Projeto Final
Mini-Calculadora

Cristiane Damacena Gonçalves de Oliveira, 18/0047906
João Pedro Gomes Covaleski Marin Antonow, 22/1006351
Willyan Marques de Melo, 22/1020940
Grupo A10

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0231 - Laboratório de Circuitos Lógicos

damacena.cris@gmail.com, jpantonow@gmail.com, willyanmarquesmelo@gmail.com

Abstract. *Through the implementation of an Arithmetic Logic Unit, digitally developed in verilog and supported by the precepts of a Mealy Finite State Machine, the software simulation was obtained and the concrete realization on the FPGA DE2 board of algebraic operations - addition, subtraction and multiplication - between two integers of one decimal digit, defined by the user through an external keyboard.*

Resumo. *Por meio da implementação de uma Unidade Lógico Aritmética, desenvolvida digitalmente em verilog e amparada sobre os preceitos de uma Máquina de Estados Finitos de Mealy, obteve-se a simulação em software e e a concreta realização na placa FPGA DE2 de uma calculadora simples que realiza operações algébricas - soma, subtração e multiplicação - entre dois números inteiros de um dígito decimal, definidos pelo usuário por meio de um teclado externo.*

1. Introdução

1.1. ULA - Unidade Lógico Aritmética

A Unidade Lógica Aritmética - ULA - consiste em um dispositivo capaz de efetuar, por meio de um conjunto de números expressos em um circuito digital, operações selecionadas através de uma entrada de controle, cujas naturezas variam do espectro aritmético - soma, subtração, multiplicação, etc. - ao campo lógico - OR, NOT, AND etc -, tendo sua saída, pois, equivalente ao resultado da operação selecionada. Para que implementemos uma ULA, devemos nos atentar aos conhecimentos previamente obtidos acerca de uma máquina de estados, pois os estágios da operação e o seu respectivo resultado dependem de uma série de validadores e circunstâncias que devem ser bem estabelecidas para o funcionamento do projeto.

1.2. Máquina de Estados

Para que seja possível a construção de um circuito que siga uma ordem de estados específicos conforme algum parâmetro de entrada, deve-se implementar uma máquina de estados, que consiste em um circuito sequencial que representa transições finitas entre os estágios presentes no funcionamento do projeto. Na criação de uma máquina de estados, é necessário seguir uma série de passos a partir de um problema conhecido, que são:

1. Obter o diagrama de estados
2. Codificar os estados
3. Escolher os Flip-Flops que serão utilizados para a implementação do circuito
4. Designar as equações para os Flip-Flops e para a saída
5. Desenhar o circuito

Em uma calculadora simples, como a requerida pelo projeto final, por exemplo, tem-se a seguinte circunstância: requer-se o resultado de uma operação escolhida pelo usuário - adição, subtração e multiplicação - entre dois números naturais positivos, inseridos no intervalo de 0 a 9.

Para implementarmos o projeto, foi necessária a adoção de uma máquina de estados, em que cada estado representará uma etapa da operação desejada pelo usuário. Simplificando a conceituação do projeto, temos que o estado inicial será o estado de espera da definição do primeiro dígito que, se for um dígito válido, fará a transição de estados para o operador. Este, se for válido, transicionará para o estado de espera do segundo número, que, caso seja um número nos limites impostos, promoverá a transição para o último estado, que é o estado em que o usuário apertará a tecla correspondente ao $=$, decorrendo em uma saída que representa o resultado da operação desejada entre os dois números escolhidos e, por fim, promovendo a transição, novamente, ao estado inicial.

Para isso, devemos adotar em nossos procedimentos os 5 passos previamente assertados de implementação de uma máquina de estados, bem como o empregar o uso de alternativas que auxiliem ou facilitem nosso processo de criação do trabalho.

1.3. Tipo de Máquina

Posto que temos entrada de dados que, por serem válidas ou inválidas, mudam, nas transições, a variável correspondente à saída, podemos assertar que um tipo de máquina fácil para a implementação do circuito é a Máquina de Mealy, que se baseia na inclusão de uma entrada e uma saída específicas para cada transição correspondente à mudanças de estado - ou permanência de estado - da máquina.

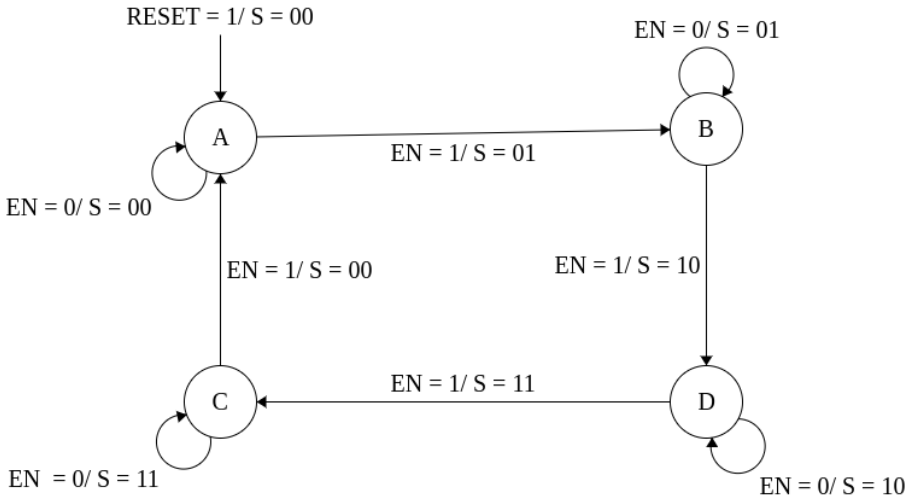

Figura 1. Exemplo de representação de uma Máquina de Mealy

1.4. Teclado Matricial de Membrana

O teclado matricial de membrana é um aparato designado para tornar a entrada de dados do circuito mais intuitiva, dado o seu design que composto por 16 teclas selecionáveis pelo toque. Ao conectarmos o teclado por meio de pinos presentes na FPGA DE2, podemos definir a entrada de dados ao pressionar as teclas do teclado, o que tornará o processo de cálculo mais espontâneo.

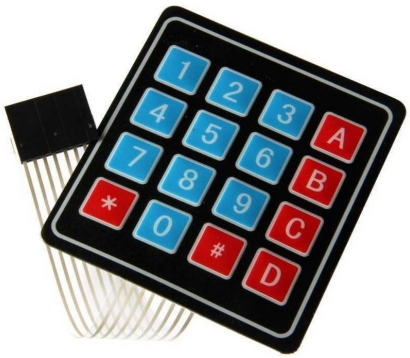

Figura 2. Foto do teclado matricial

1.5. Look Up Table

O método *Look Up Table* (LUT) consiste na construção de uma memória fixa de leitura conhecida como *Ready Only Memory* (ROM) a qual designa para cada combinação de entrada uma saída específica. Nesse sentido, uma ROM é, basicamente, uma tabela da verdade na qual suas entradas são utilizadas como forma de endereçar e localizar o dado que deve ser direcionada para saída. A vantagem de utilizar essa técnica é que não será preciso encontrar as equações —simplificadas ou não — das saídas da tabela descrita, entretanto, durante a construção da ROM, ele precisará atribuir o comportamento das saídas para todas entradas de interesse.

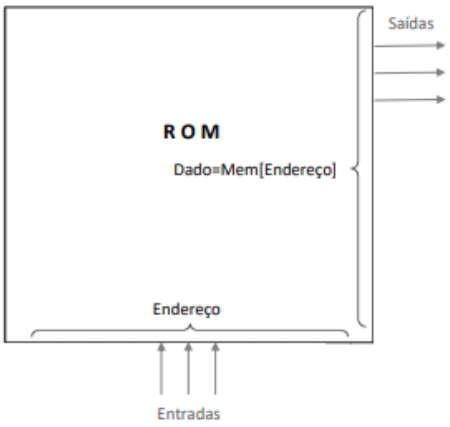

Figura 3. Representação da Memória ROM

1.6. Especificações do Projeto

Para desenvolvermos nossa calculadora, deve-se, precipuamente, estabelecer suas regras de funcionamento propostas pelo roteiro. O sistema deve, pelo LEDG[8] da FPGA, sinalizar que a máquina está em seu estado inicial, ou seja, que é possível receber valores e também que a última operação foi realizada com sucesso. Além disso, segue-se também as seguintes restrições para o seu funcionamento:

1. A entrada do primeiro valor decimal deve ser registrada, mostrada no display HEX6 e também nos LEDs vermelhos LEDR[17:14];

2. Em seguida, uma de três operações deve ser selecionada pelos botões A, B e C, que correspondem ao produto, subtração e soma, respectivamente, e mostrada nos LEDs LEDG[1:0] com as decodificações 11, 10 e 01;

3. De maneira análoga ao primeiro caso, a entrada do segundo valor deve ser registrada, mostrada no display HEX4 e também nos LEDs vermelhos LEDR[12:9], pulando-se o LED[13];

4. A tecla D simbolizará o sinal de igualdade (=) e precisa ser pressionada por último a fim de mostrar o resultado da conta para a operação entre os valores escolhidos. O resultado deve ser exposto nos displays HEX1 e HEX0 e seu sinal, caso seja menor que zero, deve ser mostrado no display HEX2;

5. Por fim, o botão KEY[0] da placa deve ser usado como RESET do circuito, que faz com que este volte para o estado inicial.

Desse modo, com o intuito de transmitir os resultados das operações efetuadas para a placa FPGA DE2, a fim de que os LEDs acendam e os valores das operações sejam mostradas no visor da placa, devem ser utilizados decodificadores para converter os resultados obtidos das operações algébricas, tornando possível a leitura das informações nos componentes da placa.

1.7. Objetivos

Deve-se implementar, por meio de uma ULA, o circuito de uma calculadora simples que efetua operações algébricas entre números inteiros - adição, subtração, multiplicação - , definidos pelo usuário ao pressionar teclas de um teclado matricial, e a conta e seu resultado expostos nos displays e LEDs da placa FPGA DE2.

1.8. Materiais

Neste experimento foram utilizados os seguintes materiais e equipamentos:

- Kit de Desenvolvimento DE2;
- 1 Mini-teclado matricial 4x4;
- Software Quartus-II v.13.0

2. Procedimentos

Antes de tudo, importamos do Moodle o arquivo do controlador do teclado externo, o compilamos e exportamos os módulos presentes no arquivo como blocos. Inserimos o bloco 'teclado' no top-level do projeto e realizamos as seguintes conexões conforme o roteiro:

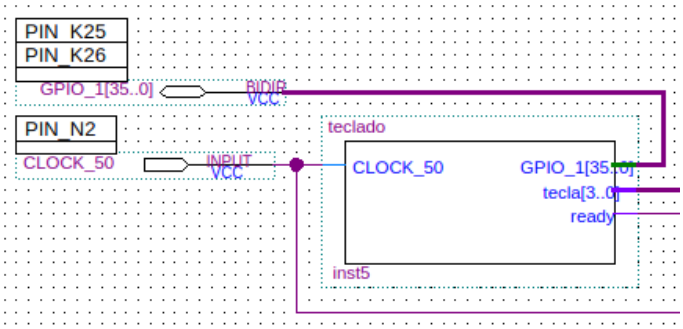

Figura 4.

De maneira simplificada, este circuito recebe como entrada os sinais enviados pelo teclado matricial, os quais são convertidos para um valor de saída binário de 4 bits (valores de 0 a 15, um para cada tecla) e também para a saída 'ready', que indica se uma tecla está ou não sendo pressionada.

2.1. Pré-Projeto

Para construção da nossa calculadora, implementaremos uma máquina de estado considerando-se as especificações do projeto. A máquina proposta será do tipo de Mealy e possuirá como entrada o valor selecionado pelo teclado e também o sinal 'ready' que indica se uma tecla está sendo pressiona, o qual funcionará como um validar dos dados de entrada. Ela terá um total de 4 estados usados para validação dos valores, de maneira a garantir que as teclas sejam sempre pressionadas dentro da sequência correta.

1. 'N1': obriga que um valor decimal seja escolhido (teclas 0 a 9);
2. 'OP': obriga que uma operação seja escolhida, isto é, que a segunda entrada seja uma das teclas A, B ou C (hexadecimais de 10, 11 e 12);
3. 'N2': assim como N1, obriga que um valor decimal seja escolhido (teclas 0 a 9);
4. 'RE': obriga que a entrada seja a tecla D (hexadecimal de 13).

O circuito possuirá duas variáveis de saída auxiliares 'E' (*Enable*) e 'S' (*Sucesso*), a primeira para ativar o circuito que realiza a operação e a segunda para indicar se a máquina pode receber o primeiro valor.

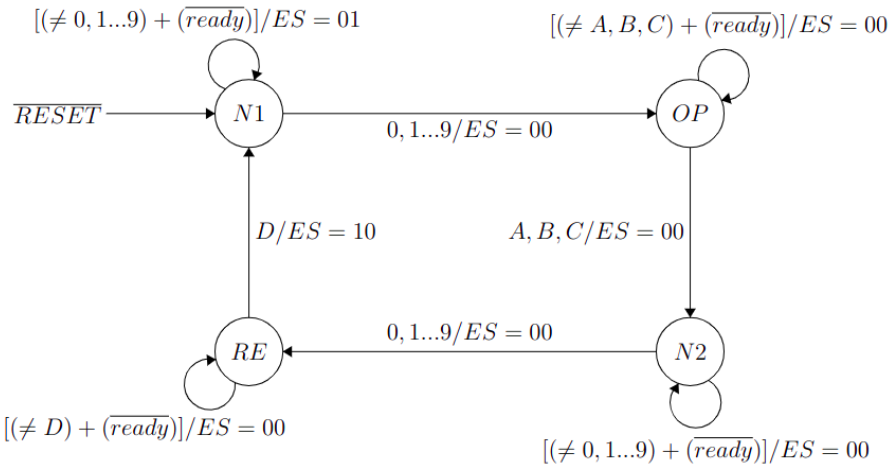

Figura 5.

A partir das informações expostas no diagrama, construímos a seguinte tabela de transição:

Atual	Entradas		Próximo	Saídas	
Q_1Q_0	ready	valor	$Q_1'Q_0'$	E	S
N1	0	x	N1	0	1
N1	1	$\neq 0,1...9$	N1	0	0
N1	1	$0,1...9$	OP	0	0
OP	0	x	OP	0	0
OP	1	$\neq A,B,C$	OP	0	0
OP	1	A,B,C	N2	0	0
N2	0	x	N2	0	0
N2	1	$\neq 0,1...9$	N2	0	0
N2	1	$0,1...9$	RE	0	0
RE	0	x	RE	0	0
RE	1	$\neq D$	RE	0	0
RE	1	D	N1	1	0

Tabela 1. Tabela de estados do projeto simplificada.

Dado que haviam apenas 4 estados diferentes, optamos por usar apenas 2 flip-flops do tipo D, por sua implementação mais simples, e também usar a codificação binária para as etapas.

Estado	Codificação
N1	00
OP	01
N2	10
RE	11

Tabela 2. Codificação dos estados.

Reescrevendo a tabela de transição temos:

Atual	Entradas		Próximo	Saídas		Equação
Q_1Q_0	ready	valor	$Q_1'Q_0'$	E	S	D_1D_0
00	0	x	00	0	1	00
00	1	$\neq 0, 1...9$	00	0	0	00
00	1	$0, 1...9$	01	0	0	01
01	0	x	01	0	0	01
01	1	$\neq A, B, C$	01	0	0	01
01	1	A, B, C	10	0	0	10
10	0	x	10	0	0	10
10	1	$\neq 0, 1...9$	10	0	0	10
10	1	$0, 1...9$	11	0	0	11
11	0	x	11	0	0	11
11	1	$\neq D$	11	0	0	11
11	1	D	00	1	0	00

Tabela 3. Tabela de estados reescrita.

Escolhemos fazer com que S assumisse valor alto quando ready fosse 0 e a máquina estivesse no estado N1 pois, assim que uma operação seja realizada e o resultado exposto, a máquina entrará automaticamente no estado N1 com o ready em 0 e, dessa maneira, exibirá imediatamente que um novo valor pode ser selecionado. Por fim, com a tabela podemos iniciar a construção da nossa máquina de estados.

2.2. Máquina de Estados

Primeiramente criamos um novo arquivo de digrama e adicionamos dois flip-flops tipo D disponibilizados pelo Quartus. Criamos uma entrada para reset conectada às entradas CLRN de ambos circuitos, visto que o estado inicial N1 é codificado como '00', e ligamos as entradas PRN ao VCC. Ao invés de construirmos as tabelas completas de cada estado para que pudéssemos realizar o Karnaugh e obter as equações de entrada dos flip-flops, optamos por utilizar uma ROM para definir os valores dos bits D_1D_0.

Primeiro, criamos um arquivo verilog do módulo 'ROM' com as seguintes entradas: 'crnt' (abreviação de *current*, estado atual), 'tecla' (valor de saída do teclado), 'ready' e 'reset'. Já para as saídas, foram declarados os outputs: 'next' (próximo estado), 'N1', 'OP', 'N2', 'E' e 'S'. Criamos em seguida um fio nomeado 'In' —abreviação de Input— para agrupar as variáveis de entrada 'ready' e 'tecla'. Declaramos o estado inicial do módulo no bloco *initial*, onde todas saídas são levada para 0 para que não interfiram no funcionamento do circuito.

Declaramos um bloco *always* para que o código descrito fosse sintetizado sequencialmente e criamos uma condicional com prioridade que checa se o valor de reset é 0, visto que o botão KEY[0] emite constantemente um sinal alto que transiciona para o nível baixo quando pressionado, para saber se as saídas da ROM devem ser zeradas. Caso o circuito não esteja com reset ativo, lê-se o estado atual a fim de definir o próximo estado ('next') e/ou armazenar os valores do teclado.

OBS: no código de descrição da ROM, as variáveis de saída possuem a mesma nomeação que os estados, mas ambos não possuem correlação.

```
 1  module ROM(
 2  input  logic [1:0] crnt,
 3  input  logic [3:0] tecla,
 4  input  logic       ready,
 5  input  logic       reset,
 6  output logic [1:0] next,
 7  output logic [3:0] N1,
 8  output logic [3:0] OP,
 9  output logic [3:0] N2,
10  output logic       E,
11  output logic       S);
12
13  wire [5:0] In;
14  assign In={ready,tecla};
15
16  initial
17     begin
18        N1   <= 0;
19        OP   <= 0;
20        N2   <= 0;
21        E    <= 0;
22        next <= 0;
23     end
24
25  always @(*)
26     begin
27        if (~reset)
28           begin
29              N1   <= 0;
30              OP   <= 0;
31              N2   <= 0;
32              E    <= 0;
33              next <= 0;
34           end
35
36        else
37           begin
38              case (crnt)
```

Figura 6. Código da ROM em verilog - 1.

Assim sendo, baseando-se na tabela de transição, caso o estado atual seja ’00’: se ’ready’ for 0, a saída E recebe 0 e a S recebe 1 e o estado não muda. Caso contrário (ready = 1), se ’tecla’ for um valor de 0 a 9, o valor é levado para a saída N1 e o próximo estado será ’01’, senão, o estado não muda. Além disso, é possível notar que quando ocorre a transição para o estado ’01’ a saída N2 recebe 0, isso ocorre para que depois que uma operação já tiver sido realizada, assim que um novo decimal válido for selecionado, o display e os LEDs que mostram segundo valor sejam limpos.

```
40              2'b 00: begin //N1
41                 casex (In)
42                    5'b 0_xxxx : begin next = 2'b 00; E<=0; S<=1; end
43
44                    5'b 1_0000 : begin next = 2'b 01; N1<= 4'd 0; N2 <= 0;end //0
45                    5'b 1_0001 : begin next = 2'b 01; N1<= 4'd 1; N2 <= 0;end //1
46                    5'b 1_0010 : begin next = 2'b 01; N1<= 4'd 2; N2 <= 0;end //2
47                    5'b 1_0011 : begin next = 2'b 01; N1<= 4'd 3; N2 <= 0;end //3
48                    5'b 1_0100 : begin next = 2'b 01; N1<= 4'd 4; N2 <= 0;end //4
49                    5'b 1_0101 : begin next = 2'b 01; N1<= 4'd 5; N2 <= 0;end //5
50                    5'b 1_0110 : begin next = 2'b 01; N1<= 4'd 6; N2 <= 0;end //6
51                    5'b 1_0111 : begin next = 2'b 01; N1<= 4'd 7; N2 <= 0;end //7
52                    5'b 1_1000 : begin next = 2'b 01; N1<= 4'd 8; N2 <= 0;end //8
53                    5'b 1_1001 : begin next = 2'b 01; N1<= 4'd 9; N2 <= 0;end //9
54
55                    5'b 1_1010 : begin next = 2'b 00; end
56                    5'b 1_1011 : begin next = 2'b 00; end
57                    5'b 1_1100 : begin next = 2'b 00; end
58                    5'b 1_1101 : begin next = 2'b 00; end
59                    5'b 1_1110 : begin next = 2'b 00; end
60                    5'b 1_1111 : begin next = 2'b 00; end
61                 endcase
62              end
```

Figura 7. Código da ROM em verilog - 2.

Caso o estado atual seja ’01’: se ready for 0 o estado não muda, caso contrário, se ’tecla’ for um dos valores 10,11 ou 12 (A, B e C em hexadecimal), a saída OP recebe esse valor e o próximo estado será ’10’, caso contrário nada acontece.

```
64          2'b 01: begin //OP
65             casex (In)
66                5'b 0_xxxx : begin next = 2'b 01; S<=0;end
67
68                5'b 1_0000 : begin next = 2'b 01; end
69                5'b 1_0001 : begin next = 2'b 01; end
70                5'b 1_0010 : begin next = 2'b 01; end
71                5'b 1_0011 : begin next = 2'b 01; end
72                5'b 1_0100 : begin next = 2'b 01; end
73                5'b 1_0101 : begin next = 2'b 01; end
74                5'b 1_0110 : begin next = 2'b 01; end
75                5'b 1_0111 : begin next = 2'b 01; end
76                5'b 1_1000 : begin next = 2'b 01; end
77                5'b 1_1001 : begin next = 2'b 01; end
78
79                5'b 1_1010 : begin next = 2'b 10; OP <= 4'd 10; end // A
80                5'b 1_1011 : begin next = 2'b 10; OP <= 4'd 11; end // B
81                5'b 1_1100 : begin next = 2'b 10; OP <= 4'd 12; end // C
82
83                5'b 1_1101 : begin next = 2'b 01; end
84                5'b 1_1110 : begin next = 2'b 01; end
85                5'b 1_1111 : begin next = 2'b 01; end
86             endcase
87          end
```

Figura 8. Código da ROM em verilog - 3.

Caso o estado atual seja ’10’: se ready = 0, nada acontece, senão, se ’tecla’ for um valor de 0 a 9, o valor é levado para N2 e o próximo estado será ’11’, caso contrário, o estado não muda.

```
89          2'b 10: begin //N2
90             casex (In)
91                5'b 0_xxxx : begin next = 2'b 10; end
92
93                5'b 1_0000 : begin next = 2'b 11; N2 <= 4'd 0; end //0
94                5'b 1_0001 : begin next = 2'b 11; N2 <= 4'd 1; end //1
95                5'b 1_0010 : begin next = 2'b 11; N2 <= 4'd 2; end //2
96                5'b 1_0011 : begin next = 2'b 11; N2 <= 4'd 3; end //3
97                5'b 1_0100 : begin next = 2'b 11; N2 <= 4'd 4; end //4
98                5'b 1_0101 : begin next = 2'b 11; N2 <= 4'd 5; end //5
99                5'b 1_0110 : begin next = 2'b 11; N2 <= 4'd 6; end //6
100               5'b 1_0111 : begin next = 2'b 11; N2 <= 4'd 7; end //7
101               5'b 1_1000 : begin next = 2'b 11; N2 <= 4'd 8; end //8
102               5'b 1_1001 : begin next = 2'b 11; N2 <= 4'd 9; end //9
103
104               5'b 1_1010 : begin next = 2'b 10; end
105               5'b 1_1011 : begin next = 2'b 10; end
106               5'b 1_1100 : begin next = 2'b 10; end
107               5'b 1_1101 : begin next = 2'b 10; end
108               5'b 1_1110 : begin next = 2'b 10; end
109               5'b 1_1111 : begin next = 2'b 10; end
110            endcase
111         end
```

Figura 9. Código da ROM em verilog - 4.

Caso a máquina chegue no último estado ’11’, ela permanecerá nele até que a tecla ’D’ seja pressionada, o que faz com que a saída ’E’ receba 1 e o circuito volte para o estado inicial.

```
113                 2'b 11: begin //RE
114                    casex (In)
115                       5'b 0_xxxx : begin next = 2'b 11;  end
116
117                       5'b 1_0000 : begin next = 2'b 11;  end
118                       5'b 1_0001 : begin next = 2'b 11;  end
119                       5'b 1_0010 : begin next = 2'b 11;  end
120                       5'b 1_0011 : begin next = 2'b 11;  end
121                       5'b 1_0100 : begin next = 2'b 11;  end
122                       5'b 1_0101 : begin next = 2'b 11;  end
123                       5'b 1_0110 : begin next = 2'b 11;  end
124                       5'b 1_0111 : begin next = 2'b 11;  end
125                       5'b 1_1000 : begin next = 2'b 11;  end
126                       5'b 1_1001 : begin next = 2'b 11;  end
127                       5'b 1_1010 : begin next = 2'b 11;  end
128                       5'b 1_1011 : begin next = 2'b 11;  end
129                       5'b 1_1100 : begin next = 2'b 11;  end
130
131                       5'b 1_1101 : begin next = 2'b 00; E<= 1; end //D
132
133                       5'b 1_1110 : begin next = 2'b 11; end
134                       5'b 1_1111 : begin next = 2'b 11; end
135                    endcase
136                 end
137
138              endcase
139           end
140     end
141
142  endmodule
```

Figura 10. Código da ROM em verilog - 5.

Com a ROM construída, exportamos o código como um diagrama em bloco para que pudesse ser usado juntamente com os flip-flops. Ligamos as saídas Q_1Q_0 à entrada 'crnt' da ROM e a saída 'next' às entradas D de cada flip-flop. Adicionamos os pinos de output para as saídas de valores da ROM e ao node 'Q' para indicar o estado atual no top-level e, com isso, exportamos o arquivo como bloco nomeando-o 'maquina' para ser usado no nível principal.

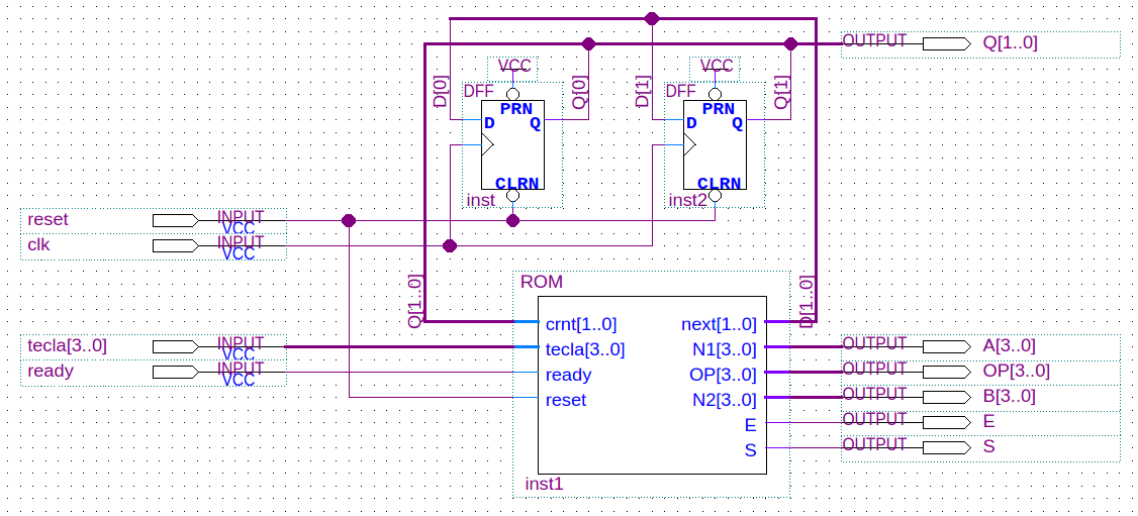

Figura 11. Diagrama em blocos da máquina de estados.

2.3. Unidade Lógica Aritmética Simples

Criamos um novo arquivo verilog para descrição da nossa ULA para o teclado, a qual funcionará em conjunto com a máquina de estados e receberá as saídas desta. O módulo declarado possui um input para reset e também as entradas 'Q', 'A','OP','B' e 'E', para, respectivamente, o estado atual, o primeiro valor, a operação, o segundo valor e o enable do circuito. Suas saídas são 'result' e 'sinal' que correspondem ao resultado da operação e o sinal deste. Vale observar que as estradas para os valores numéricos (A e B) e para a operação escolhida (OP) são guardados em registradores.

A ULA possui duas condições para reset: se a entrada reset for 0 ou se o estado atual da máquina for 'OP. Essa condição extra garante que, caso uma conta já tenha sido realizada, quando um novo valor de 0 a 9 seja escolhido e a máquina entrar no estado '01' (OP) o valor do resultado seja apagado.

Essa ULA funcionará somente quando o input E, ativado apenas na transição do estado RE para N1, for 1, garantindo que sempre dois valores e uma operação já tenham sido selecionados. Nesse caso, o módulo verificará qual operação foi escolhida e fará com que a saída 'result' receba o resultado adequado. Além do mais, como as contas são sempre realizadas baseando-se na regra de complemento de 2, a saída para o 'sinal' do resultado é simplesmente o bit mais significativo deste.

```
1  module operador(
2  input  logic       reset,
3  input  logic [1:0] Q,
4  input  reg   [3:0] A,
5  input  reg   [3:0] OP,
6  input  reg   [3:0] B,
7  input  logic       E,
8  output logic [7:0] result,
9  output logic       sinal);
10
11 initial
12    begin
13       result <= 7'd0;
14    end
15
16 always@(*)
17    begin
18       if (~reset | (~Q[1]&Q[0]))
19          result = 7'd0;
20
21       else
22          begin
23             if (E)
24                begin
25                   case (OP)
26
27                      4'b1010:
28                         result <= A*B;
29
30                      4'b1011:
31                         result <= A-B;
32
33                      4'b1100:
34                         result <= A+B;
35
36                   endcase
37             end
38          end
39    end
40
41 assign sinal = result[7];
42
43 endmodule
```

Figura 12. Código em verilog da ULA.

2.4. Representação de entrada e saída

A última etapa do projeto consiste apenas em mostrar os valores e operadores guardados pelo circuito em seus respectivos LEDs e displays conforme as especificações dadas. Para isso importamos o arquivo do decodificador para display de 7 segmentos (’decoder7’) disponibilizado no Moodle.

```
1  module decoder7 (
2  input  logic [3:0] In,
3  output logic [6:0] Out);
4
5   initial
6      begin
7         Out = ~7'b0111111;
8      end
9
10  always @(*)
11     case (In)
12
13        4'b0000 : Out = ~7'b0111111;
14        4'b0001 : Out = ~7'b0000110;
15        4'b0010 : Out = ~7'b1011011;
16        4'b0011 : Out = ~7'b1001111;
17
18        4'b0100 : Out = ~7'b1100110;
19        4'b0101 : Out = ~7'b1101101;
20        4'b0110 : Out = ~7'b1111101;
21        4'b0111 : Out = ~7'b0000111;
22
23        4'b1000 : Out = ~7'b1111111;
24        4'b1001 : Out = ~7'b1101111;
25        4'b1010 : Out = ~7'b1110111;
26        4'b1011 : Out = ~7'b1111100;
27
28        4'b1100 : Out = ~7'b0111001;
29        4'b1101 : Out = ~7'b1011110;
30        4'b1110 : Out = ~7'b1111001;
31        4'b1111 : Out = ~7'b1110001;
32
33        default : Out = ~7'b0000000;
34     endcase
35
36  endmodule
```

Figura 13. Código em verilog do decodificador para display hexadecimal.

2.4.1. Decodificação do Cálculo

A decodificação dos valores de entrada será feita apenas após estes terem sidos validados pela máquina de estados, de forma que sempre serão exibidos na mesma sequência já explicada anteriormente (valor A $\rightarrow$ operador $\rightarrow$ valor B $\rightarrow$ resultado). Com isso, usamos dois ’decoder7’ para as saídas A e B da máquina, as quais foram conectadas aos displays HEX6 e HEX4, também aos LEDs LEDR[17:14] e LEDR[12:9].

Em seguida, criamos um decodificador simples para os valores correspondentes aos operadores (’*’,’-’,’+’). Para tal, declaramos um novo módulo nomeado ’decoder_OP’ com a entrada OP, para a operação escolhida, e a saída LED, para a decodificação do operador. Se a tecla ’A’ (tecla = 1010) foi escolhido para operação, a saída gerada será os dois bits ’11’ indicando multiplicação; Se a tecla B foi pressionada, a saída será ’10’ (tecla = 1011), indicando subtração; Se a tecla pressionada foi ’C’ (tecla = 1100), a saída será ’01’, indicando soma. Se nenhuma das condições for verdadeira, a saída será sempre ’00’.

```
1  module decoder_OP(
2  input  logic [3:0] OP,
3  output logic [1:0] LED);
4
5  always@(*)
6     begin
7        case (OP)
8
9           4'b1010: LED <= 2'b 11;
10
11          4'b1011: LED <= 2'b 10;
12
13          4'b1100: LED <= 2'b 01;
14
15          default : LED <= 2'b00;
16
17       endcase
18    end
19
20 endmodule
```

Figura 14. Código em verilog de decodificação da operação escolhida.

Exportamos o arquivo como bloco e o adicionamos ao top-level, ligando a saída 'OP' da máquina à sua entrada homônima do decodificador e sua saída 'S' (*Sucesso*) ao LEDG[8]. Finalmente, conectamos a saída 'LED' do nosso decodificador aos LEDs LEDG[1:0].

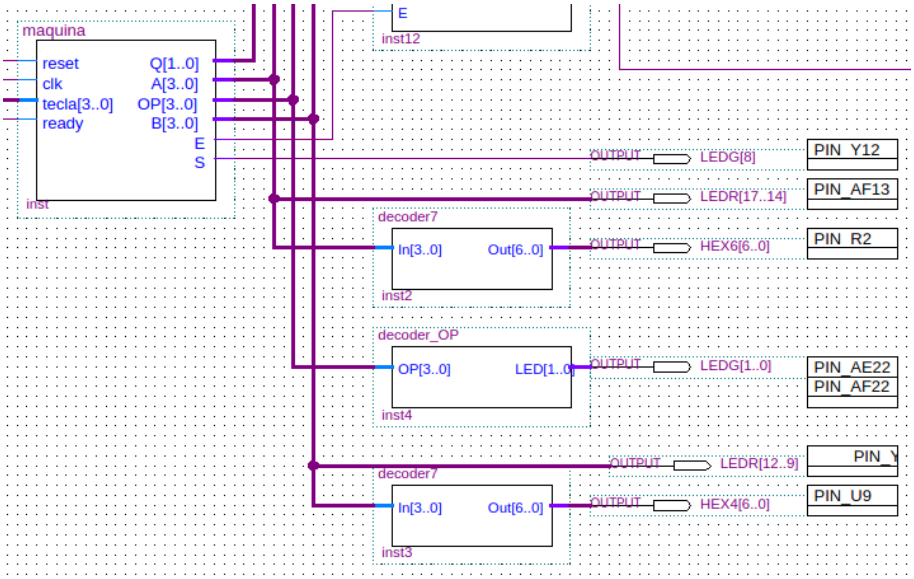

Figura 15. Diagrama em blocos para os circuitos de representação das entradas.

2.4.2. Decodificação do Resultado

Primeiro, como era mais simples, criamos em verilog um módulo 'decoder_sinal' que recebe o bit mais significante do resultado como entrada que, como já explanado, indica o sinal do valor em binário, e o decodifica para um display hexadecimal. Assim, caso o sinal seja 0, isto é, o resultado seja positivo, a decodificação gerará o símbolo do dígito 0. Senão, se o sinal for 1 (resultado negativo), a decodificação será oposta à do digito 0, ou seja, apenas o segmento central acenderá.

```
module decoder_sinal(
input  logic       sinal,
output logic [6:0] HEX);

 always @(*)
    begin
       case (sinal)

          0 : HEX = ~7'b0111111;
          1 : HEX = ~7'b1000000;

       endcase
    end
endmodule
```

Figura 16. Código em verilog de decodificação do sinal do resultado.

Agora para decodificação do resultado em si, a fim de reaproveitar o decoder para os displays, tendo em mente que as operações realizadas poderiam resultar em números no intervalo -9 (0-9) à 81 (9*9), tivemos de criar um circuito separador de valores para o dígito de dezena e de unidade. O módulo criado tem como entrada 'valor' (o valor do resultado) e como saída 'decimal' e 'unidade', que, como o nome já diz, correspondem aos dígitos de decimal e unidade da entrada.

Declaramos então um fio para o sinal do resultado (positivo ou negativo) e um fio 'absoluto' para receber o valor absoluto da entrada. Tendo-se em mente que o resultado segue o padrão de representação de complemento para 2, caso o MSBit for 0, o valor em módulo será igual ele mesmo, senão, seu módulo será obtido pela soma da negação do valor com 1. Para obtermos as unidades do valor de entrada utilizamos o operador de módulo '%' em verilog para obter o resto da divisão do valor absoluto da entrada por dez. Para conseguirmos o valor das dezenas, subtraímos a unidade do valor total do módulo da entrada e o dividimos por 10.

```
module separador(
input  logic [7:0] valor,
output logic [3:0] decimal,
output logic [3:0] unidade);

wire       sinal;
wire [7:0] absoluto;

assign sinal = valor[7];
always @(*)
   begin
      case (sinal)

         0 : absoluto = valor;
         1 : absoluto = (~valor)+1;

      endcase
   end

assign unidade = absoluto%10;
assign decimal = (absoluto-unidade)/10;

endmodule
```

Figura 17. Código em verilog do separado de dígitos decimais.

Exportamos os arquivos do decodificador do sinal e do separador de decimais como blocos e os adicionamos ao arquivo principal do projeto, ligando-se as saídas 'result' e 'sinal' da ULA a ambos, respectivamente. A saída 'HEX' do bloco decodificador foi ligada ao display HEX2 e as saídas 'decimal' e 'unidade' foram conectas a dois decodificadores para os displays HEX1 e HEX0.

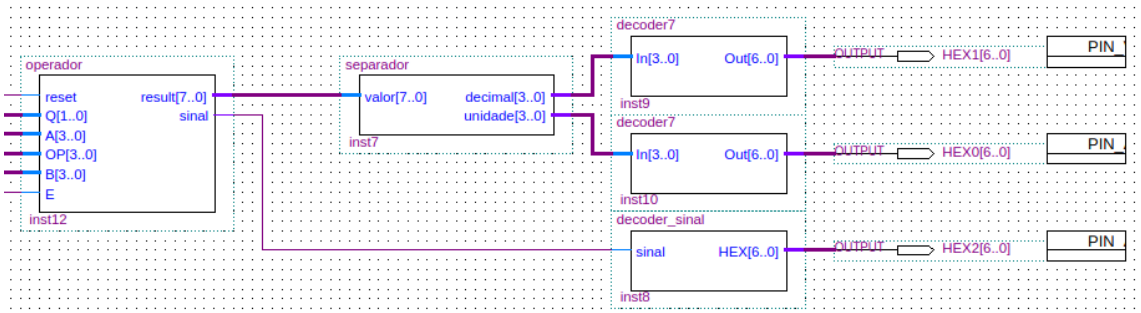

Figura 18. Diagrama em blocos para os circuitos de representação do resultado.

A fim de garantirmos o funcionamento correto dos LEDs, adicionamos um pino GRD (ground) ao top-level e conectamos à ele todos os LEDs, tanto vermelhos quanto verdes, inutilizados. Também inserimos outro decodificador de 7 segmentos com entrada aterrada e saída conetada aos displays que não apresentarão valores ou sinais.

3. Análise de Resultados

Após a descrição e elaboração de cada parte do projeto, o diagrama final do circuito da nossa mini-calculadora foi o seguinte:

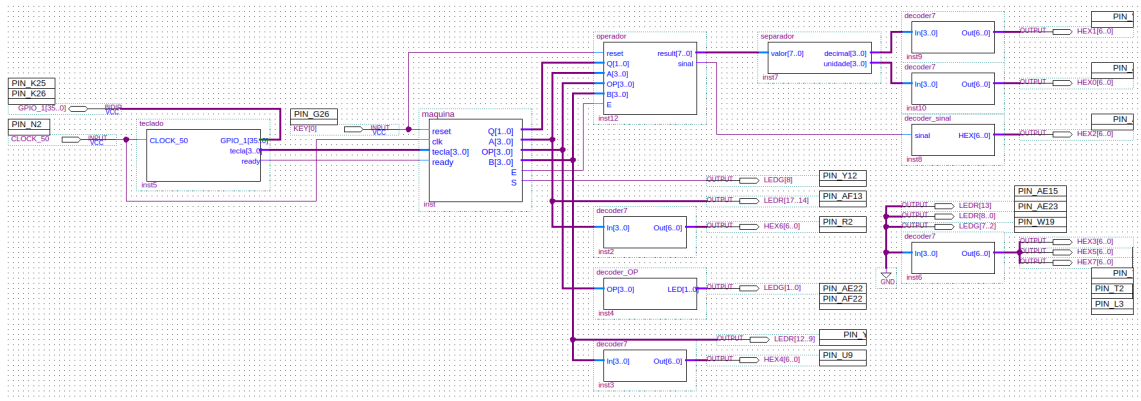

Figura 19. Diagrama em blocos da mini-calculadora — top-level.

Logo, implementamos na FPGA e obtivemos o resultado, disponível em *vídeo* (https://youtu.be/-uKG3bErS1E). Assim, devemos nos atentar ao comportamento do sistema para cada estágio da operação.

Primeiramente, ao pressionarmos KEY[0] do teclado, tem-se que o circuito vai para o estado inicial,em que o LEDG[8] acende e o sistema, como previsto, está preparado para a inserção do primeiro numerador. Ao inserirmos o primeiro número no teclado matricial, observamos um padrão em que os LEDR[17:14] acendem em uma dada sequência, representando o numerador em binário e conseguimos ver sua forma decimal no visor da placa HEX6. Ao pressionarmos a tecla da operação, vemos um comportamento em que os LEDG[1:0] acendem, e, em seguida, ao definirmos um segundo numerador, percebe-se um padrão em que os LEDR[12:9] acendem, que é codificação binária deste, e vemos sua codificação decimal no visor HEX4. Por fim, ao pressionarmos a tecla D do teclado, surge

o resultado da operação algébrica definida previamente nos visores HEX1 e HEX0, com destaque à operação de subtração em que o resultado é negativo quando o segundo valor for maior que o primeiro, sendo possível visualizar o símbolo -"acompanhando o resultado. Listado todos os pontos, percebemos que os padrões condizem com o experimento proposto.

4. Conclusão

Dado a análise de resultados, devemos rever e ressaltar se os pressupostos teóricos requeridos pelo projeto, conforme especificado na introdução, nos objetivos e na especificação do sistema digital, foram compatíveis com os resultados experimentais obtidos. Para isso, podemos aplicar o método da verificabilidade para analisar se houve a validez do experimento, devemos analisar passo a passo se os requerimentos do sistema digital compactuam com os resultados empíricos obtidos.

Precipuamente, temos, nas especificações do sistema, a exigência de que apenas inserções válidas sejam feitas nos numeradores, nos operadores e no resultado da operação. Portanto, caso haja a inserção de algum número inválido, deve-se haver a permanência no estado em questão. Como visto experimentalmente, confere-se a validez nesse tópico. Além disso, podemos assertar que definimos uma máquina de Mealy em verilog para o projeto, dado que as saídas são condicionadas pela ativação ou não dos validadores do circuito, o que condiz com o que foi estabelecido inicialmente como uma alternativa à resolução do problema.

Em relação ao comportamento dos componentes da placa, temos o seguinte: caso o botão KEY[0] da placa seja acionado, haverá o RESET do circuito, o que fará com que a máquina de estados retorne ao estado inicial, fato que, como visto em vídeo, confere validez funcional. Ademais, temos que a inserção de numeradores deve ser representada no display de 7 segmentos, bem como o resultado da operação, o que se pôde constatar em vídeo. Em relação aos mecanismos da placa ao receber operações, temos uma série de comportamento a serem seguidas, que foram todas executadas com sucesso no projeto em questão, mas que devem ser reiteradas para que a verificação seja completa.

Em uma operação de multiplicação, selecionável pela tecla A, os LEDs LEDG[1:0] devem estar ambos acesos, dado a decodificação 11 para o operador em questão e, como visto na experiência, houve a conformidade com essa exigência. Ademais, para um operador de subtração, que pode ser selecionado pela tecla, o LEDG[1], mais significativo, deve estar aceso, enquanto o LEDG[0], menos significativo, deve estar apagado, conforme a decodificação 10. Além disso, para uma operação de subtração em que o primeiro operador é menor que o segundo operador, deve aparecer um símbolo $-$ no display HEX2 representando que o resultado é um número negativo. Como esses resultados foram obtidos na FPGA, podemos, novamente, verificar que o procedimento conferiu sucesso. Por fim, em uma adição realizada pela ULA, com a decodificação 01 (LEDG[1] apagado e LEDG[0] aceso), verifica-se que o devido resultado é representado no display de 7 segmentos, validando mais uma etapa do projeto em questão.

Por conseguinte, haja vista que houve a concreta e efetiva verificação de todas as congruências entre os resultados empíricos e os requerimentos do projeto, pode-se constatar, assim, o sucesso do experimento, que cumpriu com suas metas em questão.

Referências

[Burch 2005] Burch, C. (2005). Logisim: a graphical tool for designing and simulating logic circuits. `http://www.cburch.com/logisim/`.

[Floyd 2007] Floyd, T. (2007). *Sistemas Digitais: Fundamentos e Aplicações*. Bookman, 9th edition.

[Harris 2017] Harris, D. M. e. H. S. L. (2017). *Digital design and computer architecture*. Morgan Kaufmann, 1st edition.

[Intel 2013] Intel (2013). Intel quartus ii web edition design software, version 13.0sp1.

[Pedroni 2010] Pedroni, V. (2010). *Eletrônica Digital Moderna e VHDL*. LTC, 5th edition.

[Tocci 2010] Tocci, R. J. e Widmer, N. S. (2010). *Sistemas digitais: princípios e aplicações*. LTC, 1st edition.