LAB 12

Atividade 1

a) Usando o código abaixo, como base, implementar um FF D sensível a borda de subida, com MUX 2:1 na entrada D. O MUX 2:1 deve ser feito separadamente e usado como componente (usar package) no arquivo do FF D. Fazer a simulação no MODELSIM.

b) Fazer um arquivo que mapeie as variáveis de entrada e saída nas entradas e saídas da placa e verifique o funcionamento do circuito. MAPEAMENTO:

SW(0)→ D0
SW(1)→ D1
SW(2) → SEL
Clock →KEY(0)
LEDR(0)→ Q
LEDR(1)→ D1
LEDR(2) → SEL

CÓDIGO – ATIVIDADE 1

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY muxdff IS
        PORT ( D0, D1, Sel, Clock : IN    STD_LOGIC ;
                Q                 : OUT STD_LOGIC ) ;
END muxdff ;
ARCHITECTURE Behavior OF muxdff IS
BEGIN
        PROCESS
        BEGIN
                WAIT UNTIL Clock'EVENT AND Clock = '1' ;
                IF Sel = '0' THEN
                        Q <= D0 ;
                ELSE
                        Q <= D1 ;
                END IF ;
        END PROCESS ;
END Behavior ;
```

OBSERVAÇÃO:

Clock'EVENT AND Clock* = '1' é similar a *rising_edge(clock)

Atividade 2

a) Usando o código abaixo, como base, projetar um registrador de deslocamento de 4 bits com código hierárquico, ou seja, usar o projeto da atividade 1 como componente (usar package). Fazer a simulação no MODELSIM.
b) Fazer um arquivo que mapeie as variáveis de entrada e saída nas entradas e saídas da placa e verifique o funcionamento do circuito.
MAPEAMENTO:

SW(3..0)→ R
HEX(0) → R
HEX(2) → Q
SW(5) → W
KEY(0) → Clock
KEY(1) → L

CÓDIGO – ATIVIDADE 2

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shift4 IS
        PORT ( R                : IN            STD_LOGIC_VECTOR(3 DOWNTO 0) ;
                L, w, Clock     : IN            STD_LOGIC ;
                Q               : BUFFER        STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;
ARCHITECTURE Structure OF shift4 IS
        COMPONENT muxdff
                PORT ( D0, D1, Sel, Clock  : IN   STD_LOGIC ;
                        Q                  : OUT STD_LOGIC ) ;
        END COMPONENT ;
BEGIN
        Stage3: muxdff PORT MAP ( w, R(3), L, Clock, Q(3) ) ;
        Stage2: muxdff PORT MAP ( Q(3), R(2), L, Clock, Q(2) ) ;
        Stage1: muxdff PORT MAP ( Q(2), R(1), L, Clock, Q(1) ) ;
        Stage0: muxdff PORT MAP ( Q(1), R(0), L, Clock, Q(0) ) ;
END Structure ;
```

Atividade 3

a) Usando o código abaixo, como base, projetar contador crescente de 4 bits. Fazer a simulação no MODELSIM.

b) Fazer um arquivo que mapeie as variáveis de entrada e saída nas entradas e saídas da placa e verifique o funcionamento do circuito. MAPEAMENTO:

HEX(0) → Q
SW(0)→ E
SW(5) → W
Clock→ KEY(0)
Resetn → KEY(1)

CÓDIGO – ATIVIDADE 3

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
        PORT (Clock, Resetn, E : IN      STD_LOGIC ;
                Q              : OUT   STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount ;
ARCHITECTURE Behavior OF upcount IS
        SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
        PROCESS ( Clock, Resetn )
        BEGIN
                IF Resetn = '0' THEN
                        Count <= "0000" ;
                ELSIF (Clock'EVENT AND Clock = '1') THEN
                        IF E = '1' THEN
                                Count <= Count + 1 ;
                        ELSE
                                Count <= Count ;
                        END IF ;
                END IF ;
        END PROCESS ;
        Q <= Count ;
END Behavior ;
```

Atividade 4

a) Usando o código abaixo, como base, projetar contador crescente de 4 bits com carga paralela, usando sinais tipo INTEGER. Fazer a simulação no MODELSIM.

b) Fazer um arquivo que mapeie as variáveis de entrada e saída nas entradas e saídas da placa e verifique o funcionamento do circuito. MAPEAMENTO:

SW(3..0)→ R
HEX(0) → Q
SW(0)→ E
SW(5) → W
Clock→ KEY(0)
Resetn → KEY(1)
L → KEY(2)

CÓDIGO – ATIVIDADE 4

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY upcount IS
        PORT ( R                    : IN                INTEGER RANGE 0 TO 15 ;
                Clock, Resetn, L        : IN                STD_LOGIC ;
                Q                   : BUFFER  INTEGER RANGE 0 TO 15 ) ;
END upcount ;
ARCHITECTURE Behavior OF upcount IS
BEGIN
        PROCESS (Clock, Resetn )
        BEGIN
                IF Resetn = '0' THEN
                        Q <= 0 ;
                ELSIF (Clock'EVENT AND Clock = '1') THEN
                        IF L = '1' THEN
                                Q <= R ;
                        ELSE
                                Q <= Q + 1 ;
                        END IF;
                END IF;
        END PROCESS;
END Behavior;
```

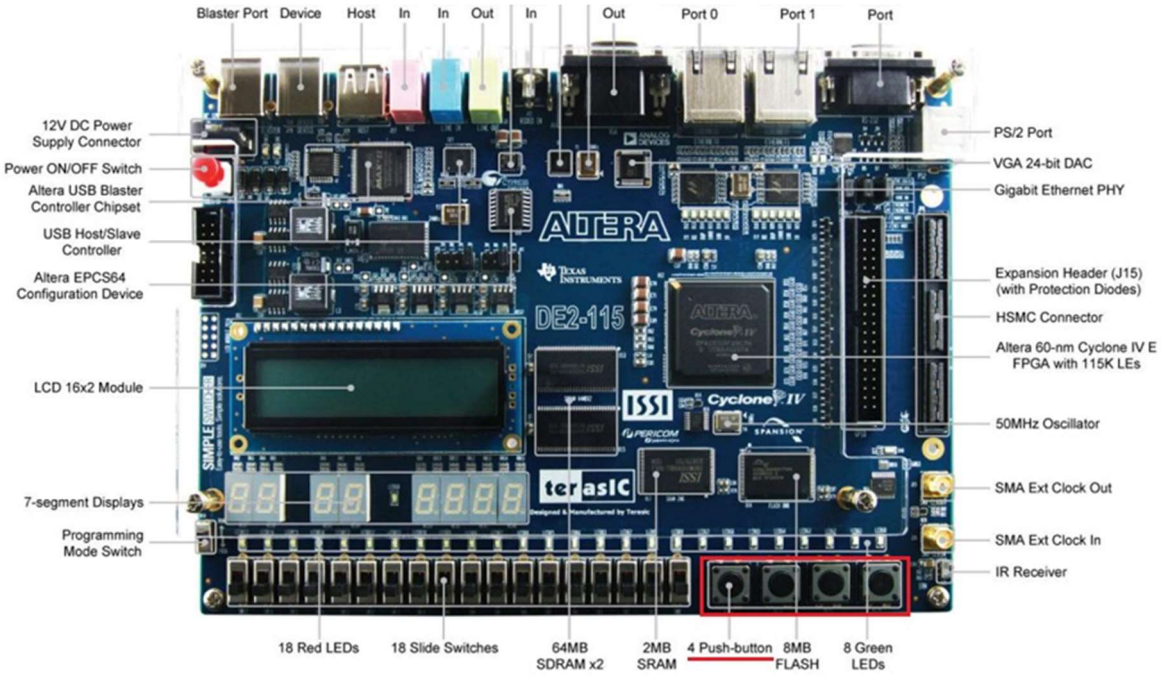
ANEXO

UTILIZAÇÃO DE BOTÕES NA PLACA

Vinicius Galassi

INTRODUÇÃO

A placa DE2-115 possui quatro botões que podem ser utilizados para emitir sinais lógicos. Neste material, será abordado como utilizar esses botões para simular sinais de clocks e reset em componentes como flip-flops.
Os botões estão localizados na parte inferior direita da placa:



REFERENCIANDO BOTÕES NO CÓDIGO

Para utilizar os botões, primeiramente, vamos criar um projeto no Quartus e um arquivo.vhd.
Para referenciar corretamente os botões da placa no nosso código, é preciso importar o Pin Assignments. Em seguida, vá para a aba Assignments e clique em Edit Assignments para chegar na seguinte tela:

	Status	From	To	Assignment Name	Value	Enabled	Entity
60	? '...		LEDG[6]	I/O Standard	2.5 V	Yes	DE2_115
61	? '...		LEDG[7]	I/O Standard	2.5 V	Yes	DE2_115
62	? '...		LEDG[8]	I/O Standard	2.5 V	Yes	DE2_115
63	? '...		KEY[0]	I/O Standard	3.3-V LVTTL	Yes	DE2_115
64	? '...		KEY[1]	I/O Standard	3.3-V LVTTL	Yes	DE2_115
65	? '...		KEY[2]	I/O Standard	3.3-V LVTTL	Yes	DE2_115
66	? '...		KEY[3]	I/O Standard	3.3-V LVTTL	Yes	DE2_115
67	? '...		EX_IO[0]	I/O Standard	3.3-V LVTTL	Yes	DE2_115
68	? '...		EX_IO[1]	I/O Standard	3.3-V LVTTL	Yes	DE2_115
69	? '...		EX_IO[2]	I/O Standard	3.3-V LVTTL	Yes	DE2_115
70	? '...		EX_IO[3]	I/O Standard	3.3-V LVTTL	Yes	DE2_115
71	? '...		EX_IO[4]	I/O Standard	3.3-V LVTTL	Yes	DE2_115
72	? '...		EX_IO[5]	I/O Standard	3.3-V LVTTL	Yes	DE2_115

Nessa tela, é possível ver os nomes que devem ser utilizados para conectar os sinais aos componentes físicos da placa.
No caso dos botões, eles são referenciados como KEY[0], KEY[1], KEY[2] e KEY[3]. Por isso, ao declarar as nossas entradas no nosso código VHDL, devemos usar exatamente o nome KEY. Os colchetes indicam que se trata de um vetor.
Agora vamos adicionar os componentes no código de um flip-flop, incluindo os *push buttons, switches* e *leds.*

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3       ENTITY ff_keys IS
4           PORT (
5           -- Botões Push Ups
6           KEY: in std_logic_vector(1 downto 0);
7           -- Switches
8           SW: in std_logic_vector(0 downto 0);
9           -- LEDS
10          LEDR: out std_logic_vector(0 downto 0)
11          ) ;
12      END ff_keys ;
```

Nesse exemplo, utilizaremos apenas dois botões, por isso, o vetor KEY terá apenas dois elementos.
Em seguida, será necessário atribuir sinais aos componentes da placa.

```
15  ARCHITECTURE Behavior OF ff_keys IS
16      signal D,Resetn, clock,Q: std_logic;
17  BEGIN
18      -- Sinal de clock será o botão 0
19      clock <= KEY(0);
20      -- Sinal de reset será o botão 1
21      Resetn <= KEY(1);
22      -- Q será o led
23      LEDR(0) <= Q;
24      -- D será o SW 1
25      D <= SW(0);
```

Com isso, está definido que:

- O botão 0 será o sinal de clock;
- O botão 1 será o sinal de reset;
- O led 0 será o sinal Q;
- O sinal D será o switch 0;

Por fim, vamos implementar a lógica de um flip flop assíncrono.

```
27      PROCESS ( Resetn, Clock )
28          BEGIN
29          IF Resetn = '0' THEN
30              Q <= '0' ;
31          ELSIF Clock'EVENT AND Clock = '1' THEN
32              Q <= D ;
33          END IF ;
34      END PROCESS ;
35
36
37  END Behavior ;
```

Vale lembrar que o comando *process* é utilizado para que a execução do código dentro dele seja sequencial, fazendo com que a ordem das atribuições importe.
Além disso, a lista de sensibilidade (parênteses, ao lado do *process*) indica que o código dentro do bloco será executado sempre que algum sinal da lista for alterado. No exemplo mostrado, o código será executado toda vez que Resetn ou Clock mudar.

POR QUE UTILIZAR BOTÕES PARA O SINAL DE CLOCK E RESET

A utilização de botões em vez de switches facilita a simulação do comportamento do clock e reset na placa, pois os botões geram um pulso momentâneo, ao contrário dos switches, que mantêm um sinal contínuo. Além disso, a placa consegue identificar com mais facilidade as bordas de saída e subida do sinal quando este é gerado por um botão.