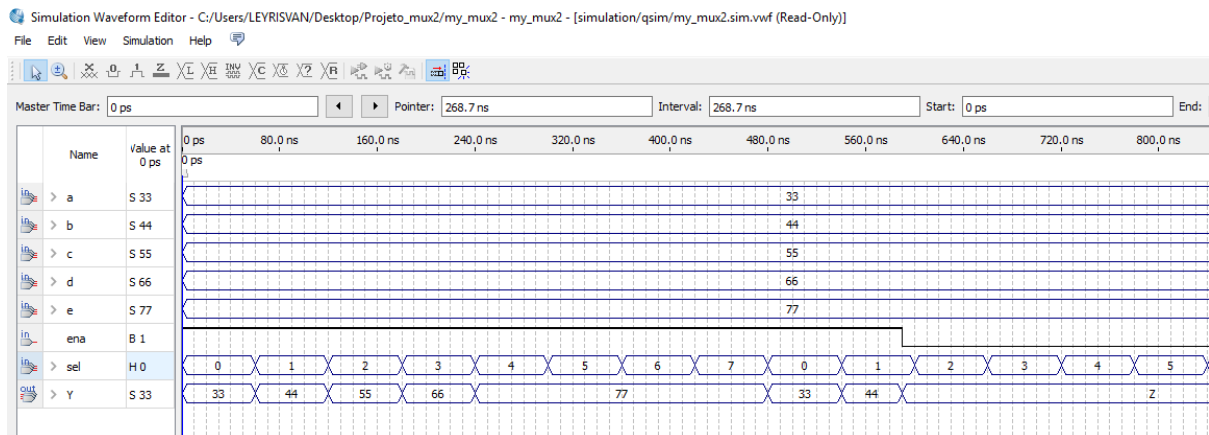


```
1 LIBRARY ieee; ----Faz a importação das bibliotecas ieee
2 USE ieee.std_logic_1164.all; --importa os pacotes das funções de uso
3
4 ENTITY my_mux2 IS -- Declaração da minha entidade
5 port( -- chamada da função port pra criação de portas
6     a,b,c,d,e: IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- declaração das variáveis de entrada
7     sel: IN NATURAL_RANGE 0 TO 4; -- variavel de seleção de tamanho 5 (0 a 4)
8     ena: IN STD_LOGIC; ---variavel pra ativar o estado logico
9     Y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0) -- variavel de saída
10 ); --parentese de fechamento da função
11 END my_mux2; --fim de my_mux2 da minha entidade
12
13 ARCHITECTURE HARDWARE OF my_mux2 IS -- arquitetura do hardware para meu mux
14 SIGNAL X: STD_LOGIC_VECTOR (7 DOWNTO 0); -- sinal que receberá o estado logico do vetor
15
16 BEGIN -- inicio de logica da arquitetura
17 X<=a WHEN sel=0 ELSE -- variavel sel=0 entao ele envia o valor de a para X
18   b WHEN sel=1 ELSE -- variavel sel=1 entao ele envia o valor de b para X
19   c WHEN sel=2 ELSE -- variavel sel=2 entao ele envia o valor de c para X
20   d WHEN sel=3 ELSE -- variavel sel=3 entao ele envia o valor de d para X
21   e; -- caso nenhum do anteriores não aconteça entao o valor de e vai pra x
22
23 Y<=X WHEN ena='1' ELSE -- condição da variavel de saída
24   (others => 'Z'); -- caso contrario
25 END HARDWARE ; -- finaliza hardware
```

Messages

Type	ID	Message
Information	204019	Generated file my_mux2_vhd.sdo in folder "C:/Users/LEYRISVAN/Desktop/Projeto_mux2/simulation/modelsim/" for EDI
Information		Quartus II 64-Bit EDA Netlist Writer was successful. 0 errors, 0 warnings
Information	293000	Quartus II Full Compilation was successful. 0 errors, 9 warnings

Figura 3: Código fonte da atividade 2.



**Figura 4:** Simulação do código fonte do my\_mux2.

**Resultado:** Seguindo o mesmo resultado que ocorreu no experimento anterior na Atividade 01, após adição de uma porta de entrada “e” é que para todas as variáveis anteriores os valores se mantiveram, mas porém para a porta de entrada **e**, o valor se expandiu para o restante do intervalo correspondente ao tamanho do vetor que foi definido pela variável **sel** e assim o seu valor 77 obteve o valor com clock até finalizar o vetor na posição 7 como mostra a **Figura 4**. Após isso o processo repete novamente. Defendendo o próprio código descreve o resultado pois na linha 21 **e**; mostra que para o restante da condição o valor será o que está na variável ‘e’ como mostra a **Figura 3**.

## Atividade 03

Implementar o código fonte para um somador de 4 bits.

```

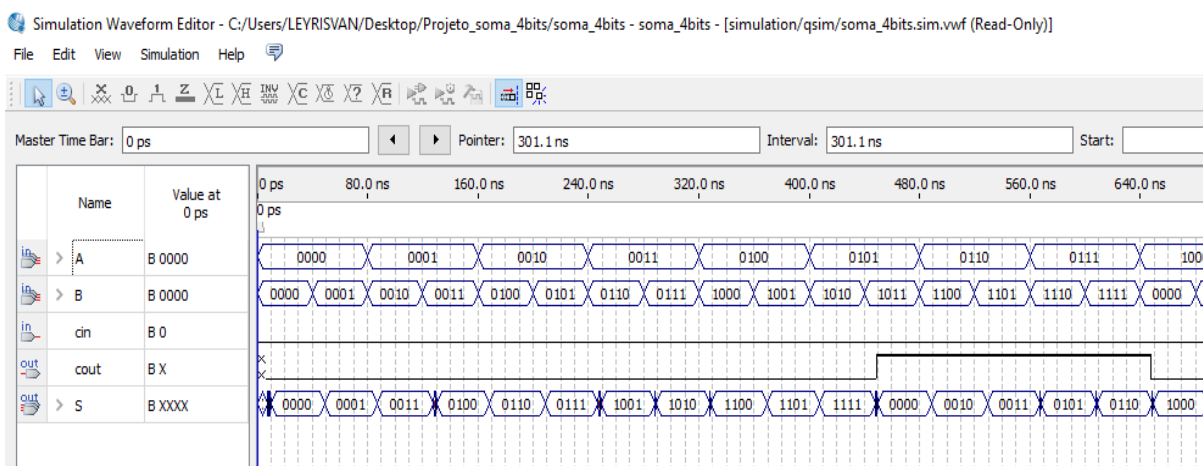
1  library ieee; -- Bibliotecas
2  use ieee.std_logic_1164.all; -- importa as funcoes de uso
3
4  entity soma_4bits is -- criação da entidade
5
6  port
7  (
8    A: in std_logic_vector(3 downto 0); -- variavel de entrada do valor A
9    B: in std_logic_vector(3 downto 0); -- variavel de entrada do valor B
10   cin: in std_logic; -- vai 1 de entrada
11   cout: out std_logic; -- vai 1 de saída da operação
12   S: out std_logic_vector(3 downto 0) -- variavel do valor se saída S
13 );
14 end soma_4bits; -- final da entidade
15
  
```

```

16 architecture somador of soma_4bits is
17 begin
18
19     process (A,B,cin) --processo para o meio somador
20     variable temp:std_logic_vector(3 downto 0); -- recebera os valores das somas de A e B
21     variable c:std_logic; -- fará a funcao do cin e do cout
22     begin
23         c := cin; -- atualiza o vai 1 de entrada para que as operacoes
24
25         --0-eracoes logicas para bits
26         temp(0) := A(0) xor B(0) xor c;
27         c := (A(0) and B(0)) or ((A(0) xor B(0)) and c);
28
29         temp(1) := A(1) xor B(1) xor c;
30         c := (A(1) and B(1)) or ((A(1) xor B(1)) and c);
31
32         temp(2) := A(2) xor B(2) xor c;
33         c := (A(2) and B(2)) or ((A(2) xor B(2)) and c);
34
35         temp(3) := A(3) xor B(3) xor c;
36         c := (A(3) and B(3)) or ((A(3) xor B(3)) and c);
37
38         cout <= c; -- variavel cout recebera o valor de c
39         S <= temp; -- variavel temp recebera o valor da saida S
40     end process; -- final do processo
41 end somador; -- final do somador

```

**Figura 5:** Figura do código fonte de um somador de 4 bits.



**Figura 6:** Figura de simulação do código fonte de um somador de 4 bits.

**Resultado:** Necessita-se do resultado da soma e do cout do A na posição de 0, pois o VHDL executa o código fonte de forma paralela e não tem como fazer um valor depender de outro na forma convencional. Portanto pra isso foi declarado o **process** como complemento da arquitetura como um meio somador pra que o código respondesse de forma sequencial e poder atender a necessidade da soma de 4 bits e assim atendendo o valor que depende do valor anterior, sem atrapalhar o código; pois as variáveis dentro do processo são validas somente dentro do processo como mostra o código na **Figura 5**. Após a compilação a simulação foi feito com os dados em binários com um clock de 80ns. Observando o resultado quando a soma ultrapassou o valor máximo de 4 bits o cout recebe o vai 1, que funciona como um



estouro ou cary, mas ainda sim a soma continua no seu ritmo mais com o cout em nível 1. Contudo todos os resultados foram alcançados com êxito como mostra a **Figura 6**.

## Atividade 04

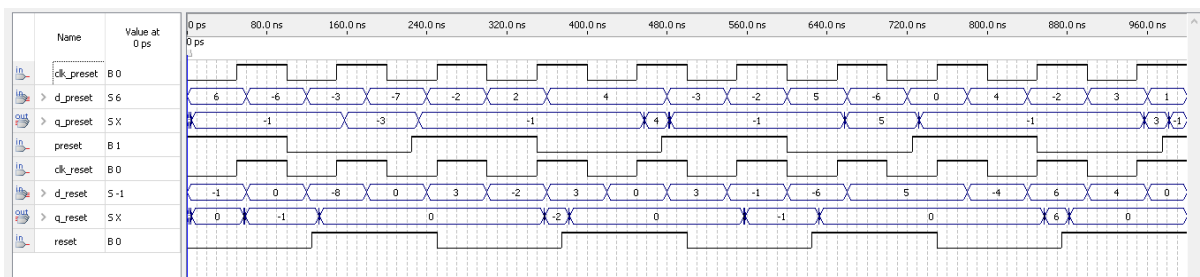
Implementar, comentar e executar o código fonte para o circuito assíncrono reset e preset.

```

1  LIBRARY ieee; -- declara bibliotecas externas ieee
2  USE ieee.std_logic_1164.all; -- chama funcoes da biblioteca
3
4  ENTITY dff_logic_mix IS -- declara entidade
5  PORT ( -- declara pinagem do modelo
6      clk_reset, clk_preset, preset, reset : IN std_logic; -- declara portas de entrada preset e reset
7      d_preset : IN std_logic_vector(3 downto 0); -- declara vetores de entrada preset
8      q_preset : OUT std_logic_vector(3 downto 0); -- declare vetor de saida preset
9      d_reset : IN std_logic_vector(3 downto 0); -- declara vetor de entrada reset
10     q_reset : OUT std_logic_vector(3 downto 0) -- declara vetor de saida reset
11 ); -- finaliza declaracao de pinagem
12 END dff_logic_mix; -- finaliza entidade
13
14
15 ARCHITECTURE behavior_mix OF dff_logic_mix IS -- declara arquitetura que usa a entidade dff_logic_mix
16 BEGIN -- inicio arquitetura
17     PROCESS(clk_preset, clk_reset, preset, reset) -- para verificar os valores
18     BEGIN -- inicio verificacao
19         -- parte do preset -----
20         IF preset = '1' THEN -- se preset ativo faca
21             q_preset <= (others => '1'); -- vetor saida do preset com todos os valores dos indices igual a 1
22         ELSIF RISING_EDGE(clk_preset) THEN -- senao se clk_preset na borda de subida faca
23             q_preset <= d_preset; -- vetor saida vai ser igual ao vetor de entrada do preset atual
24         END IF; -- fim do if do preset
25         -- finaliza preset -----
26         -- parte do reset -----
27         IF reset = '1' THEN -- se reset ativo faca
28             q_reset <= (others => '0'); -- vetor saida do reset com todos os valores dos indices igual a 0
29         ELSIF RISING_EDGE(clk_reset) THEN -- senao se clk_reset na borda de subida faca
30             q_reset <= d_reset; -- vetor saida do reset vai ser igual ao vetor entrada do reset atual
31         END IF; -- fim do if do reset
32         -- finaliza reset -----
33     END PROCESS; -- finaliza verificacao
34 END behavior_mix; -- finaliza arquitetura
35

```

**Figura 7:** Código fonte circuito assíncrono reset-preset.



**Figura 8:** Simulação do código compilado no Quartus II

**Resultado:** O código do circuito assíncrono foi escrito e compilado com sucesso no Quartus II. Na figura 7 temos o código completo todo comentado. A entidade foi aproveitada a partir do código do flip-flop d. o reset e o preset foram unificados na mesma arquitetura, porém cada um possui suas entradas e saídas separadas.

Na figura 8 é possível notar o resultado da execução. Seguindo a tabela verdade do preset, quando preset ativo a saída é 1, caso ocorra borda de subida com preset ativo, o valor da entrada atual é retornado na saída.

Por fim, para a tabela verdade do reset, enquanto reset ativo a saída é 0, caso ocorra borda de subida com reset ativo, o valor da entrada atual é retornado na saída.

Como é possível verificar na figura 8, a simulação ocorreu conforme o esperado.

## Atividade 05

Implementar o funcionamento de um Flip Flop D, comentar o seu funcionamento e executar uma simulação do código desenvolvido.

Antes de mostrar o código do Flip Flop D é essencial que seja compreendido como é o seu funcionamento, na figura 9 podemos ver a tabela verdade do Flip Flop onde mostra o seu comportamento de acordo a entrada D e o CLOCK, podemos verificar que a saída resultante depende da ativação do clock e da entrada D, resumindo a saída somente será alterada no momento da ativação do clock, caso o clock seja ativado, a saída q será atribuída o valor de entrada D e assim por diante continuamente de acordo com as ativações do clock.

D	CLK	SAÍDA
0	↑	Q = 0
1	↑	Q = 1

**Figura 9: Tabela Verdade do Flip Flop D**

Agora que já sabemos como funciona o Flip Flop D podemos partir para a implementação, ou seja, o código fonte que irá simular o seu funcionamento no software quartus web, na figura 10 encontra-se o código fonte implementado onde foi

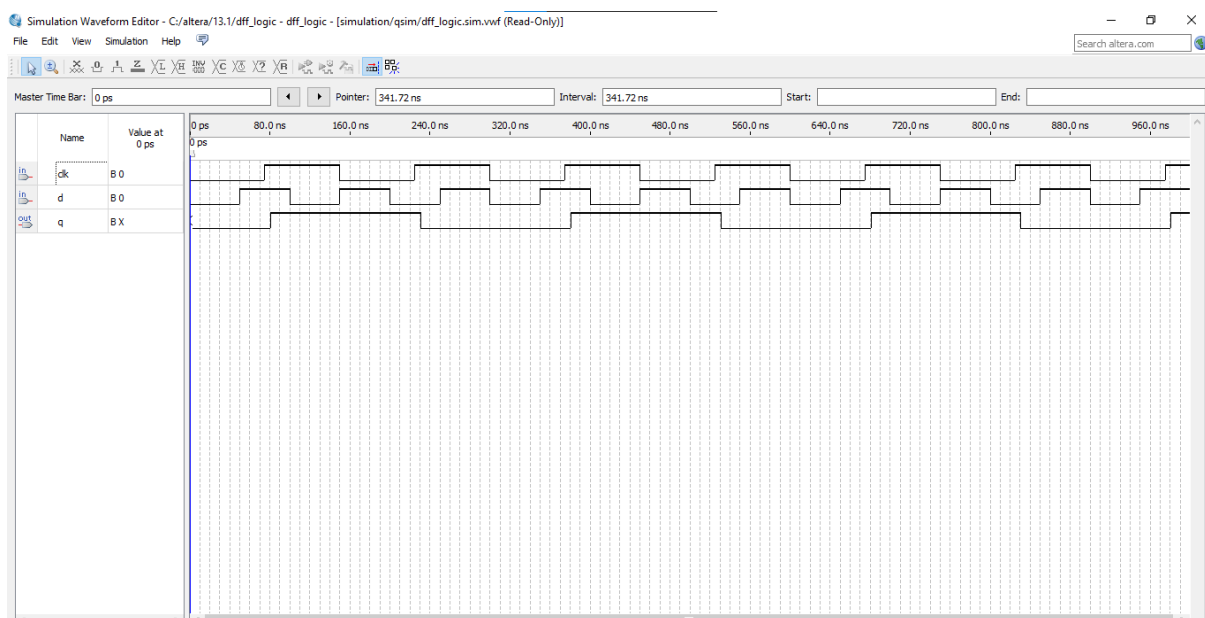
utilizado a biblioteca ieeee para utilizar as funções e as definições necessárias para o seu correto funcionamento. Além disso, foi criado a entidade dff\_logic que é onde são declarados as entradas e saídas do nosso circuito pretendido, no caso do Flip Flop D foram declaradas as entradas **d** e **clk** e como saída o **q** todos do tipo **BIT**, pois na prática seu funcionamento só admite entradas 0 e 1. Além disso, foi definido sua arquitetura chamada de behavior que descreve o comportamento do software, ou seja, o que o código realmente irá fazer, dentro da arquitetura behavior podemos verificar com o auxílio da estrutura condicional **if** que a saída **q** só é atribuída com o valor de entrada **d** caso o **clk** seja ativado demonstrando assim que a lógica do código está de acordo com o funcionamento prático do Flip Flop D.

```

1  library ieee; --DEFINICAO DA BIBLIOTECA IEEE para usar definicoes do tipo bit booleano entre outras coisas
2  USE ieee.std_logic_1164.all; --DECLARACAO NECESSARIA PARA USAR OS DADOS CORRESPONDENTES A LÓGICA PADRAO DA BIBLIOTECA
3  ENTITY dff_logic IS --DECLARACAO DA ENTIDADE CONTENDO OS PINOS I/O.
4  PORT (d,clk:IN BIT; --DECLARACAO DAS ENTRADAS D E CLOCK DE TIPOS BITS
5        q:OUT BIT); --DECLARACAO DA SAIDA Q DO TIPO BIT
6  END dff_logic; -- FIM DA DEFINICAO DA ENTIDADE
7  ARCHITECTURE behavior OF dff_logic IS --DECLARACAO DA ARQUITETURA, OU SEJA, A LÓGICA DO CIRCUITO
8  BEGIN --INICIO DA ARQUITETURA
9      PROCESS(clk) --CLK CORRESPONDE A LISTA DE SINAIS QUE PODEM ALTERAR A SAIDA DO CIRCUITO
10     BEGIN --Inicio da descricao logica do processo
11         IF (clk'event AND clk = '1') THEN --INICIO DA CONDICIONAL
12             q <= d; --ACAO QUE DEVERA FAZER CASO A CONDICAO SEJA FAVORECIDA
13         END IF; --FIM DA CONDICIONAL
14     END PROCESS; --FIM DO PROCESSO
15 END behavior; --FIM DA ARQUITETURA, OU SEJA, DA LÓGICA DO CIRCUITO

```

**Figura 10: Código Fonte do Flip Flop D**



**Figura 11: Simulação do Flip Flop D**

**Resultado:** Verifica-se que a simulação funcionou corretamente, pois os valores de saída **q** estão mudando de acordo com as ativações do **clock** e o valor de entrada **d**,

demonstrando assim que o código implementado está correto, dessa forma alcançando os resultados esperados.

## **Conclusões**

Concluiu-se que para obter êxito nas atividades foram encontradas algumas dificuldades na execução do código e também na instalação do Quartus II para a instalação e devido a sua política de uso doméstico. Por ser um programa robusto em que usamos o programa de forma gratuita, então é normal que ocorra alguns bugs.

Contudo foi favorável o resultado devido a persistência e a correção de alguma linha de código e sobretudo a representação do corpo do código.

Pra finalizar obtivemos uma experiencia sobre o VHDL e o que ele representa para a disciplina e o aprendizado em Sistemas Embarcados.

## **Referências**

<https://stackoverflow.com/questions/59000409/vhdl-usage-of-high-impedance>

Acesso em 04/07/2021.