Projeto Final
Calculadora Digital com Banco de Registradores

João Vitor Abadio Siqueira, 18/0123394
Pedro Henrique de Brito Agnes, 18/0026305

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 129020 - Laboratório de Circuitos Lógicos - Grupo B9

joaovitorabadio06@gmail.com, pedenite@gmail.com

Abstract. *This experiment aims to build a calculator capable of performing addition and subtraction operations, in addition to these arithmetic operations, the calculator aims to perform load and store operations using a bank of registers.*

Resumo. *Este experimento tem como objetivo construir uma calculadora capaz de realizar as operações de soma e subtração, além dessas operações aritméticas, a calculadora tem como objetivo realizar as operações de load e store utilizando um banco de registradores.*

1. Introdução

A calculadora é um instrumento de extrema importância. Seja em qualquer área das ciências naturais, o uso da calculadora é, em algumas situações, até obrigatório. Uma calculadora tem a capacidade de realizar operações aritmética utilizando circuitos combinacionais aritméticos com números binários, logo, uma das funções das calculadoras é converter esses números em base binária para decimal.

Algumas calculadoras têm em sua implementação recursos mais avançados, como o cálculo de raízes, expoentes e afins. Além das operações de cálculo, algumas calculadoras podem guardar certos resultados requeridos pelo usuário. Para a gravação de um número, utilizamos banco de registradores, estes são capazes de armazenar *bits* em sua memória pois são circuitos sequênciais.

1.1. Objetivos

Implementaremos uma calculadora capaz de realizar cálculos de adição e subtração de até 8 bits em complemento de 2. O resultado deve ser mostrado nos *displays* de 7 segmentos do *kit* de desenvolvimento FPGA D2, a entrada do usuário deve ser feita através de um mini-teclado 4x4 de membrana. Além das operações aritméticas, a calculadora deve realizar as operações de *load* e *store* utilizando 10 bancos de registradores.

1.2. Materiais

Neste experimento foram utilizados os seguintes materiais e equipamentos:

- *Software* Quartus-II v13.0 SPI
- *Kit* de desenvolvimento FPGA DE2
- Mini-teclado matricial 4x4

2. Procedimentos

2.1. Pré-Projeto

A seguir, temos o circuito completo do projeto, em que a parte do pré-projeto está destacada dentro de um quadrado preto:

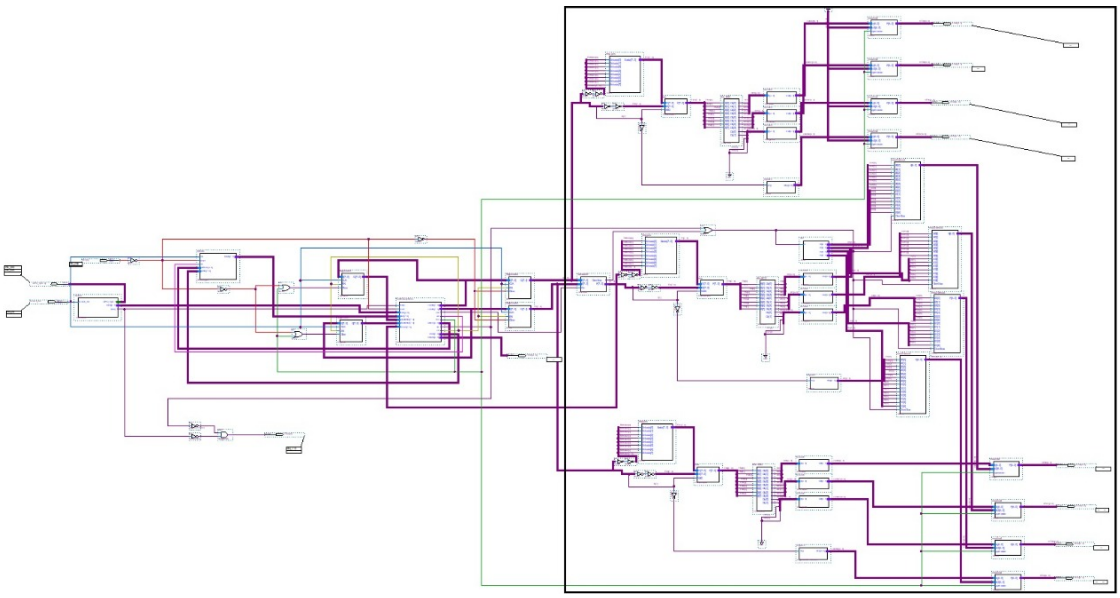

Figura 1. Circuito Completo da Calculadora

De forma resumida, a parte do pré projeto irá receber os valores de A, B, sinal de Cin (*Carry in*) e res para mostrar o resultado ou não. Assim, a partir dos circuitos feitos na etapa de pré-projeto, convertemos o valor transmitido de complemento de 2 para BCD e, assim para decimal nos displays de 7 segmentos. A partir do sinal de res, é definido se deverá mostrar os valores de A e B nos displays ou se irá apagar os displays de A e mostrar o resultado da soma ou subtração nos displays de B.

2.1.1. Somador e Subtrator

Para a implementação do somador, utilizamos o seguinte código em verilog:

```
module full_adder(
        input A, B, Cin,
        output F, Cout
);
assign {Cout, F} = A+B+Cin;

endmodule
```

Este é o código para um full-adder. No projeto, utilizamos oito deles com multiplexadores para criar um somador e subtrator de 8 bits. A operação é definida pelo *carry in* inicial, que se tiver nível lógico 0, funcionará como somador e se for 1, subtrator. Tudo

em complemento de 2. Além disso, temos um detector de *overflow*, que é feito utilizando uma porta XOR com entradas sendo a entrada e saída do último *full-adder*. Este detector irá transmitir um sinal ”avisando”o sistema se houve ou não *overflow*. se houver, uma mensagem de erro será mostrada nos displays quando o sinal res estiver em 1. O circuito descrito:

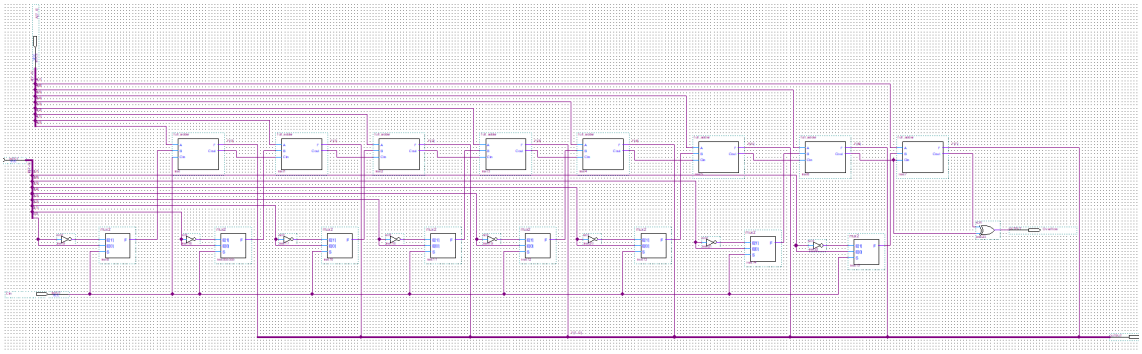

Figura 2. Somador e subtrator de 8 bits

2.2. Teclado 4x4

O módulo responsável por receber os valores do teclado recebe o sinal de *clock*, que será gerado pela FPGA DE2 e tem as entradas da GPIO_1 para receber os sinais vindos do teclado. Desta forma, ele gera o sinal de *ready*, que indica quando uma tecla está pressionada e também gera o sinal que indica a tecla pressionada. Assim, utilizamos ele da seguinte forma:

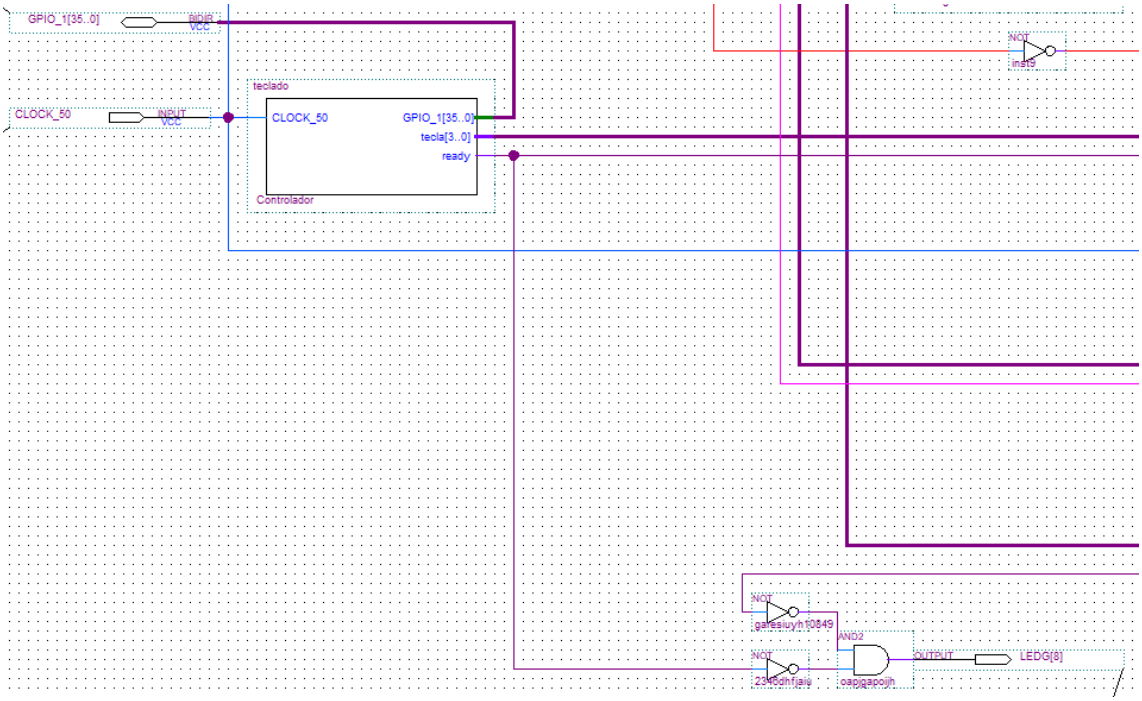

Figura 3. Gerador de Sinal do Teclado

Assim, ligamos as saídas tecla e ready na máquina de estados que será mostrada a seguir,

e o sinal de ready também foi usado para indicar pelo LEDG[8] que está pronto para receber um novo valor do teclado e para mostrar quando uma operação foi concluída com sucesso.

2.3. Load e Store

A operação de *load* utiliza um banco de registradores, o código em SystemVerilog utilizado foi o seguinte:

```
module regbank(
        input logic clk,
        input logic reset,
        input logic we,
        input logic [3:0] address,
        input logic [7:0] wdata,
        output logic [7:0] rdata);

        logic [7:0] bancoreg[9:0];

        always_ff @(posedge clk)
        if(reset) bancoreg <= '{default:0};
        else
        if (we) bancoreg[address] <= wdata;

        assign rdata = bancoreg[address];

endmodule
```

No código, quando reset for igual a 1, todos os bancos são zerados, quando we (*write enable*) for igual a 1, o número contido em wdata será escrito no registrador de endereço address. O código cria 10 registradores com espaço total de 8 bits.

O banco de registradores será utilizado em ambas as operações unárias, no caso de *store*, será pedido ao usuário o número e em seguida o endereço (índice) do banco de registradores em que deseja salvar, o valor não será exibido no *display* de 7 segmentos. Em *load*, será solicitado ao usuário o endereço do banco de registradores em que se deseja resgatar o número, o valor será exibido no *display* de 7 segmentos.

2.4. Máquina de Estados

É responsável por controlar as operações da calculadora. Seu módulo foi utilizado da seguinte forma:

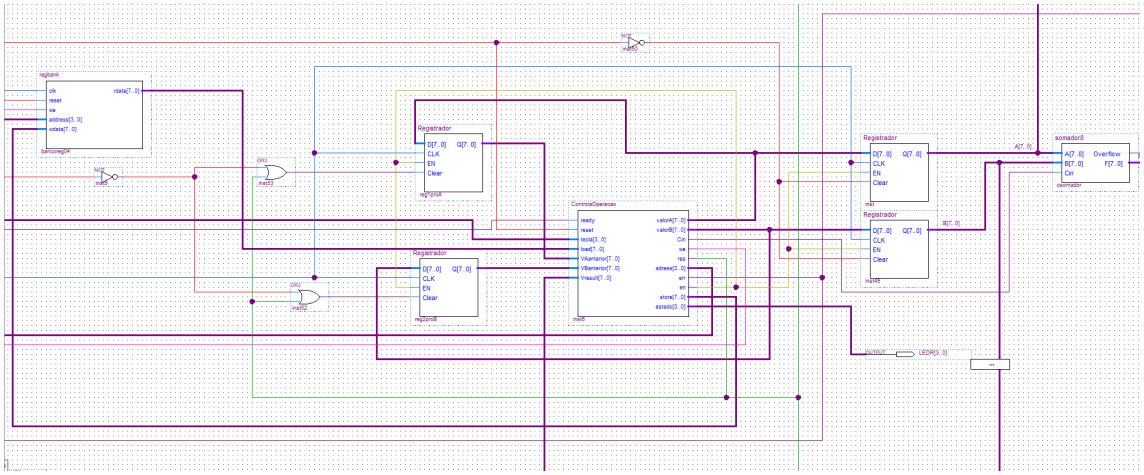

Figura 4. Máquina de Estados Controladora de Operações da Calculadora

Ela foi feita em system verilog, mas, por seu código ter ficado muito extenso, serão mostrados apenas trechos. Primeiramente, nela temos as seguintes entradas e saídas:

```
module ControlaOperacao(
        input logic ready,
        input logic reset,
        input logic [3:0] tecla,
        input logic [7:0] load,
        input logic [7:0] VAanterior, VBanterior, Vresult,
        output logic [7:0] valorA,
        output logic [7:0] valorB,
        output logic Cin,
        output logic we,
        output logic res,
        output logic [3:0] adress,
        output logic err,
        output logic en,
        output logic [7:0] store,
        output logic [3:0] estado
);
```

Na máquina, o sinal de *ready* servirá como *clock* para ela, logo, teremos um pulso de *clock* sempre queuma tecla do teclado for pressionada. Também temos o sinal de reset que é gerado pela chave KEY[0] da FPGA. Ele fará com que a máquina volte ao seu estado inicial e também fará com que todos os registradores do circuito tenham o valor 0 armazenado, assim fazendo com que toda a calculadora seja reiniciada.

A entrada tecla, vinda do teclado 4x4 (2.2) simplesmente indicará qual tecla foi pressionada. Já o *load* vem do banco de registradores (2.3) e serve para a função *load* da calculadora, que carregará o valor salvo no registrador definido pela saída *adress* da máquina que serve como entrada do banco de registradores que indica qual registrador indexado de 0 a 9 foi chamado. O mesmo vale para a saída *store* que no caso vai para a entrada do banco para armazenar um valor em algum registrador indicado por *adress*.

Além disso, temos as entradas VAanterior, em que vem de um registrador que armazerna o último valor de A armazenado por um estado que tem a saída en = 1, a VBanterior que vem de outro registrador que armazena o valor de B da mesma forma que o VAanterior e Vresult que devolve o valor do último resultado para que se possa armazená-lo em um registrador do banco caso seja solicitado.

Agora, as saídas restantes funcionam de forma que valorA vai mandar o valor de A provido da tecla digitada pelo usuário, assim como o valor de B é dado pela saída valorB. Temos também os sinais de Cin e res já explicados anteriormente (secção 2.1.1), e a saída we também já foi explicada anteriormente (secção 2.3), que no caso será ligado na entrada we do banco de registradores. Por fim, temos as saídas err, que transmitirá uma mensagem de erro nos displays, assim como quando temos *overflow* e estados, que é apenas para fins de visualização de qual estado a máquina está.

Em seguida, temos a declaração de estados e a forma como será a transição de estados.

```
typedef enum logic [3:0] {INICIAL, INICIAL_A, INICIAL_B,
 INICIAL_SOMA, INICIAL_SUB, RESULT_SOMA, RESULT_SUB, SOMA,
  SUBTRACAO, STORE_A, STORE_RES_SOMA, STORE_RES_SUB, LOAD_A,
   LOAD_SOMA, LOAD_SUB, ERRO} statetype;
        statetype state, nextstate;

        always_ff @(posedge ready, posedge reset)
                if (reset) state <= INICIAL;
                else state <= nextstate;
```

A partir deste ponto, temos apenas a descrição de circuitos combinacionais que definirão as saídas de cada estado da máquina e para qual estado ela irá dependendo do estado atual e da entrada. Tudo feito de forma que a calculadora atue da forma em que foi solicitado.

3. Análise dos Resultados

Neste projeto, tivemos que criar uma calculadora de 8 bits em complemento de 2 com operações de soma, subtração, *store* e *load*. O usuário digita o valor ou operação no teclado matricial 4x4 e assim, o circuito da calculadora reconhecerá a tecla digitada e, assim realizará a operação desejada.

Link do vídeo do projeto: https://youtu.be/4ABfbB8m7zg

Boa parte do projeto foi aproveitado da etapa do pré-projeto, como dito acima (secção 2.1), portanto, todas as funcionalidades do pré-projeto também funcionam no projeto final, como a mensagem de erro nos displays, por exemplo, entre outros.

Link do vídeo do pré-projeto: https://youtu.be/3qOHkVqhqIg

4. Conclusão

O projeto, no geral, foi bem complexo, apresentando dificuldades, principalmente na parte final, mais especificamente, a máquina de estados, que ao longo de seu desenvolvimento, foi sendo necessário cria cada vez mais entradas, saídas e estados, e assim, ocasi-

onalmente, resultando em *bugs*, que apresentavam dificuldades diversas para solucionar. Porém, no final, obtivemos o resultado procurado.

Referências

[1] FLOYD, *Thomas L. Sistemas Digitais: fundamentos e aplicações: 9.ed - Porto Alegre*. Bookman, 2007.