Departamento de Ciência da Computação

GCC260 – Laboratório de Circuitos Digitais

Arquitetura Registrador-Acumulador

1. Objetivos

- Aprender a implementar registradores em Verilog;
- Aplicar o uso de registradores no projeto de um processador simples baseado na arquitetura registrador-acumulador.

2. Arquitetura Registrador-Acumulador

Nesta prática, vamos implementar um circuito capaz de realizar operações lógicas e aritméticas, acumulando (salvando) os resultados em um registrador especial chamado de acumulador. O funcionamento é similar ao de uma calculadora, onde o resultado acumulado de cada conta fica visível no *display* e podemos usar o resultado acumulado em uma nova operação. O circuito, que será implementado na DE10-LITE, está representado na Figura 1.

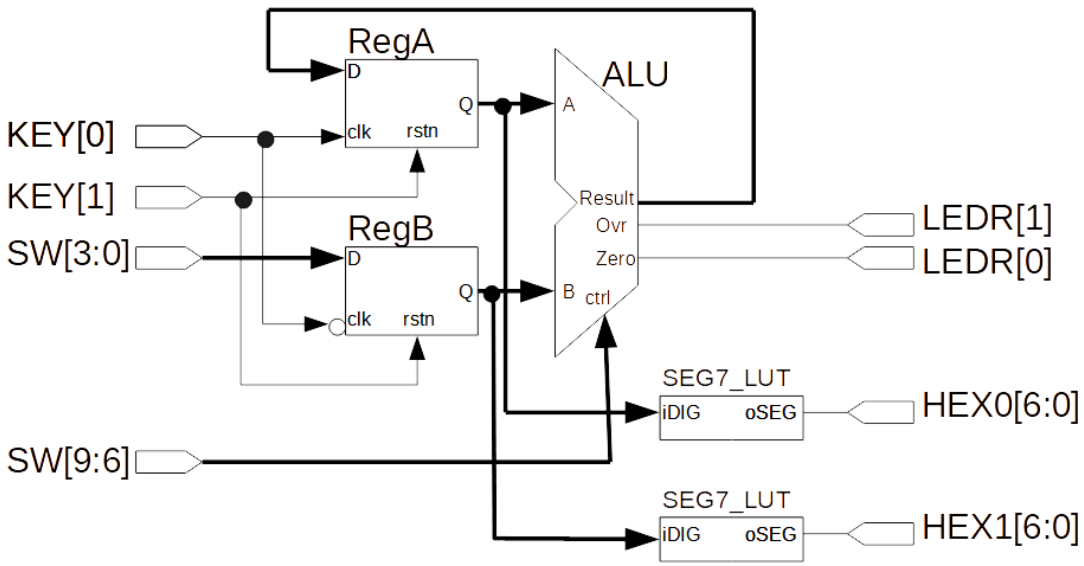

Figura 1: Processador de 4 bits com dois registradores, onde um deles é acumulador. As conexões em negrito representam conexões de 4 bits.

Como pode ser visto na Figura 1, o botão KEY[1] é usado como *reset* geral do circuito, ou seja, quando pressionado, a entrada *rstn* dos registradores recebe o valor 0 e isso faz com que a saída dos dois registradores RegA e RegB passem para zero. O *clock* do circuito é realizado manualmente (embora esta não seja a forma ideal, mas foi feito assim para fins didáticos) por meio do botão KEY[0]. A unidade lógica e aritmética (ALU, do inglês *Arithmetic-Logic Unit*) recebe o valor armazenado nos registradores RegA e RegB e faz a operação cujo código é colocado nas chaves SW[9:6]. Os valores atualmente armazenados em RegA e RegB podem ser visualizados nos *displays* de sete segmentos HEX0 e HEX1, respectivamente. Para converter o valor binário dos registradores para hexadecimal no display de sete segmentos, é usado o decodificador implementado em SEG7_LUT. RegA inicialmente armazena o valor zero. Em seguida, um valor binário para RegB deve ser colocado nas chaves SW[3:0] e um código de operação da ALU deve ser colocado em SW[9:6]. Feito isso, o botão KEY[0] deve ser pressionado. Ao ser pressionado, ele irá gerar uma transição negativa do *clock* (borda de descida) e isso faz com que RegB armazene o valor colocado em sua entrada SW[3:0]. Quase instantaneamente, a ALU realiza a operação codificada em SW[9:6] usando os valores de RegA e RegB e o resultado da conta é enviado para a entrada do RegA. Como o RegA é um registrador acionado pela transição positiva do *clock*, o resultado da conta, colocado em sua entrada, é armazenado quando o usuário deixa de pressionar o botão KEY[0]. O processo se repete para realizar novas operações sobre os dados. Obs: os botões KEY[0] e KEY[1] valem zero quando pressionados.

3. Passo a passo

O passo a passo para implementar e testar este projeto consiste nas seguintes etapas:

- Crie uma nova pasta na Área de Trabalho, chamada PraticaAcumulador;
- Baixe os arquivos dos módulos que serão necessários:
  - SEG7_LUT.v (https://github.com/brabreus/GCC260-UFLA/blob/main/PraticaAcumulador/SEG7_LUT.v);
  - ALU.v (https://github.com/brabreus/GCC260-UFLA/blob/main/PraticaAcumulador/ALU.v) e;
  - Register.v (https://github.com/brabreus/GCC260-UFLA/blob/main/PraticaAcumulador/Register.v)
  - Salve na pasta criada anteriormente;
- Crie um projeto no Quartus Prime, mude o diretório de criação do projeto para a pasta criada anteriormente, adicione os arquivos baixados ao projeto, selecione o *Board* como *Family*: MAX10 e o kit DE10-LITE. Lembre-se de deixar marcada a opção "*Create top-level design file*";
- Entenda os códigos *Register.v* e *ALU.v*;
- Reveja a Figura 1 e complete o código *DE10_LITE_Golden_Top.v* para atender às características pedidas para o circuito;
- Compile o projeto (*Processing->Start Compilation*);

- Use a ferramenta em *Tools->Netlist Viewers->RTL Viewer* para visualizar o desenho gerado pelo seu código verilog e verificar se o projeto está codificado corretamente;
- Conecte a FPGA ao computador;
- Programe a FPGA em *Tools->Programmer*;
- Verifique o funcionamento do circuito na placa realizando diversas operações como soma e subtração. Releia a seção 2 deste documento para relembrar como o circuito funciona para que você consiga testar adequadamente;
- Apresente o projeto funcionando para o Professor e envie para o Campus Virtual.

3. Códigos usados

3.1. Descrição de um registrador usando o Flip-Flop tipo D:

```
// Register.v
module Register(clk, rstn, D, Q);
	input wire clk, rstn;
	input wire [3:0] D;
	output reg [3:0] Q;

	always@(posedge clk, negedge rstn)
	begin
		if(rstn == 0)
			Q <= 4'b0000;
		else
			Q <= D;
	end

endmodule
```

3.2. Descrição da ALU:

```
// ALU.v
module ALU(
	input wire [3:0] A, B,
	input wire [3:0] ctrl,
	output reg [3:0] Result,
	output reg Ovr,
	output wire Zero
);
	always@*
	begin
		case(ctrl)
			4'b0000: {Ovr, Result} = A & B;
			4'b0001: {Ovr, Result} = A | B;
			4'b0010: {Ovr, Result} = A + B;
			4'b0110: {Ovr, Result} = A - B;
			4'b0111: {Ovr, Result} = (A < B)? 1: 0;
			4'b1100: {Ovr, Result} = ~(A | B);
			default: {Ovr, Result} = 0;
		endcase
	end
	assign Zero = (Result == 0);
endmodule
```

3.2. Descrição do SEG7_LUT:

```
module SEG7_LUT      (
       input  wire [3:0]    iDIG,
       output  reg [6:0]    oSEG
);

always @(iDIG)
begin
       case(iDIG)
              4'h0: oSEG = 7'b1000000;
              4'h1: oSEG = 7'b1111001;    // ---t----
              4'h2: oSEG = 7'b0100100;    // |      |
              4'h3: oSEG = 7'b0110000;    // lt    rt
              4'h4: oSEG = 7'b0011001;    // |      |
              4'h5: oSEG = 7'b0010010;    // ---m----
              4'h6: oSEG = 7'b0000010;    // |      |
              4'h7: oSEG = 7'b1111000;    // lb    rb
              4'h8: oSEG = 7'b0000000;    // |      |
              4'h9: oSEG = 7'b0011000;    // ---b----
              4'ha: oSEG = 7'b0001000;
              4'hb: oSEG = 7'b0000011;
              4'hc: oSEG = 7'b1000110;
              4'hd: oSEG = 7'b0100001;
              4'he: oSEG = 7'b0000110;
              4'hf: oSEG = 7'b0001110;
       endcase
end

endmodule
```