UNIVERSIDADE DE BRASÍLIA

Adrianne Alves da Silva - 160047595
Ana Paula Gomes de Matos - 160023629

TURMA "F"
PRÁTICA DE ELETRÔNICA DIGITAL - 119466
Experimento 06: Flip - Flops

Brasília - DF

02 de junho de 2017

UNIVERSIDADE DE BRASÍLIA

PRÁTICA DE ELETRÔNICA DIGITAL - 119466

Relatório referente ao quarto experimento da disciplina Prática de Eletrônica Digital, ministrada pela professora Lourdes Mattos Brasil, realizado no dia 26 de maio de 2017.

Alunas:

Adrianne Alves da Silva

Ana Paula Gomes de Matos

Brasília - DF

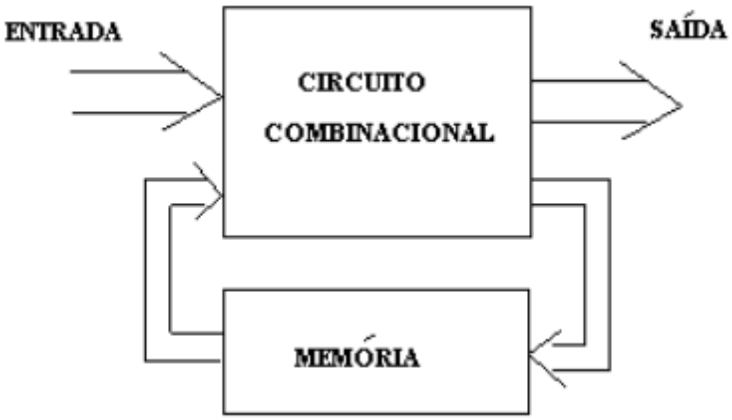
SUMÁRIO

1 INTRODUÇÃO

Podemos classificar os circuitos digitais como combinacionais ou sequenciais. Os primeiros são aqueles em que suas saídas dependem apenas dos níveis lógicos aplicados na entrada, ou seja, a mesma combinação de entrada sempre produzirá o mesmo resultado na saída, pois os circuitos combinacionais não possuem memória. O segundo tipo de circuito aqui abordado são aqueles em que suas saídas, em um determinado período do tempo, não dependem apenas das entradas naquele instante, mas depende também das entradas anteriores e da sequência como elas foram aplicadas. Em circuitos sequenciais, o circuito de memória mais utilizados são os Flip-Flops e os LATCHS que são dispositivos biestáveis, ou seja, possuem dois estados estáveis (0,1), desta forma, eles permanecem em um desses estados até que ocorra algum evento que o faça assumir um outro estado considerado estável. O que caracteriza um flip-flop como sendo um dispositivo de memória é o fato dele manter uma informação ao longo do tempo. [1]

Figura 01: Circuito combinacional

Figura 02: Circuito sequencial

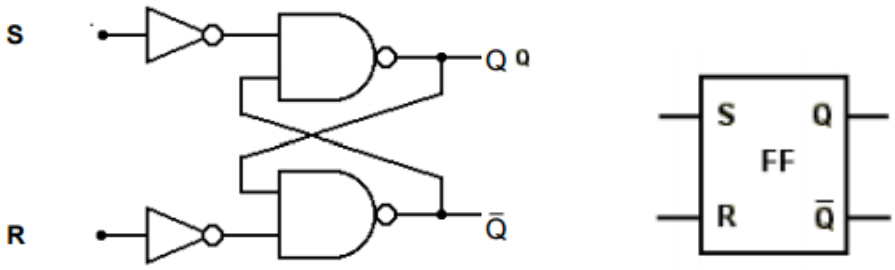


1.1 Diferença entre Flip-Flops e Latch

Como visto anteriormente, assim como o flip-flop o Latch também é um dispositivo de armazenamento temporário que possui dois estados estáveis (biestável) e que pode permanecer em um dos dois estados usando uma configuração de realimentação, na qual as saídas são ligadas às entradas opostas. A principal diferença entre esses dispositivos é o método usado em cada estado, pois nos Latchs a mudança de estado ocorre de forma assíncrona e nos flip-flops é de maneira síncrona, ou seja, por meio de uma ação de disparo. Vale mencionar que os latchs são mais utilizados como memória e os flip-flops como contadores. O latch utiliza diretamente o nível alto ou baixo para mudar suas saídas, em resumo, o latch é sensível ao nível e o flip-flop à borda. [1]

1.2 Flip-Flop R – S (Reset – Set)

Figura 03: Circuito Flip-Flop R-S com porta NAND e sua simbologia



Nesse tipo de circuito o estado futuro do **Q** e de seu complemento dependem das entradas R e S, isso fora o estado atual das saídas. Ao observar a tabela verdade de um flip-flop, **Q** se refere ao estado atual da saída e $\mathbf{Q_0}$ ao estado anterior da saída de **Q**. Caso as saídas de **Q** e $\bar{\mathbf{Q}}$ não sejam complementares, será indicado com um (*) por ser um estado proibido.[1]

Tabela 01: Tabela verdade de um Flip-Flop R-S

S	R	Q	$\bar{Q}$
0	0	Q_0	$\bar{Q}_0$
1	0	1	0
0	1	0	1
1	1	*	*

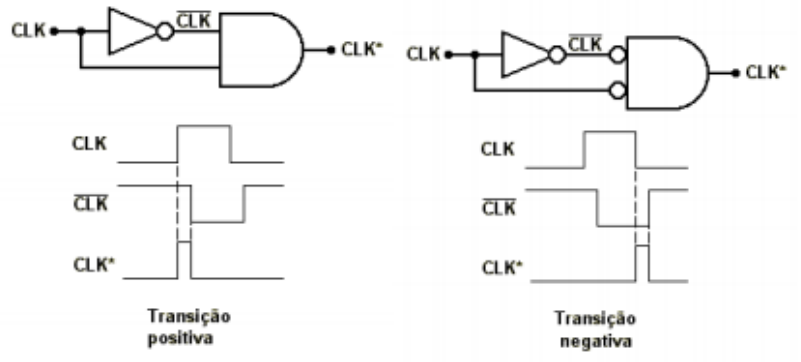
Observação: o circuito do flip-flop R-S também pode ser implementado usando portas

NOR.

1.3 Flip-Flops com clock

Circuitos síncronos são aqueles que utilizam clock, muitos flip-flops utilizam o sinal de clock para identificar o momento em que suas saídas mudarão de estado. Em geral o clock é uma onda quadrada e é uma parte comum para todas as partes do circuito, o detector de transição é um circuito que habilita as entradas durante a mudança de clock, veja abaixo um circuito típico de um detector de transição. [2]

Figura 04: Circuitos detectores de transição positiva e negativa



Os tempos dos pulsos de CLK correspondem aos tempos de atraso de uma porta inversora.

1.4 Flip-Flop R-S com clock

A imagem a seguir mostra o circuito interno de um flip-flop R-S e seu símbolo com clock.

Figura 05: Circuito interno do flip-flop R-S e símbolo de clk.

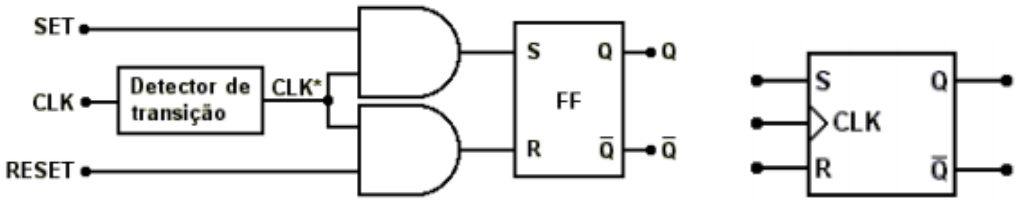


Tabela 02: Tabela verdade do flip-flop R-S com clock

S	R	Clk	Q	$\bar{Q}$
X	X	0	Q_0	$\bar{Q}_0$
0	0	↑	Q_0	$\bar{Q}_0$
1	0	↑	1	0
0	1	↑	0	1
1	1	↑	*	*

Quando o nível de clck está ativo ele altera a saída de acordo com as variáveis de entrada.

1.5 Flip-Flop J-K

A diferença entre o J-K e o R-S está no fato em que o J-K não possui uma condição proibida como pode ser visto na tabela abaixo.

Tabela 03: Tabela verdade do flip-flop J-K.

J	K	Clk	Q	$\bar{Q}$
X	X	0	Q_0	$\bar{Q}_0$
0	0	↑	Q_0	$\bar{Q}_0$
1	0	↑	1	0
0	1	↑	0	1
1	1	↑	$\bar{Q}_0$	Q_0

1.6 Flip-Flop T (Toggle)

Possui uma única entrada e J-K são conectados em um ponto determinado da entrada T, se a entrada for elevada em nível alto este flip-flop opera como divisor de frequência.

Tabela 04: Tabela verdade do flip-flop T.

T	CLK	Q	$\bar{Q}$
X	0,1	Q_0	$\bar{Q}_0$
0	↑	Q_0	$\bar{Q}_0$
1	↑	$\bar{Q}_0$	Q_0

1.7 Flip-Flop D

Também possui entrada única, onde J e K (ou R e S) são conectados por meio de um inversor em um único ponto denominado de entrada T.

Tabela 05: Tabela verdade do flip-flop D.

D	Clk	Q	$\bar{Q}$
X	0	Q_0	$\bar{Q}_0$
0	↑	0	1
1	↑	1	0

1.8 Entradas assíncronas

Todas as entradas que dependem do sinal de clock são chamadas de entradas síncronas e as que não dependem desse sinal são denominadas assíncronas. Essas entradas são usadas para alterar a qualquer instante o estado do flip-flop. A tabela abaixo mostra a tabela verdade das entradas síncronas PRESET e CLEAR. Estas entradas geralmente são ativas em nível baixo, pois a tecnologia TTL a corrente de entrada em nível alto é muito menos que no nível baixo, isso resulta em um menor consumo de potência no CI. [2]

Tabela 06: Tabela verdade das entradas síncronas PRESET e CLEAR.

$\overline{PRE}$	$\overline{CLR}$	Q	$\bar{Q}$
1	1	operação normal	
0	1	1	0
1	0	0	1
0	0	*	*

Para operação normal do flip-flop, as entradas RESET e CLEAR devem estar em “1”, contudo, a qualquer momento pode-se mudar a saída **Q** para “0” ou “1” utilizando estas entradas, contudo, a última combinação não pode ser usada. A imagem abaixo mostra as entradas assíncronas de um flip-flop J-K e sua tabela verdade:

Tabela 07: Entradas assíncronas de um flip-flop J-K e sua tabela verdade.

$\overline{PRE}$	$\overline{CLR}$	J	K	Clk	Q	$\bar{Q}$
0	1	X	X	X	1	0
1	0	X	X	X	0	1
1	1	X	X	0	Q_0	$\bar{Q}_0$
1	1	0	0	↑	Q_0	$\bar{Q}_0$
1	1	1	0	↑	1	0
1	1	0	1	↑	0	1
1	1	1	1	↑	$\bar{Q}_0$	Q_0

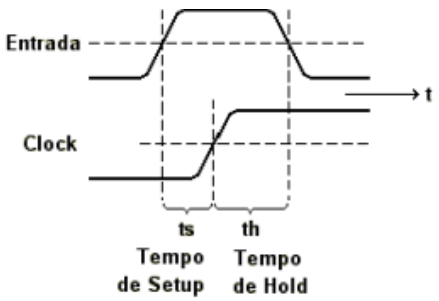
1.9 Temporizações dos flip-flops

As características que serão apresentadas a seguir devem ser respeitadas para que os flip-flops funcionem corretamente.

1.9.1 Tempo de Ajuste (setup) e conservação (hold)

Os tempos de setup e hold são parâmetros que devem ser observados para que o flip-flop possa trabalhar de modo confiável. O primeiro corresponde ao intervalo mínimo de tempo no qual as entradas devem permanecer estáveis antes do clock ser alterado. Já o tempo de hold faz referência ao intervalo mínimo no qual as entradas devem permanecer estáveis depois da transição de clock. [1]

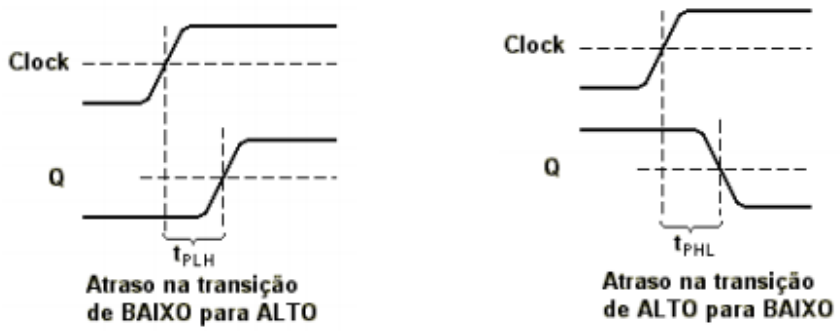
Figura 06: Tempos de setup e hold



1.9.2 Atrasos de propagação

Atrasos de propagação consiste no intervalo de tempo entre a aplicação de um sinal na entrada e o momento de mudança da saída. Esse atraso pode ocorrer em uma transição de descida ou subida. [1]

Figura 07: Atrasos de propagação



1.9.3 Frequência máxima de clock

É a frequência mais alta sob a qual um flip-flop pode ser submetido de modo a garantir seu funcionamento de forma confiável.

Figura 08: Tempos de duração do clock em ALTO e BAIXO

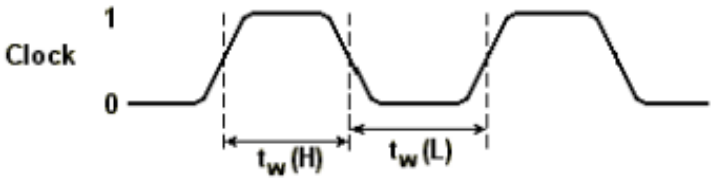
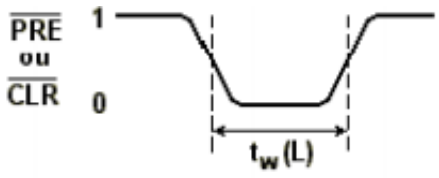


Figura 09: Largura dos pulsos assíncronos



1.9.4 Tempos de transição do clock

Para garantir o devido funcionamento do flip-flop, o tempo de transição dp clck precisa ser o menor possível. Nos dispositivos TTL esse tempo é menor ou igual a 50 ns e

para dispositivos CMOS é menor ou igual a 200 ns.

1.10 Objetivo

Mostrar ao aluno o funcionamento dos flip-flops e sua implementação em registradores e divisores de frequência.

1.11 Justificativa

O uso de memória para o armazenamento de informações é de extrema importância para qualquer atividade desenvolvida atualmente. Nesse sentido, o uso desses dados em esquemas tecnológicos retroalimentados tornou-se algo comum. Dessa forma, a compreensão dos circuitos que realizam essa atividade é de suma importância para o estudante.

Os circuitos que utilizam a lógica sequencial presente nos flips flops tornaram-se alvo de estudos pois estes são os componentes utilizados nas mais diversas aplicações que envolvam memória, de certa forma, eles dominam a tecnologia atual. Com isso em mente é possível compreender que o conhecimento desta tecnologia abre um leque de oportunidades de implementação, o que justifica o estudo a ser realizado neste experimento. É de suma importância, por tanto, não só conhecer, mas testar e presenciar o seu funcionamento.

Tendo em vista o que foi exposto, o experimento realizado no dia 26 de maio de 2017, no laboratório de Eletrônica Digital, foi capaz de unir a relevância dos circuitos sequenciais de memória às aplicações práticas e aplicáveis, informando e preparando o estudante para prováveis demandas de mercado. Dessa maneira, é de suma importância registrar todo o processo por meio de documentos relatórios. Essa perspectiva demonstra a real relevância e absorção de conteúdo que a visualização prática desses conceitos traz aos alunos.

2 PARTE EXPERIMENTAL

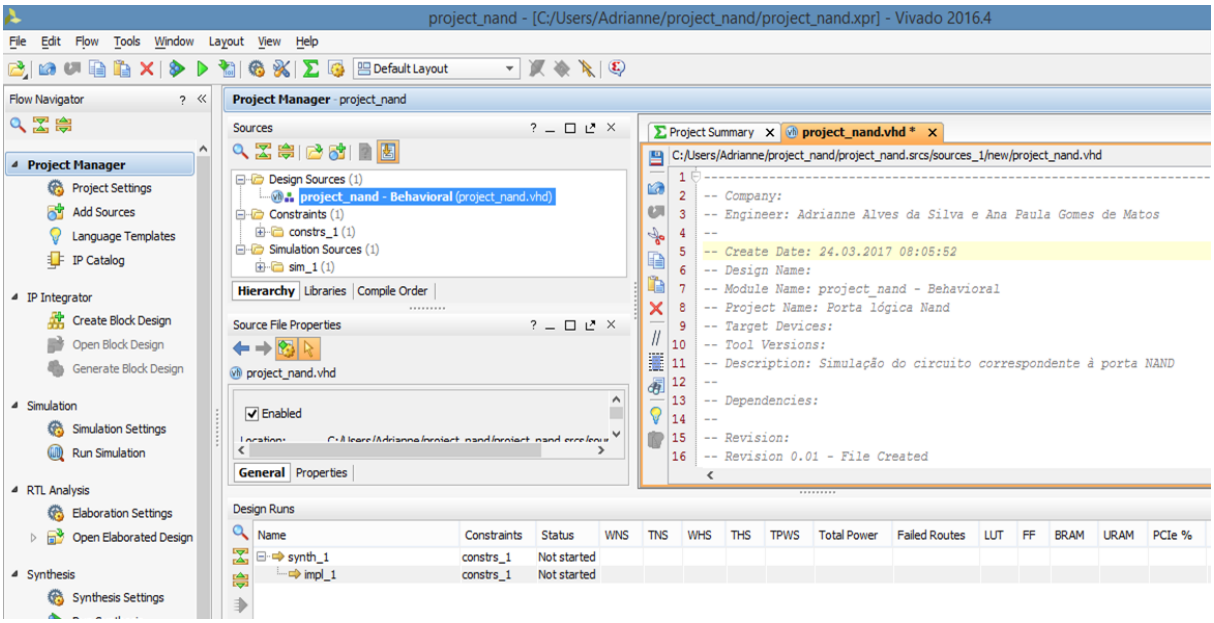
O software utilizado para codificar e simular os experimentos foi o VIVADO, versão 2013.4, rodando no sistema operacional Windows 8, em uma máquina pessoal. Parte do experimento foi realizado em sala, nos computadores da universidade.

2.1 Preparação do ambiente de simulação

O passo inicial para a realização do experimento foi criar um novo projeto no *software* Vivado. A configuração do mesmo foi realizada adicionando a placa Basys 3 ao projeto, por meio da aba "*add or create constraints*". Após esses passos foi necessário apenas criar o arquivo relativo ao *design* , adicionando o arquivo com o tipo vhdl, para que o código pudesse ser implementado nesta linguagem. O módulo foi então definido de acordo com o número de portas de entrada e saída do circuito referente à cada atividade solicitada, nomeadas as entradas, em sua maioria como A,B, va, vb , Cin,sel e Saída com S.

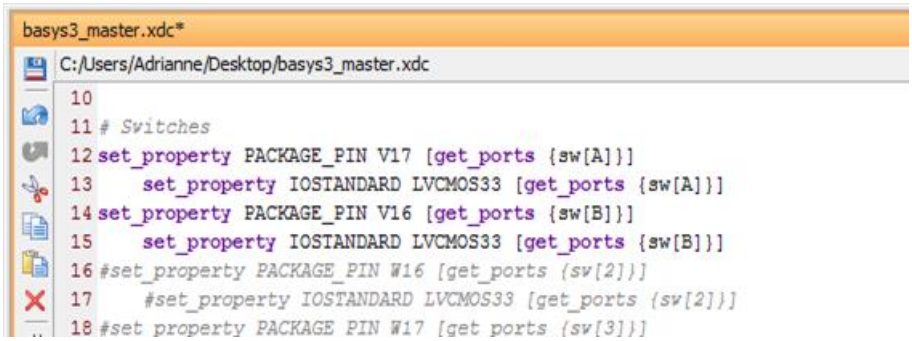
É necessário então ativar na aba de *project settings* , *bitstream* o arquivo bin_file que possibilita escrita binária sem uso do cabeçalho. Assim, obteve-se o ambiente demonstrado na figura a seguir.

Figura 10: Ambiente de trabalho



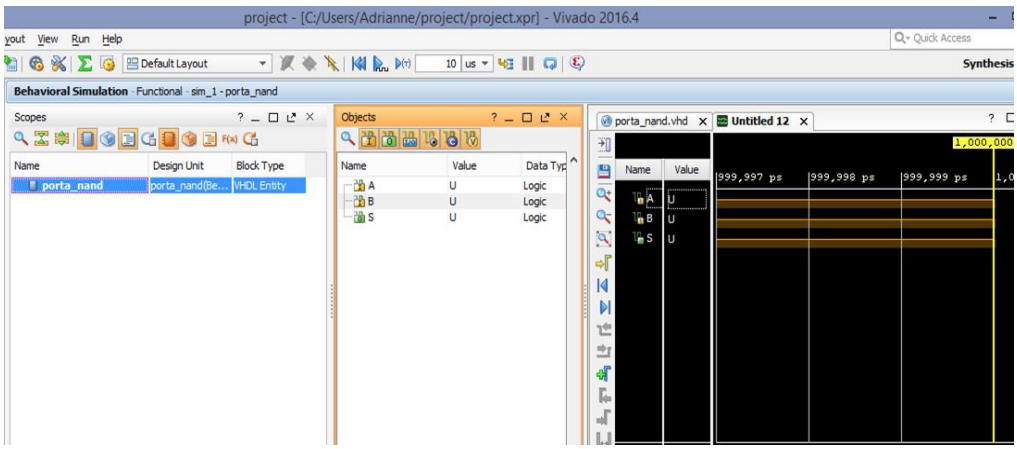
Sendo assim, foi possível abrir o arquivo vhdl definido e implementá-lo. Para cada atividade, forneceu-se ao arquivo o código e as expressões booleanas adequadas. Por sequência do passo-a-passo fornecido, foram linkados os pinos de entrada e saída da placa Basys com os dispositivos de entrada e saída implementadas. No arquivo basys3.xdc (**figura 8**), isso foi feito por meio da retirada do comentário das linhas referentes ao número de entradas e saídas presentes em cada atividade, além da adequada atribuição de cada uma das portas, segundo a nomenclatura adotada (J, K, S, R, Sel, Q,,S), essa atividade seria essencial para a verificação diretamente numa FPGA.

Figura 11: Linkagem dos pinos da placa Basys 3



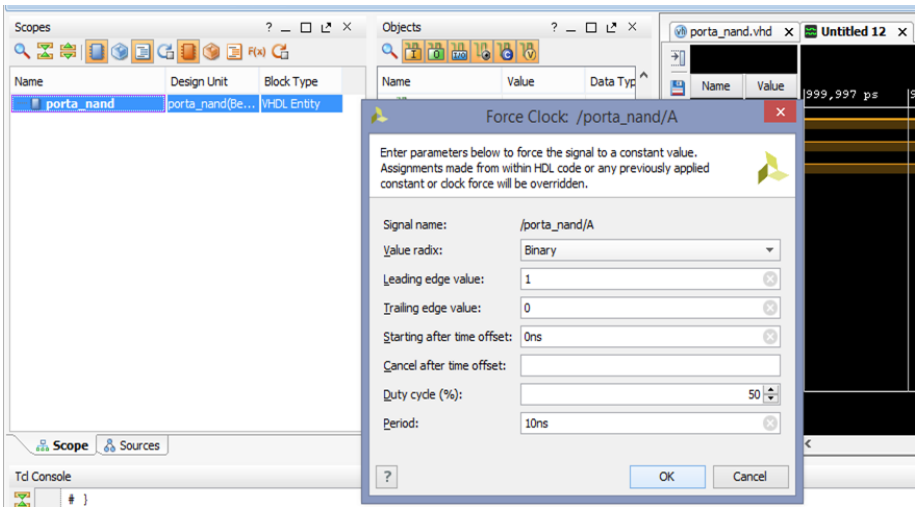
Após todos esses passos foi necessário sintetizar o circuito por meio do *Run Synthesis* , gerando um esquemático facilmente visualizado no *RTL Analysis* . Nesse ponto, é que tornou-se possível realizar a simulação do experimento, bastando clicar em *run simulation* que gera um ambiente de simulação (**figura 12**).

Figura 12: Ambiente de Simulação do circuito



Nesse passo, é possível configurar os valores referentes à cada entrada, assim como o tempo de mudança de sinal (0 ou 1) para a geração de ondas . Isso é feito clicando sobre cada entrada com o botão direito, na opção de *Force Clock*, conforme a figura a seguir.

Figura 13: Configuração de entradas para simulação



A primeira configuração realizada é a modificação do argumento *value* para o tipo binário informando entradas diferenciadas para os argumento *Leading edge value* e *Trailing edge value*, que seja 0 ou 1 . Define-se um período para as ondas, diferentes para cada entrada. É legal se o valor de uma for o dobro da outra, por exemplo. E então, basta executar a simulação.

2.2 Experimento

O experimento consistiu em três diferentes atividades envolvendo o flipflops, seja em termos de componente isolado, ou agrupamento para construção de registradores e divisores de frequência. Os termos técnicos e ou teóricos dos mesmos foram explanados anteriormente para total compreensão do que foi observado.

2.2.1 Simulação de Flip Flop

Tendo em vista os tipos de flip flops tratados , este experimento consistiu em codificar e simular um circuito que realizasse a função de um flip flop dos tipos JK, D e T. Para cada um desses tipos dedicou-se um total de 3 experimentos, sendo um voltado para um flip flop cujo clock é controlado por borda de subida e outro para bordas de descida. O terceiro ponto consistiu em fundir ambas as opções em um único programa.

2.2.1.1 Flip Flop JK

O problema relativo aos flip flops são relativamente simples, precisando de no

máximo três entradas e uma saída. O flip flop JK, especialmente, possui a seguinte tabela:

Tabela 08: Tabela verdade do flip flop jk

J	K	C	Q	$\overline{Q}$
0	0	p	no change	
0	1	p	0	1
1	0	p	1	0
1	1	p	toggle	

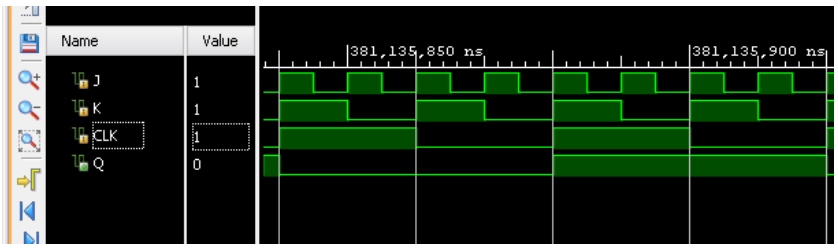
Dessa forma, bastava implementar as condições desta tabela. Isso foi feito utilizando-se process para garantir a execução sequencial dos comandos. A partir desse conceito, obteve-se o seguinte código para o flip flop por borda de subida:

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity jksubida is
        PORT( J,K,CLK: in BIT;
        Q: out BIT
        );
end jksubida;

Architecture behavioral of jksubida is
begin
        PROCESS(CLK)
        variable temp: BIT;
        begin
        if(CLK='1' and CLK'EVENT) then
                if(J='0' and K='0')then
                temp:=temp;
                elsif(J='1' and K='1')then
                temp:= not temp;
```

```
        elsif(J='0' and K='1')then
          temp:='0';
          else
          temp:='1';
          end if;
    end if;
    Q<=temp;
    end PROCESS;
end behavioral;
```

A simulação forneceu as seguintes formas de onda, tal que adotou-se, nas entradas para a simulação, os valores padrão de 1,0 e tempo de 10 ns que dobrou de uma entrada do circuito para outra (figura 14).

Figura 14: Ondas do flip flop JK por borda de subida



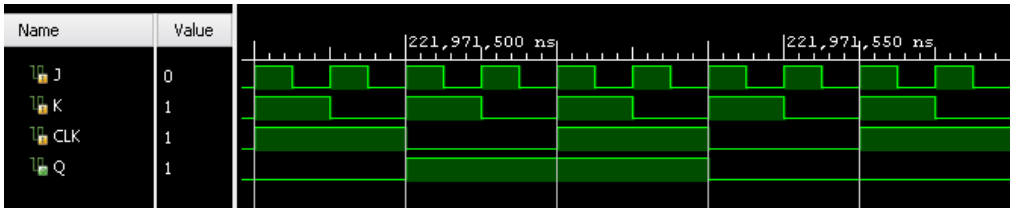
Para alterar os parâmetros sobre os quais varia o clock, bastou trocar a condição que acompanha o event, tal que a marcação seria agora para quando o clock possuísse o valor 0. Assim, obteve-se o código a seguir.

```
 library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity jkdescida is
    PORT( J,K,CLK: in BIT;
    Q: out BIT
    );
```

```
end jkdescida ;

Architecture behavioral of jkdescida is
begin
        PROCESS(CLK)
        variable temp: BIT;
        begin
        if(CLK='0' and CLK'EVENT) then
                if(J='0' and K='0')then
                temp:=temp;
                elsif(J='1' and K='1')then
                temp:= not temp;
               elsif(J='0' and K='1')then
                temp:='0';
                else
                temp:='1';
                end if;
        end if;
        Q<=temp;
        end PROCESS;
end behavioral;
```

Entretanto, como é de se esperar as ondas devem ser semelhantes, como pode ser observado a seguir.

Figura 15: Ondas do flip flop JK por borda de descida



Os esquemáticos encontram-se ao fim deste documento. Por fim, como solicitado, procurou-se unir ambas as opções através de uma seletora, cujo código simplesmente pôde ser escrito como:

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

entity Seletora is
    Port ( Sel : in bit;
    SS : in bit;
    SD : in bit;
    S : out bit);
end Seletora;

architecture Behavioral of Seletora is

begin
process (Sel)
begin
    if(Sel = '1') then
    S <= SS ;
    else
    S <= SD;
    end if;

end process;

end Behavioral;
```

Além disso, para unir o código relativo a cada flipflop foi preciso uma função principal que realizasse a gerência. O código implementado pode ser analisado abaixo.

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

entity conexao is
    Port ( s : in bit;
```

```
        r : in bit;
        clock : in bit;
        sel : in bit;
        sa : out bit);
end conexao;

architecture Behavioral of conexao is

component Seletora
        Port ( Sel : in bit;
        SS : in bit;
        SD : in bit;
        S : out bit);
end component;

component jkdescida
        PORT( J,K,CLK: in bit;
        Q: out bit
        );
end component;

component jksubida
        PORT( J,K,CLK: in bit;
        Q: out bit
        );
end component;

SIGNAL S1 : bit;
SIGNAL S2 : bit;

begin

descida : jkdescida port map(J =>S , K => R , CLK => Clock, Q=>S1 );
subida : jksubida port map(J =>S , K => R , CLK => Clock, Q=>S2);
```

sele : Seletora port map(Sel => sel, SS => S2, SD =>S1, S=>sa);

end Behavioral;

O esquema e a lógica utilizada pode ser mais facilmente observada através dos esquemáticos disponíveis nas últimas páginas.

2.2.1.2 Flip Flop D

O flip flop D possui uma lógica ainda menos complicada, possuindo apenas duas entradas, com uma responsável pelo clock, e uma única saída. A sua lógica obedece à seguinte relação

Tabela 09: Tabela verdade do flip flop D

clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

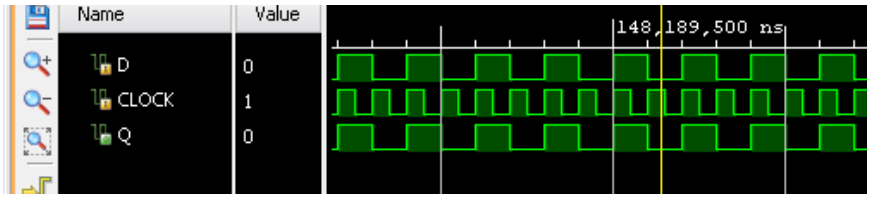
Semelhantemente ao flip flop apresentado anteriormente, utilizou-se dos artifícios de processos para resolver a questão de maneira mais rápida. Obteve-se então o seguinte código.

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity D_FF_subida is
        PORT( D,CLOCK: in bit;
        Q: out bit
        );
```

```
end D_FF_subida;

architecture behavioral of D_FF_subida is
        begin
        process(CLOCK)
        begin
                if(CLOCK ='1' and CLOCK'EVENT) then
                Q <= D;
        end if;
        end process;
end behavioral;
```

Dessa forma, obteve-se ondas quadradas condizentes, conforme está expresso a seguir, e o esquemático no fim deste documento.

Figura 16: Ondas do flip flop D por borda de subida



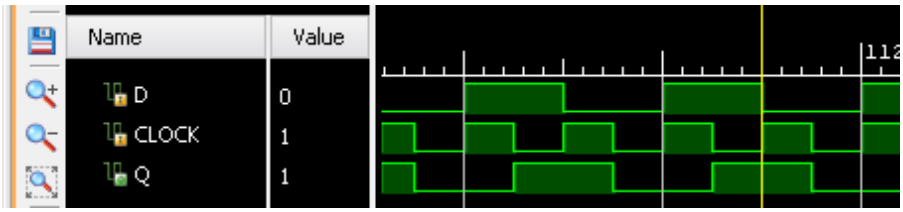
Do mesmo modo, como foi dito anteriormente, para mudar essa lógica de maneira que seja ativa durante bordas de descida, basta modificar as condições a cerca da captura de evento. Assim, o código abaixo é semelhante ao apresentado.

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity D_FF_subida is
        PORT( D,CLOCK: in bit;
        Q: out bit
        );
```

```
end D_FF_subida;

architecture behavioral of D_FF_subida is
        begin
        process(CLOCK)
        begin
                if(CLOCK ='0' and CLOCK'EVENT) then
                Q <= D;
        end if;
        end process;
end behavioral;
```

Seguindo a mesma lógica, as ondas obtidas possuem o mesmo formato, tal que :

Figura 17: Ondas do flip flop D por borda de descida



Afim de unir os dois métodos de implementação, precisou-se apenas da seletora implementada anteriormente e a classe principal de controle. Esta, pode ser visualizada a seguir:

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

entity conexao is
        Port ( d : in STD_LOGIC;
        clock : in STD_LOGIC;
        sel : in STD_LOGIC;
        sa : out STD_LOGIC);
end conexao;
```

```
architecture Behavioral of conexao is

component D_FF_subida
        PORT( D,CLOCK: in std_logic;
        Q: out std_logic
        );
end component;

component D_FF_descida
        PORT( D,CLOCK: in std_logic;
        Q: out std_logic
        );
end component;

component Seletora
        Port ( Sel : in STD_LOGIC;
        SS : in STD_LOGIC;
        SD : in STD_LOGIC;
        S : out STD_LOGIC);
end component;

SIGNAL S1 : std_logic;
SIGNAL S2 : std_logic;

begin

descida : D_FF_descida port map(D =>d ,CLOCK => clock, Q=>S1 );
subida : D_FF_subida port map(D =>d ,CLOCK => clock, Q=>S2);
sele : Seletora port map(Sel => sel, SS => S2, SD =>S1, S=>sa);

end Behavioral;
```

Para unir através de uma variável de seleção , utiliza-se a mesma lógica da implementada anteriormente, por esse motivo, o código não será apresentado, entretanto, o

esquemático do mesmo pode ser visto ao fim deste documento.

2.2.1.3 Flip Flop T

O flip-flop do tipo T funciona aparentemente de maneira semelhante ao flip flop do tipo D, mas possui alguma peculiaridade, como pode ser visto na tabela verdade a seguir.

Figura 18: Tabela verdade do flip - flop T

T	Q	C	Q^*	
0	0	↓	0	Hold
0	1	↓	1	
1	0	↓	1	Toggle
1	1	↓	0	

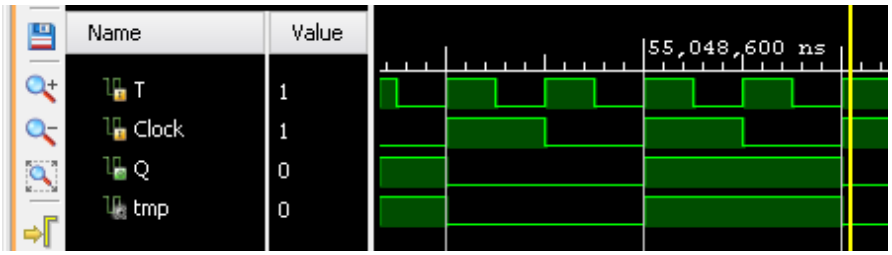
Dessa forma, obteve-se o seguinte código para a borda de subida.

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

entity T_FF is
        port( T: in bit;
        Clock: in bit;
        Q: out bit);
end T_FF;

architecture Behavioral of T_FF is
        signal tmp: bit;
        begin
        process (Clock)
                begin
                if Clock'event and Clock='1' then
                        if T='0' then
                                tmp <= tmp;
```

```
            elsif T='1' then
                tmp <= not (tmp);
            end if;
        end if;
    end process;
    Q <= tmp;
end Behavioral;
```

Dessa forma, as ondas obtidas foram as seguintes:

Figura 19: Ondas do flip flop T



Através deste obteve-se os esquemáticos constantes no fim deste documento e também os códigos para borda de descida e a união por meio da seletora, tal que se segue:

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

entity T_FF is
    port( T: in bit;
    Clock: in bit;
    Q: out bit);
end T_FF;

architecture Behavioral of T_FF is
    signal tmp: bit;
    begin
    process (Clock)
        begin
```

```
            if Clock'event and Clock='0' then
                if T='0' then
                    tmp <= tmp;
                elsif T='1' then
                    tmp <= not (tmp);
                end if;
            end if;
        end process;
        Q <= tmp;
    end Behavioral;
```

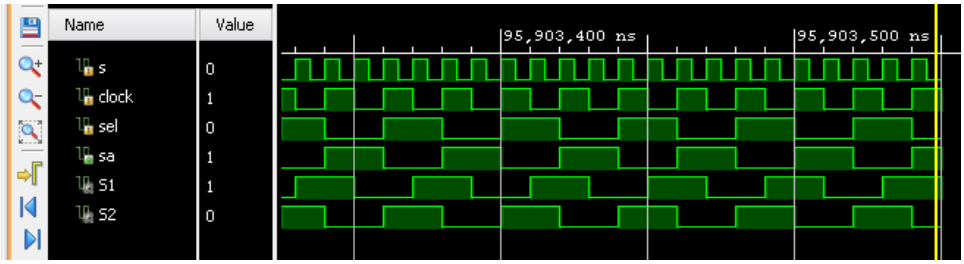
Para a união aproveitou-se a seletora implementada anteriormente e conectou-se os componentes por meio do código a seguir:

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

entity conexao is
   Port ( s : in bit;
        clock : in bit;
        sel : in bit;
        sa : out bit);
end conexao;

architecture Behavioral of conexao is

component Seletora
   Port ( Sel : in bit;
        SS : in bit;
        SD : in bit;
        S : out bit);
end component;

component T_FF_DESCIDA
```

```
    port( T: in bit;
          CLK: in bit;
          Q: out bit
         );
end component;

component T_FF_SUBIDA
    port( T: in bit;
          CLK: in bit;
          Q: out bit
         );
end component;

SIGNAL S1 : bit;
SIGNAL S2 : bit;

begin

descida : T_FF_DESCIDA port map(T => s , CLK => clock, Q=>S1);
subida : T_FF_SUBIDA port map(T =>s , CLK => Clock, Q=>S2);
sele : Seletora port map(Sel => sel, SS => S2, SD =>S1, S=>sa);

end Behavioral;
```

Obtendo, após a simulação as ondas abaixo:

Figura 20: Ondas do flip flop T



2.2.2 Registrador de deslocamento de 8 bits

A segunda atividade experimental diz respeito à um registrador de deslocamento de 4 bits. Esse tipo de implementação é feita utilizando-se 4 flipflops combinados, pois cada flip flop consegue armazenar um bit. Dessa maneira, criou-se um componente do flip-flop do tipo D, por ser mais simples e por meio de uma classe principal multiplicou-se e realizou-se as combinações necessárias. O código para o flip flop do tipo D está expresso a seguir.

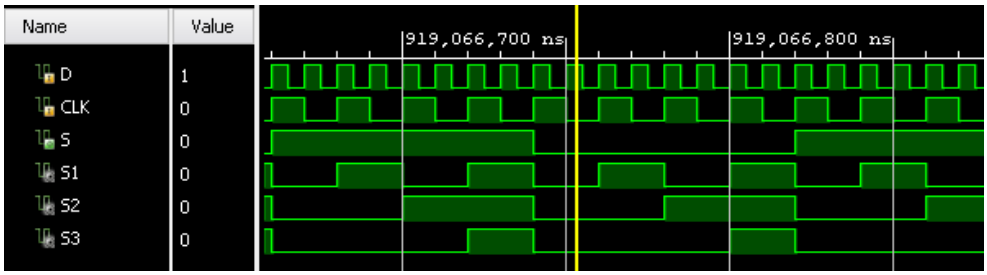
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FFD is
   port( T: in bit;
      Clock: in bit;
      Q: out bit);
end FFD;

architecture Behavioral of FFD is
   signal tmp: bit;
   begin
   process (Clock)
      begin
      if Clock'event and Clock='1' then
         if T='0' then
          tmp <= tmp;
         elsif T='1' then
          tmp <= not (tmp);
         end if;
      end if;
   end process;
   Q <= tmp;
end Behavioral;
```

A conexão foi realizada conforme o código a seguir

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


entity registradordeslocador4 is
    Port ( D : in bit;
           CLK : in bit;
           S : out bit);
end registradordeslocador4;

architecture Behavioral of registradordeslocador4 is

component FFD is
    port( T: in bit;
          Clock: in bit;
          Q: out bit);
end component;

signal S1 : bit;
signal S2 : bit;
signal S3 : bit;

begin

FFD0 : FFD port map(T => D, Clock => CLK, Q=>S1);
FFD1 : FFD port map(T => S1, Clock => CLK, Q=>S2);
FFD2 : FFD port map(T => S2, Clock => CLK, Q=>S3);
FFD3 : FFD port map(T => S3, Clock => CLK, Q=>S);

end Behavioral;
```

Com a simulação obteve-se a onda representada abaixo.O esquemático localiza-se no fim deste documento.

Figura 21: Ondas do registrador de deslocamento



Para demonstrar o deslocamento dos bits é necessário implementar um multiplexador e um codificador para displays de 7 segmentos, de modo que cada um destes represente um número das 4 possíveis saídas. O efeito disto é que através de cada ativação do clock obtém-se um deslocamento dentro do vetor.

O divisor de clock implementado posteriormente deu-se segundo o código a seguir :

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

entity divisoclock  is
    Port ( CLK_IN : in std_logic;
           CLK_OUT : out std_logic_vector (1 downto 0));
end divisoclock  ;

architecture Behavioral of divisoclock  is

signal cont : std_logic_vector (19 downto 0) := (others => '0');

begin

process (CLK_IN)

    begin

       if rising_edge (CLK_IN) then
```

```
        cont <= cont + 1;
      end if;

      CLK_OUT (1) <= cont (18);
      CLK_OUT (0) <= cont (19);
   end process;
end Behavioral;
```

Este clock auxiliou na seleção do display a ser acionado, assim, este também utilizou de um multiplexador com 2 seletoras, tal que :

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

entity mux is
   Port (  E: in std_logic_vector (3 downto 0);
           SEL: in std_logic_vector (1 downto 0);
           S: out std_logic);
end mux;

architecture Behavioral of mux is

begin

with SEL select

   S <= E (0) when "00",
           E (1) when "01",
           E (2) when "10",
           E (3) when others;
end Behavioral;
```

Para realizar a seleção do display criou-se um componente seletor para setar os anôdos e cátodos correspondentes, assim :

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

entity seletora is
   Port (  CLK: in std_logic_vector (1 downto 0);
         AN: out std_logic_vector (3 downto 0));
end seletora;

architecture Behavioral of seletora is

begin

with CLK select
   AN <=  "1110" when "00",
         "1101" when "01",
         "1011" when "10",
         "0111" when others;
end Behavioral;
```

Por fim, o decodificador binário para 7 segmentos, para mostrar as informações.

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

entity decodificador is
   Port (  E: in std_logic;
         S: out std_logic_vector (6 downto 0));
end decodificador;

architecture Behavioral of decodificador is

begin
```

```
with E select
   S <= "1000000" when '0',
        "1111001" when others;

end Behavioral;
```

2.2.3 Divisor de sinal de entrada (em 8)

A terceira atividade proposta diz respeito a um divisor de sinal em 8 , isso é feito por meio da associação de 3 flips flops. Dessa maneira, foi necessário criar o componente flipflop e depois ligá-lo . O flip flop implementado foi o JK, conforme explícito no código a seguir.

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity JK_FF is
   PORT( CLK: in BIT;
         Q: out BIT
   );
end JK_FF;

Architecture behavioral of JK_FF is
begin
   PROCESS(CLK)
      variable temp: BIT;
      begin
         if(CLK='1' and CLK'EVENT) then
            temp:= not temp;
         end if;
         Q<=temp;
      end PROCESS;
end behavioral;
```

Utilizou-se uma classe principal para realizar as ligações por meio do port map, assim, obteve-se:

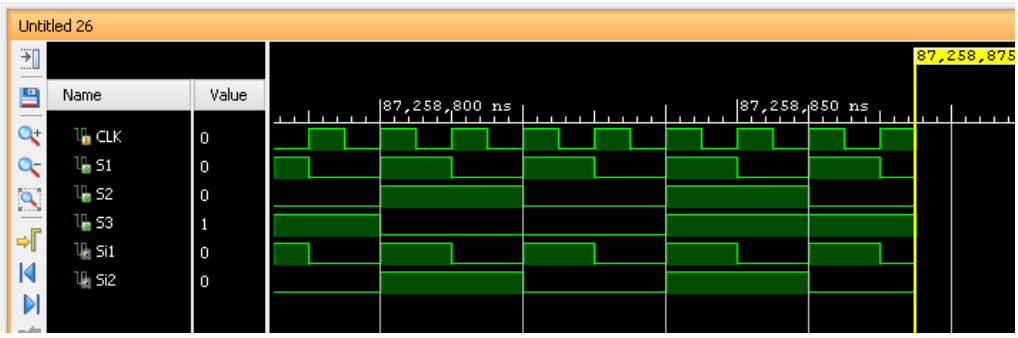
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity divisor8 is
   Port ( CLK : in BIT;
          S1 : out BIT;
          S2 : out BIT;
          S3 : out BIT
          );
end divisor8;

architecture Behavioral of divisor8 is

component JK_FF is
   PORT( CLK: in BIT;
         Q: out BIT
   );
end component;

signal Si1 : BIT;
signal Si2 : BIT;

begin

ff1 : JK_FF port map(CLK => CLK, Q=> Si1);
ff2 : JK_FF port map(CLK => Si1, Q=> Si2);
ff3 : JK_FF port map(CLK => Si2, Q=> S3);

S1<= Si1;
```

S2 <= Si2;

end Behavioral;

Com a simulação obteve-se as ondas abaixo, tal que observando S1, S2 e S3 percebe-se que a divisão do sinal de fato acontece.

Figura 22: Ondas do divisor de sinal



3 DISCUSSÃO

Os circuitos referentes a esse experimento envolvem conceitos relacionados a memória ou armazenamento de dados. Entretanto, não se restringe a ideia de memória que conhecemos, como um armazenador ou retentor de informação por longos períodos. Na realidade, esse tipo de implementação envolve de certa forma o reaproveitamento de um dado, que em linguagem técnica significa retroalimentação ou reutilização de um estado anterior, esse tipo de aplicação vai bem além de flip flops e registradores individualizados, possui ampla aplicação em situações diárias, sendo aplicável em calculadoras, computadores, contadores, entre outros.

A primeira implementação relaciona o conceito de escolha de dados e controle de tempo, isso por que diz respeito à flip flops e explicita a importância do controle na transferência das informações. Dessa maneira, é possível utilizar sincronia entre o recebimento e envio de dados, da mesma forma, a possibilidade de controle por tempo do processo traz inúmeras possibilidades de sistemas a serem montados. Os flip flops, são a mínima estrutura para realização desse tipo de comportamento.

Assim como a maioria das lógicas criadas, há diversas maneiras de manipular uma informação de acordo com o tempo, e foi acerca disso que se fundamentaram os diferentes tipos de flip flops. Na primeira parte desse experimento foi preciso realizar um total de 9 simulações, sendo 3 de cada tipo de flip flop (JK, D e T). As três implementações eram relativas à ativação do clock por borda de subida, borda de descida e possibilidade de escolher entre elas em uma mesma implementação.

A detecção de mudança de nível lógico, de alto para baixo qualifica uma borda de descida e o inverso a borda de subida. As diferenças são mínimas e os programas se comportam de forma semelhante, sendo que todos eles satisfizeram o comportamento esperado. No geral, os resultados são os mesmos, em termos de ondas, e a união dos flip flops de decida e subida, utilizados como componentes, e um multiplexador para a escolha de qual metodologia usar, qualifica a última implementação, dessa forma, não foi preciso realizar grandes modificações.

O segundo experimento, por sua vez, demonstrou uma ideia bem mais elaborada de memória, tratando-se de um registrador de deslocamento de 4 bits. Para realizar tal ideia foram utilizados 4 flip plops do tipo D, devido a sua simplicidade, isto por que cada flip flop corresponde à 1 bit de memória, por assim dizer. Para que se comportasse como um deslocador, ligou-se a todos eles um clock em comum e uma entrada D, sendo que a saída de um flip flop tornava-se a entrada do flip flop seguinte.

Essa implementação no geral não apresentou grandes problemas, foi possível observar pelas respostas dos sinais na implementação que o circuito de fato obteve a resposta esperada . Para realizar a demonstração dinâmica dessa mudança de posições procurou-se demonstrar utilizando um display de 7 segmentos que dada a saída de cada flip-flop utilizava um decodificador para mostrar no display. Entretanto, para acender os 4 ao mesmo tempo foi preciso utilizar uma seletora ou multiplexador em conjunto com um divisor de clock para fornecer o tempo entre o evento de um display apagar e o outro acender.

No terceiro e último experimento apresentou-se a ideia de um divisor de frequência utilizando flip flops, objetivava-se dividí-la em 8, para obter o número de flips flops a serem utilizados, basta fazer 2^(N-1) = 8 , constatando que são necessários 3 flipflops. Cada flip flop divide a frequência anterior por 2 até que se obtém a frequência desejada. O flip flop utilizado foi o flip flop JK e para obter o resultado esperado foi necessário apenas conectar os 3 flip flops de maneira que a saída de um constituía o sinal de clock do outro, no geral o resultado foi satisfatório e observou-se pelos formatos de onda que realmente a frequência havia sido divida.

Para cada um dos 12 experimentos foi realizada síntese, implementação, simulação e análise prévia dos resultados, de modo que estes foram satisfatórios na maioria das vezes. Esse tipo de observação trouxe uma noção da capacidade de implementação com as ferramentas estudadas. Isso porque, a união de toda a lógica apresentada representa um poder computacional incrível.

4 CONCLUSÕES

Diante do que foi exposto, nota-se que todas as atividades solicitadas no experimento foram executadas com resultados satisfatórios, ou seja, foi possível simular o circuito responsável pela implementação dos flip flops nas suas diversas aplicações (isolados e combinados para formar registradores e divisores de frequência) no software Vivado, logo, o objetivo do trabalho foi alcançado, contudo, sob muitas limitações devido a combinação do registrador com o decodificador para display de 7 segmentos, pois a implementação na placa apresentou sérios problemas, além disso, o conteúdo relativo a flip flop foi desafiador devido ao fato da prática não acompanhar as aulas teóricas. Além disso, é preciso citar o mau

funcionamento de alguns computadores do laboratório, a demora na compilação dos códigos, por exemplo. Seria mais proveitoso se as turmas fossem menores para maior aproveitamento do professor e monitor presentes, além disso, uma orientação prévia sobre os experimento também poderiam sanar eventuais dúvidas e tornar o processo de implementação mais eficiente.

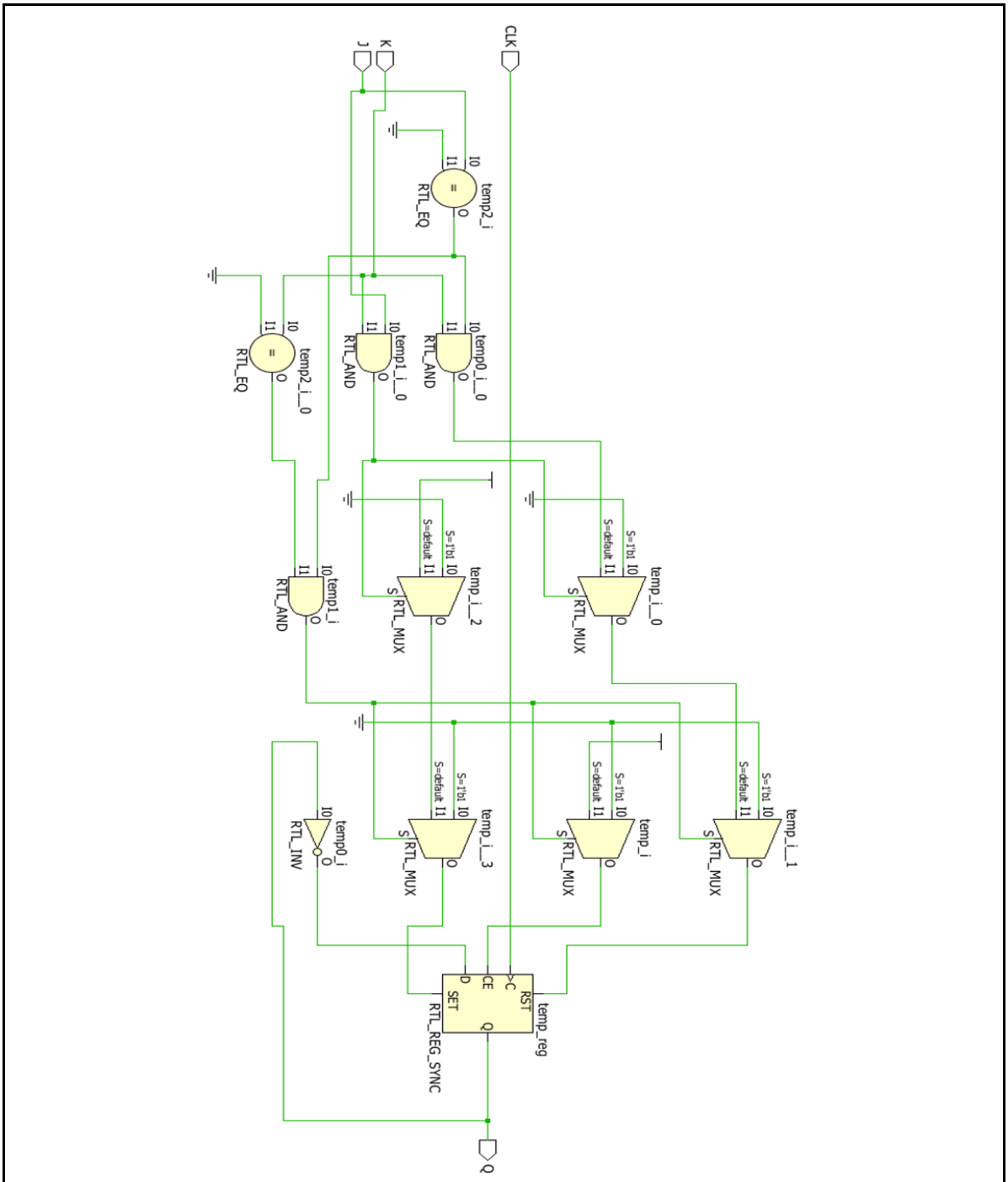
REFERÊNCIAS BIBLIOGRÁFICAS

[1] BAÚ, N. **Apostila de Eletrônica Digital – Flip-Flops**, CEFET/SC, 1999.

[2] FLOYD, Thomas. Portas lógicas. In _____. **Sistemas digitais: fundamentos e aplicações**. Bookman Editora, 2009. Cap. 7, p. 386-410.

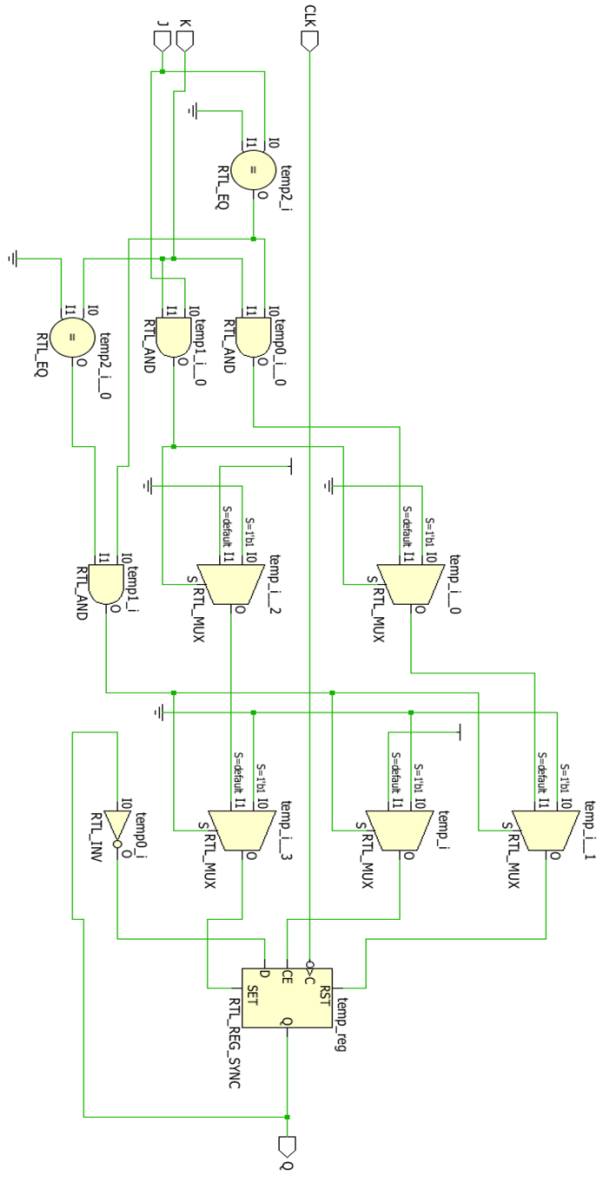
DIAGRAMAS ESQUEMÁTICOS

Figura 23: Flip Flop JK com borda de subida



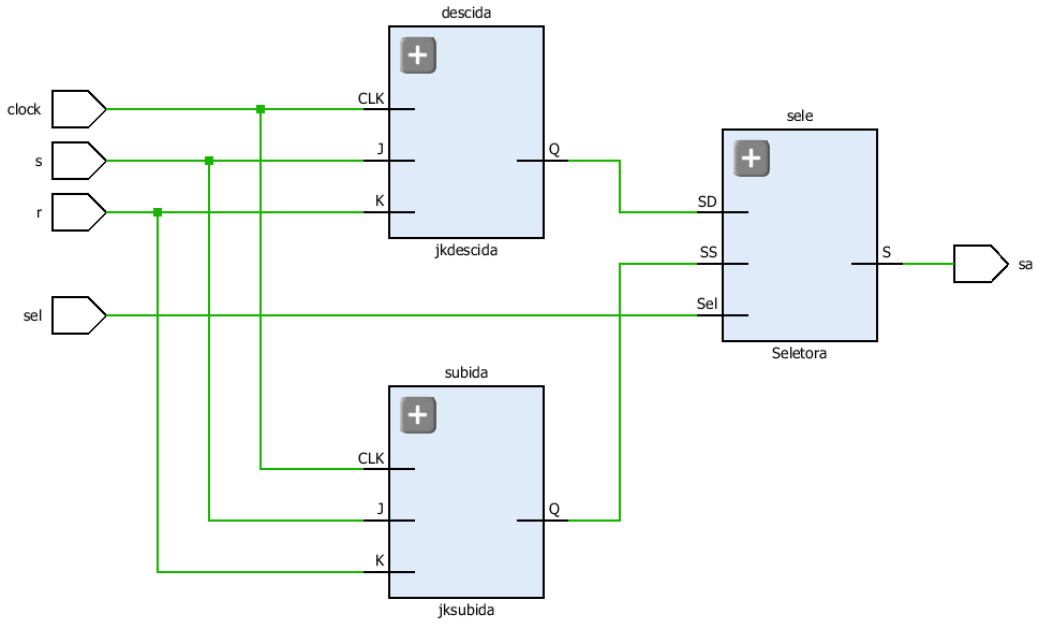
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	**Revisado por:**	Ana Paula Gomes de Matos

Figura 24: Flip Flop JK com borda de descida



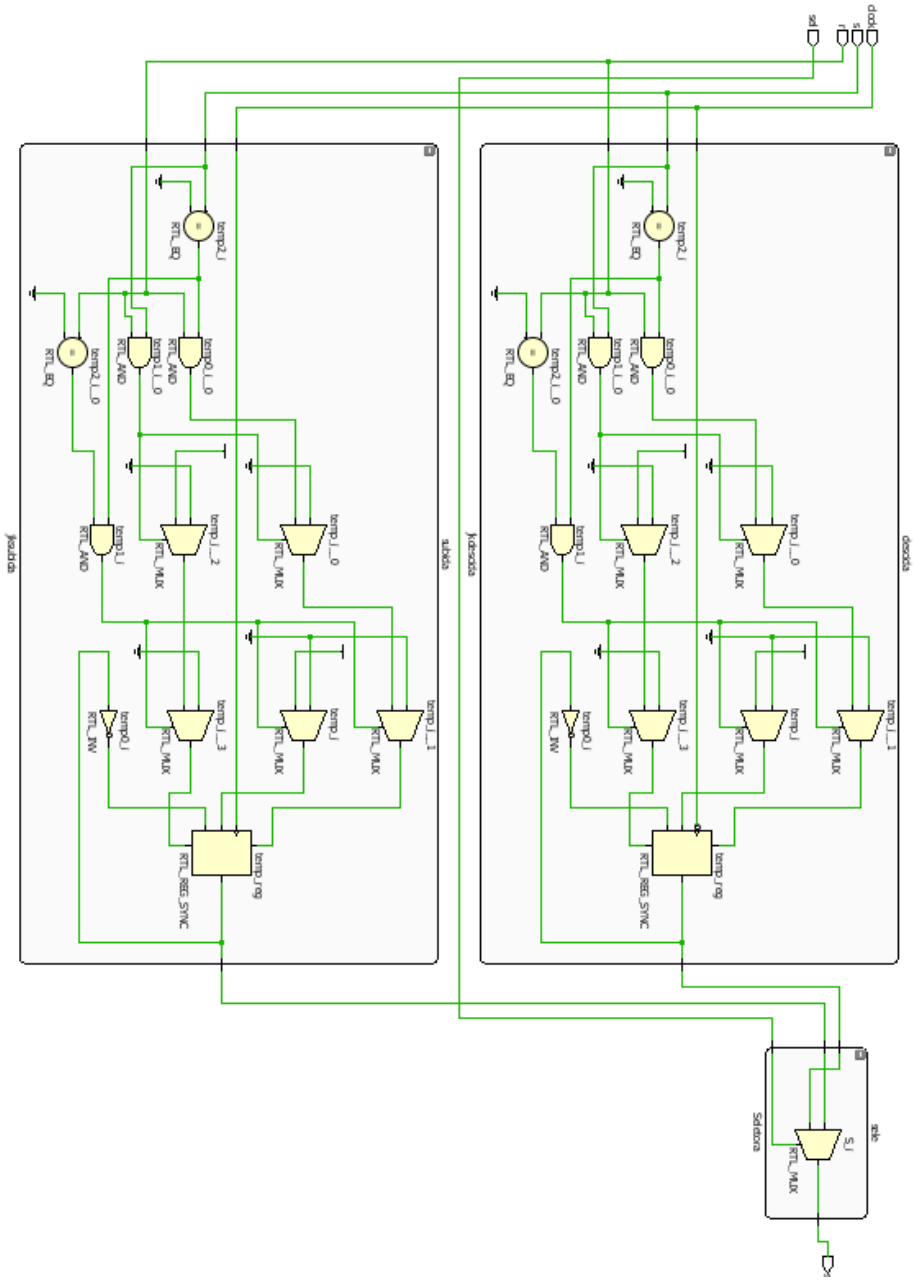
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 25: Flip Flop JK com seletora (esquemático em blocos)



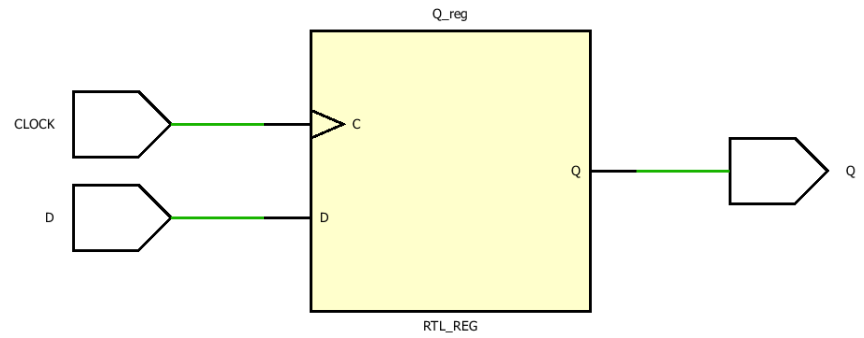
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 26: Flip Flop JK com seletora



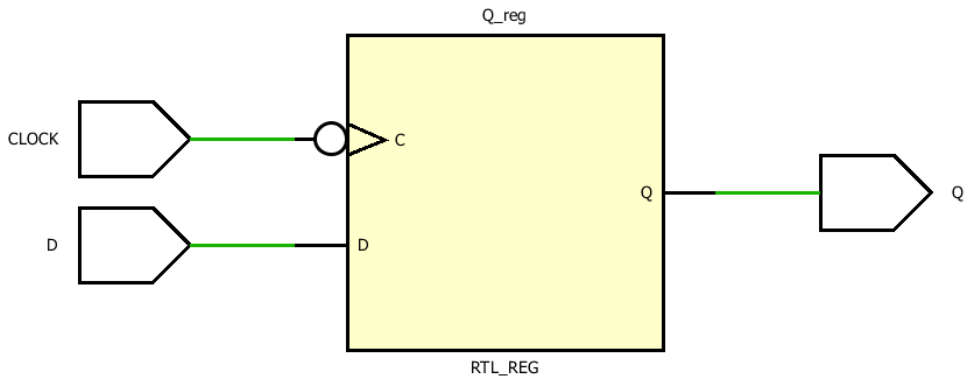
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 27: Flip Flop D borda de subida



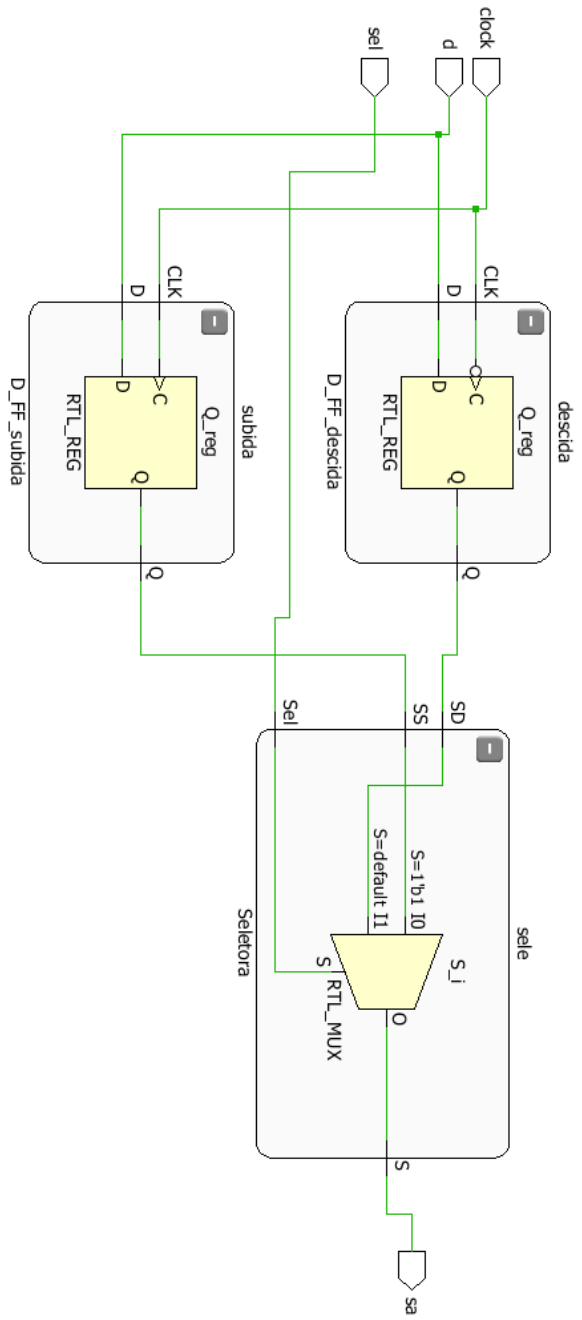
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 28: Flip Flop D borda de descida



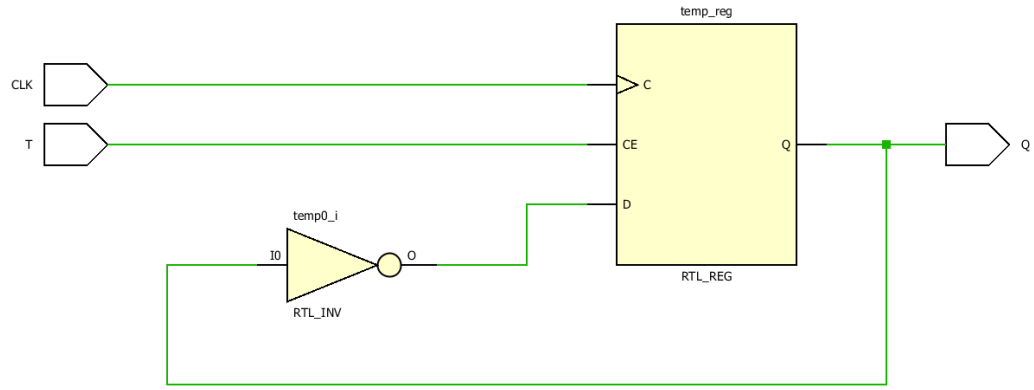
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 29: Flip Flop D com seletora



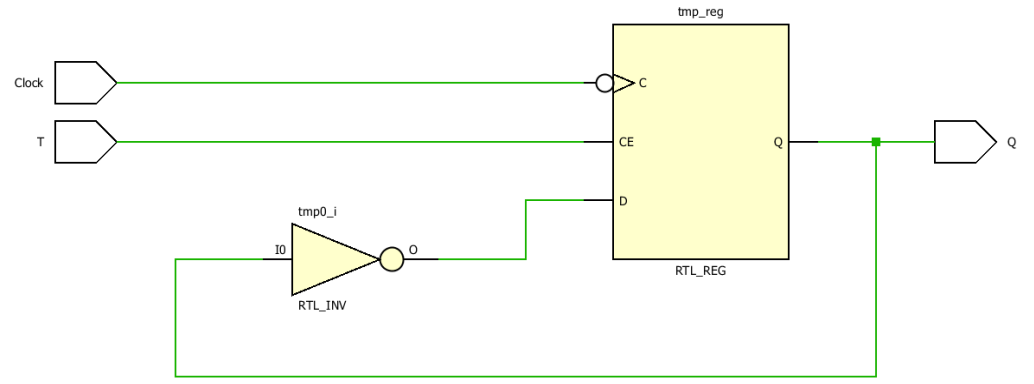
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	**Revisado por:**	Ana Paula Gomes de Matos

Figura 30: Flip Flop T com borda de subida



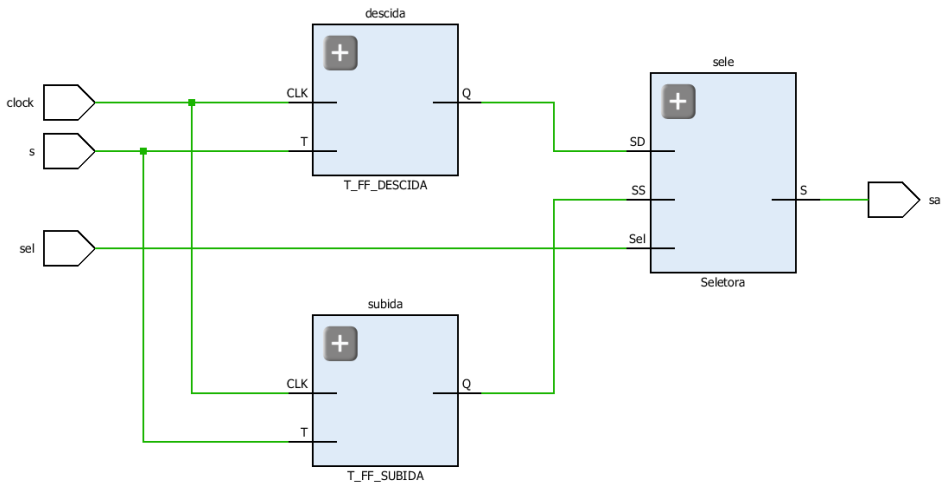
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 31: Flip Flop T borda de descida



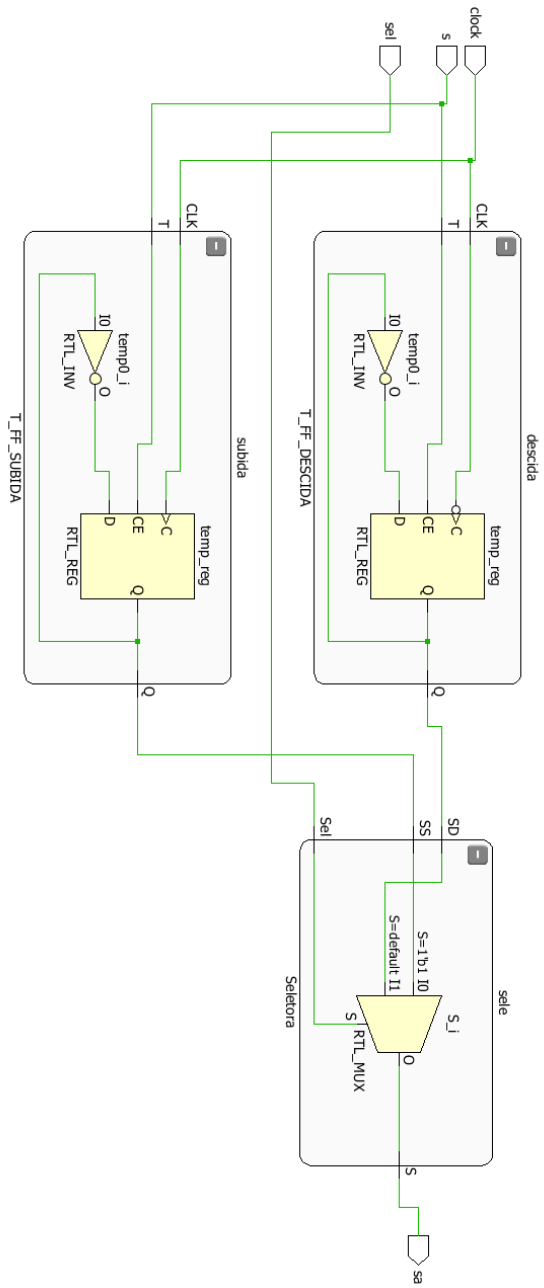
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 32: Flip Flop T com seletora - Em bloco



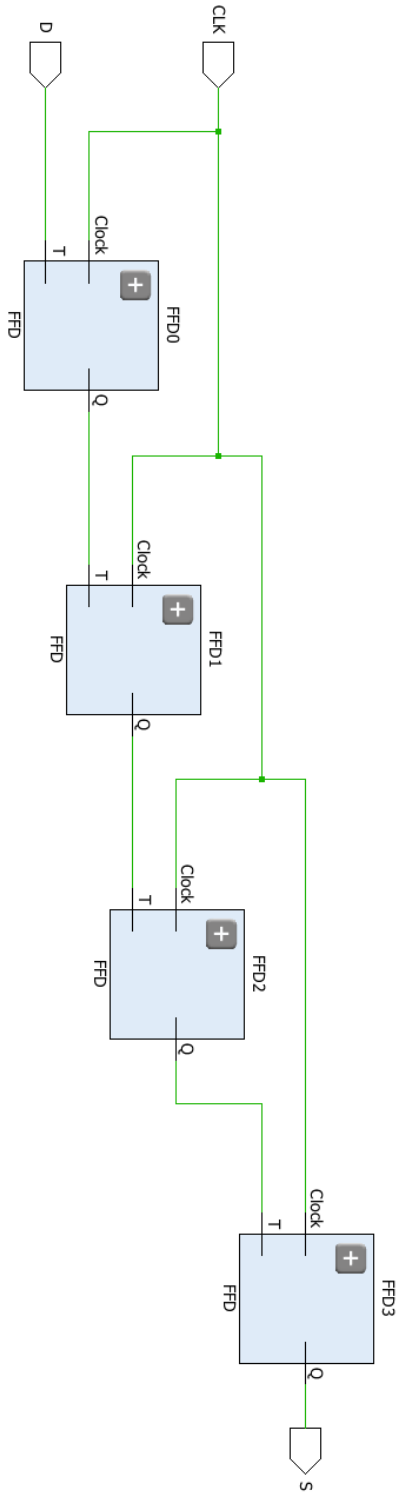
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 33: Flip Flop T com seletora



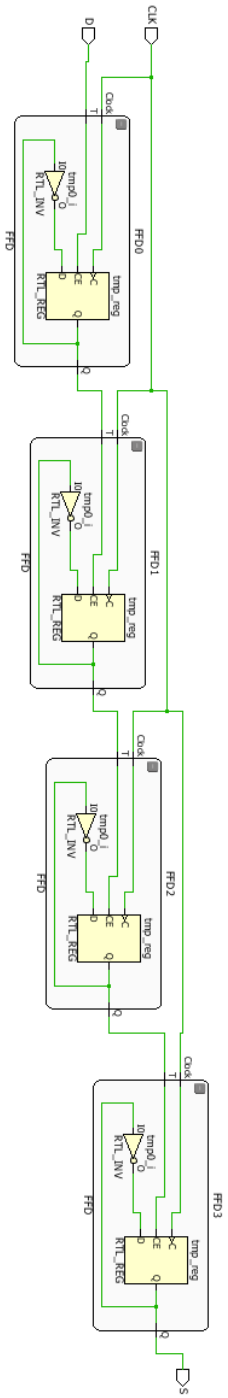
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 34: Registrador de deslocamento – Diagrama em blocos



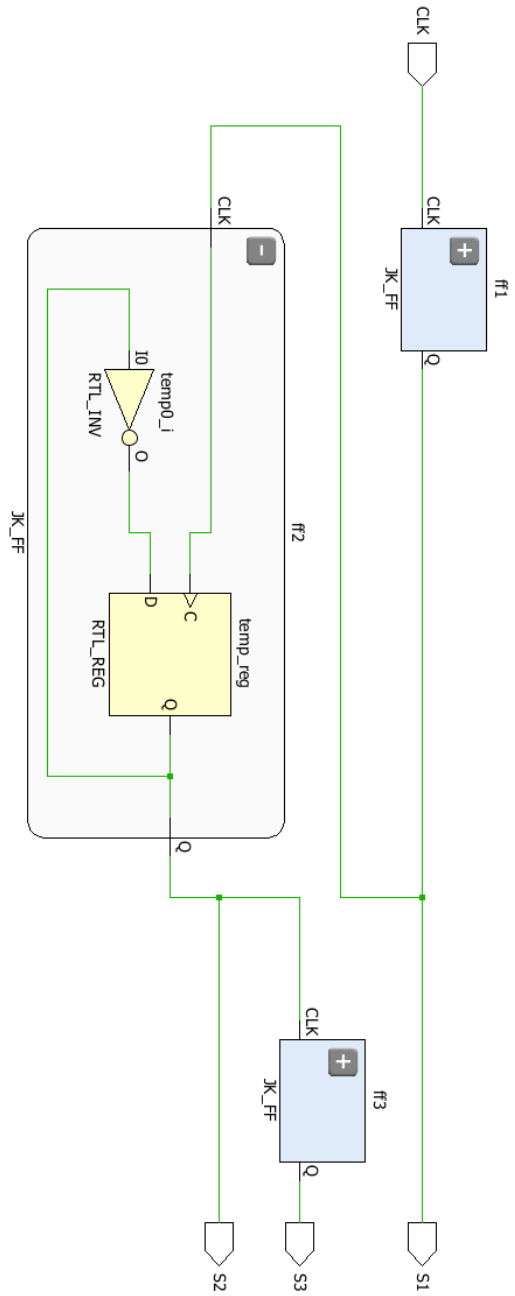
UnB - FGA	**Prancha 1**	**Versão**	**Data**	**Elaborador por:**	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	**Revisado por:**	Ana Paula Gomes de Matos

Figura 35: Registrador de deslocamento



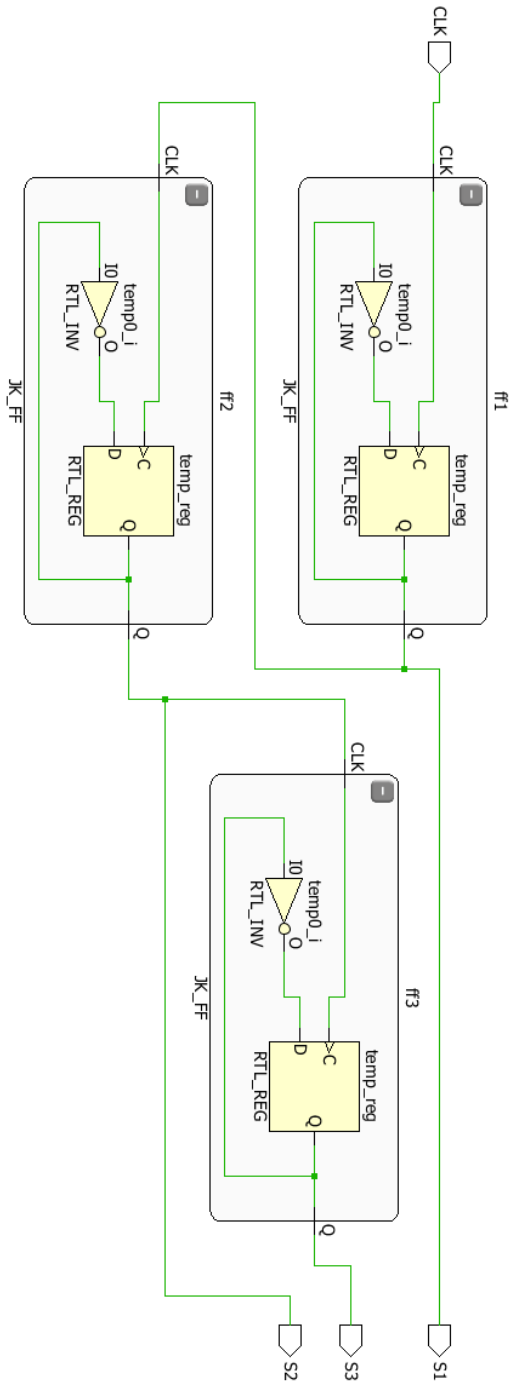
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 36: Esquemático em blocos - Divisor de sinal



UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 37: Circuito Divisor de sinal



UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	Flip Flops	1.0	26/04/2017	Revisado por:	Ana Paula Gomes de Matos