Departamento de Ciência da Computação

Prof. Bruno de Abreu Silva

GCC260 – Laboratório de Circuitos Digitais

Flip-flops e registradores em Verilog

1. Objetivos

- Aprender sobre circuitos sequenciais básicos muito comuns, como Flip-flops do tipo D, SR, JK e T e registradores;
- Aprender estruturas básicas em Verilog para circuitos sequenciais comuns;
- Aprender conceitos e boas práticas de projetos envolvendo Flip-flops.

2. Introdução

Os circuitos digitais podem ser divididos em dois tipos: circuitos combinacionais e circuitos sequenciais. Os circuitos combinacionais são circuitos que não possuem estruturas de memória e a saída depende apenas dos valores atuais das entradas do circuito. Exemplos de circuitos combinacionais são circuitos que são projetados por meio de equações booleanas e simplificação por mapa de Karnaugh, multiplexadores, demultiplexadores, codificadores, decodificadores, comparadores de magnitude, unidade lógica e aritmética, conversores de código. Por outro lado, os circuitos sequenciais são circuitos em que a saída é definida não só pelos valores atuais das entradas, mas também pelos valores anteriores. Sendo assim, para que o circuito armazene os valores anteriores das entradas, é necessário haver alguma estrutura de memória nesses circuitos. E a estrutura básica de memória presente em circuitos digitais é o **Flip-Flop (FF)**. Um FF é capaz de armazenar um único bit e, por isso, para armazenar mais de um bit é necessário um conjunto de FF, que é chamado de **registrador**. Além dos FFs e dos registradores, outros exemplos de circuitos sequenciais básicos são os contadores e divisores de frequência, registradores de deslocamento e máquinas de estados.

3. Flip-flops

Nesta prática, aprenderemos como implementar em Verilog diversos tipos diferentes de FFs e também registradores. A seguir serão apresentados os símbolos, as tabelas-verdade e os códigos dos flip-flops D, SR, JK e T,

respectivamente, começando pela explicação do funcionamento das entradas assíncronas.

3.1. Entradas assíncronas PRESET e RESET

As entradas assíncronas são utilizadas para alterar a saída Q do FF independente do pulso de clock. E muitas vezes são ativadas quando possuem o valor 0. Assim, para setar a saída Q para 1, envia-se o sinal 0 para a entrada PRESET. E, para resetar a saída Q para 0, envia-se o sinal 0 para a entrada RESET. Quando não são ativadas, as entradas PRESET e RESET devem permanecer em 1.

3.2. Flip-flop D

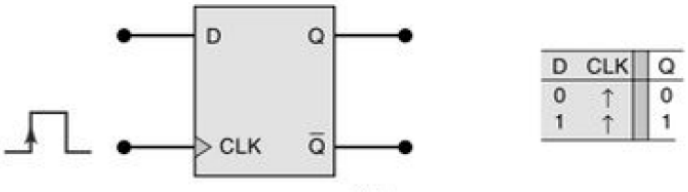
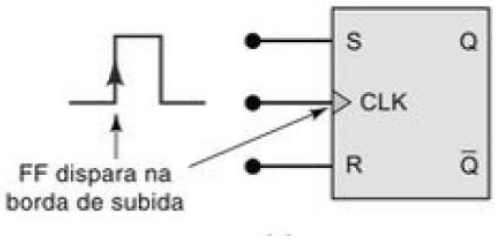


Figura 1: FFD.

```
module FFD(
	input wire clk, rstn, prstn, en, D,
	output reg Q
);

always@(posedge clk, negedge rstn, negedge prstn)
begin
	if(!rstn)
		Q <= 1'b0;
	else if(!prstn)
		Q <= 1'b1;
	else if(en)
		Q <= D;
end
endmodule
```

3.3. Flip-flop SR



Entradas			Saída
S	R	CLK	Q
0	0	↑	Q_0 (não muda)
1	0	↑	1
0	1	↑	0
1	1	↑	Ambíguo

Q_0 é o nível de saída anterior a ↑ de CLK.
↓ de CLK não produz mudança em Q.

Figura 2: FFSR.

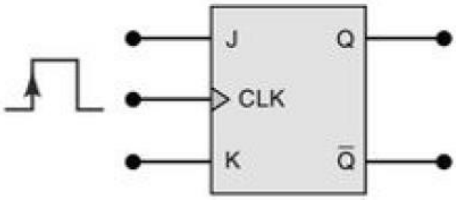
```
module FFSR(
	input wire clk, rstn, prstn, en, S, R,
	output reg Q
);

always@(posedge clk, negedge rstn, negedge prstn)
begin
	if(!rstn)
		Q <= 1'b0;
	else if(!prstn)
		Q <= 1'b1;
	else if(en)
		case({S,R})
			2'b00: Q <= Q;
			2'b01: Q <= 1'b0;
			2'b10: Q <= 1'b1;
			2'b11: Q <= 1'bx;
		Endcase

end

endmodule
```

3.4. Flip-flop JK



J	K	CLK	Q
0	0	↑	Q_0 (não muda)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q_0}$ (comuta)

Figura 3: FFJK.

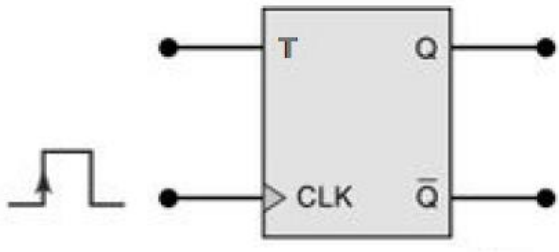
```
module FFJK(
	input wire clk, rstn, prstn, en, J, K,
	output reg Q
);

always@(posedge clk, negedge rstn, negedge prstn)
begin
	if(!rstn)
		Q <= 1'b0;
	else if(!prstn)
		Q <= 1'b1;
	else if(en)
		case({J,K})
			2'b00: Q <= Q;
			2'b01: Q <= 1'b0;
			2'b10: Q <= 1'b1;
			2'b11: Q <= ~Q;
		endcase

end

endmodule
```

3.5. Flip-flop T



T	CLK	Q
0	↑	Q_0 (não muda)
1	↑	$\overline{Q_0}$ (comuta)

Figura 4: FFT.

```
module FFT(
	input wire clk, rstn, prstn, en, T,
	output reg Q
);

always@(posedge clk, negedge rstn, negedge prstn)
begin
	if(!rstn)
		Q <= 1'b0;
	else if(!prstn)
		Q <= 1'b1;
	else if(en)
		Q <= (T) ? ~Q : Q;

end

endmodule
```

4. Registradores

Como já mencionado anteriormente, um registrador nada mais é do que um conjunto de FFs. São muito utilizados para armazenamento de dados. Eles podem possuir diferentes formas de entrada e saída de dados, variando entre serial e paralela. Normalmente, o FF do tipo D é usado como base para a construção de registradores.

Em Verilog, o mesmo código usado para o FFD pode ser usado para implementar um registrador simples. A única diferença é que, em vez de a entrada D e a saída Q serem de apenas um bit, serão declarados como vetores de bits. O código abaixo ilustra uma implementação de um registrador de 32 bits.

```
module Register
#(
	parameter N = 32
)
(
	input wire clk, rstn, prstn, en,
	input wire [N-1:0] D,
	output reg [N-1:0] Q
);

always@(posedge clk, negedge rstn, negedge prstn)
begin
	if(!rstn)
		Q <= {N{1'b0}};
	else if(!prstn)
		Q <= {N{1'b1}};
	else if(en)
		Q <= D;

end

endmodule
```

4.1. Registrador de deslocamento

Além da função de armazenamento de dados, os registradores também são úteis para realizar movimentação de dados. E um dos tipos de movimentação de dados consiste em deslocar os bits algumas casas para a direita ou para a esquerda. A cada pulso de clock, os bits são deslocados uma casa para a direita ou esquerda. Esta é a função de um registrador de deslocamento.

A Figura 5 apresenta o funcionamento de um registrador de deslocamento que pode fazer entrada paralela de dados (todos os bits são armazenados em um único ciclo de clock) e saída paralela (todos os bits armazenados podem ser acessados em um único ciclo de clock). Além disso, a função de deslocamento é realizada uma casa para a direita a cada pulso de clock e de forma circular, ou seja, o bit mais à direita é jogado para a posição mais à esquerda quando ocorre o deslocamento.

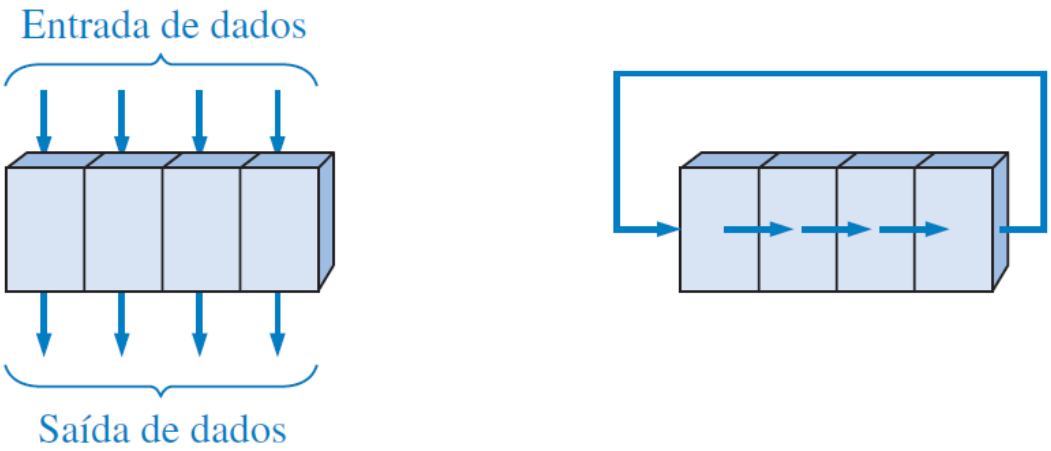


Figura 5: Registrador de deslocamento circular para a direita com entrada e saída de dados paralelas.

O código a seguir, implementa um registrador de deslocamento (*Shift Register*) de 6 bits que faz o deslocamento circular para a direita quando a entrada *enable* (*en*) vale 1. A entrada de *reset* (*rstn*) é acionada em 0 e atribui 1 a todos os bits do registrador. A entrada *loadn*, quando vale 0, permite o carregamento paralelo de dados pela entrada *D* do registrador. Por fim, a saída Q fornece o acesso paralelo ao valor armazenado no registrador.

```
module ShiftRegister
#(
	parameter N = 6
)
(
	input wire clk, rstn, loadn, en,
	input wire [N-1:0] D,
	output reg [N-1:0] Q
);

always@(posedge clk, negedge rstn, negedge loadn)
begin
	if(!rstn)
		Q <= {N{1'b1}};
	else if(!loadn)
		Q <= D;
	else if(en)
		Q <= {Q[0], Q[N-1:1]};

end

endmodule
```

5. Visão geral do projeto Pratica8

A Figura 6 apresenta um diagrama em alto nível simplificado do circuito implementado no projeto Pratica8. Os flips-flops D, SR, JK e T estão implementados assim como um registrador de deslocamento de 6 bits. Também foi implementada toda a lógica necessária para preset e reset assíncrono, circuito de enable e clock. Todas as entradas e saídas utilizadas do kit DE10-Lite estão representadas na Figura.

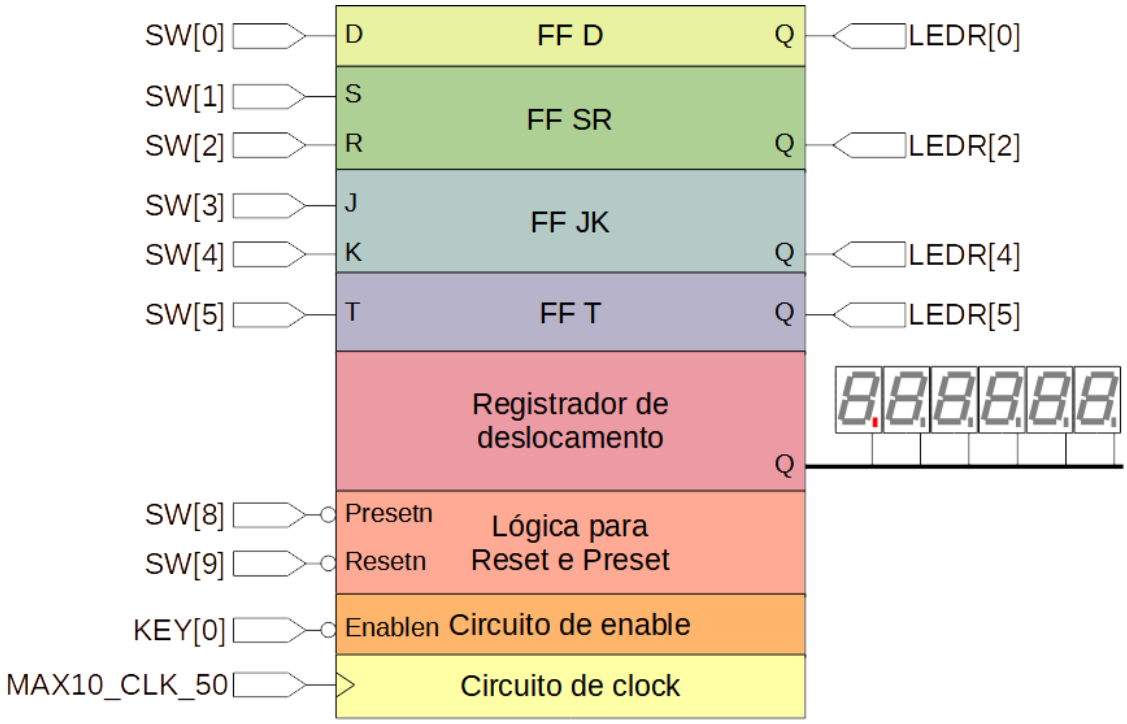


Figura 6: Visão geral do circuito da Prática 8.

As entradas e saídas dos flip-flops estão ligadas da seguinte maneira:

- A entrada D do FF D está ligada na SW[0] e a saída Q em LEDR[0];
- Para o FF SR, a entrada S está em SW[1], R em SW[2] e Q no LEDR[2];
- Para o FF JK, a entrada J está em SW[3], K em SW[4] e Q no LEDR[4];
- A entrada T do FF T em SW[5] e a saída Q está ligada no LEDR[5];
- Todos os FFs têm entrada assíncrona PRESETN, ligadas em SW[8];
- Todos os FFs têm entrada assíncrona RESETN, ligadas em SW[9];
- Para o FF “salvar” o valor de sua saída conforme o comando das entradas, a entrada enable deve ser acionada, por meio da chave KEY[0].

O registrador de deslocamento possui 6 bits. Também está conectado ao PRESETN e RESETN pelas chaves SW[8] e SW[9], respectivamente. Ao se acionar o RESETN, o registrador é zerado (saída Q = 6’b000000). Ao se acionar o PRESETN, o registrador recebe seu valor padrão (saída Q = 6’b011111) acendendo apenas um ponto decimal à esquerda. Para o registrador deslocar os bits uma casa para a direita, o enable deve ser ativado por meio da KEY[0].

6. Passo-a-passo

As seguintes etapas devem ser seguidas para se concluir esta prática:

1. Baixar o arquivo Pratica8.qar em https://github.com/brabreus/GCC260-UFLA/tree/main/Pratica8 e salvar na Área de Trabalho do computador;
2. Clicar duas vezes no arquivo para abri-lo no Quartus Prime (o Quartus irá solicitar a criação de uma pasta com nome *Pratica8_restored* na Área de Trabalho para descompactar o projeto, clique em OK);
3. Após o projeto ser descompactado e aberto, faça a compilação do mesmo (*Processing->Start Compilation*);
4. Programe o kit do laboratório *Tools->Programmer* (lembre-se de configurar o *Hardware setup* para *USB-Blaster* e de clicar em *Start*);
5. Em conjunto com as explicações e orientações do Professor:
   a) Verifique o funcionamento do circuito testando cada FF e o registrador de deslocamento;
   b) Entenda os códigos dos FFs D, SR, JK e T;
   c) Entenda o código do registrador de deslocamento;
6. Atente-se para a explicação do Professor sobre conceitos importantes envolvendo projetos de circuitos com flip-flops;
   a) Estude e entenda os códigos dos sincronizadores de entradas assíncronas de dados e de *reset* e *preset*;
   b) Estude e entenda o código Pratica8.v;
7. Visualize o circuito do projeto acessando *Tools->Netlist Viewers->RTL Viewer*).