Gerson Daniel Santos Marques

José Raimundo Alves dos Santos Júnior

Israel Rocha Santana

Artur Pompílio Coité Araújo

Salvador - BA - Brasil

2021

SUMÁRIO

Introdução

A HDL(Hardware description language) é uma linguagem de modelagem e design de circuitos eletrônicos. Devido à sua flexibilidade e alternativas de alto nível, sua implementação reduz exponencialmente o tempo de desenvolvimento de um projeto, além de reduzir erros em uma área tão criteriosa como a de desenvolvimento eletrônico, e portanto é essencial para projetos do mercado. Dentre as HDL 's mais usadas, será usada para este projeto aquela que apresenta maior proximidade com linguagens de programação usuais: verilog.
O verilog é uma linguagem de descrição de hardware orientada a módulos. Sua simplicidade é perceptível ao comparar a eficiência de seu código, a qual realiza as mesmas operações que o VHDL com menos linhas e formalidades.

VHDL

```
library IEEE;
use IEEE.STD_Logic_1164.all;
entity HALF_ADDER is
  port(a,b:in std_logic;
       sum,carry:out std_logic);
end entity HALF_ADDER;


architecture STRUCT of HALF_ADDER is

component xor2
port(a,b:in std_logic; c:out std_logic);
end component;

component and2
port(a,b:in std_logic; c:out std_logic);
end component;

begin
X1: xor2 port map(a=>a,b=>b,c=>sum);
A1: and2 port map(a=>a,b=>b,c=>carry);
end STRUCT;
```

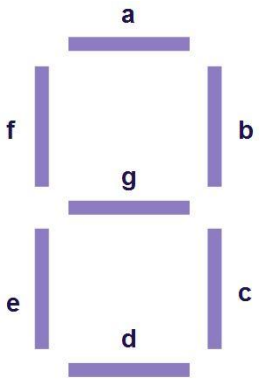
Verilog

```
module HALF_ADDER(a,b,sum,carry);
input a,b;
output sum,carry;

xor X1(sum,a,b);
and A1(carry,a,b);

endmodule
```

Para a escrita do código, existem três estilos possíveis de implementação: estrutural, dataflow e comportamental. O estilo estrutural se limita a descrição do circuito a ser efetuado, semelhante a um diagrama de blocos. Subindo um pouco o nível, encontra-se o nível dataflow, em que a descrição ocorre por meio de expressões lógicas e aritméticas. Por fim, temos o comportamental em que é descrito o comportamento procurado no circuito. Devido a sua proximidade com linguagens de programação, o estilo escolhido para este projeto será o comportamental, o que possibilita implementar um circuito complexo trivialmente.
Ademais, para o projeto será usada a placa DE2-115 da Altera, bem como o software Quartus II versão 20.1 e modelsim para devidos testes. O display escolhido para a saída dos resultados será o de 7 segmentos.



No display de 7 segmentos temos 7 leds ordenados de ‘’a’’ a ‘’g’’, em que podemos formar números de 0 a 9 conforme uma combinação dos leds.

Projeto

Módulo principal

À priori, é criado o módulo principal chamado calculadora. Este será responsável por definir qual operação a ser feita, bem como instanciar os outros módulos do projeto.

```
1   module calculadora(S,A,B,op,clk,rst,HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7);
2     input [7:0]A,B;
3     input  [1:0]op;
4     input clk,rst;
5     output reg [26:0]S;
6
7     //Para instanciação:
8     wire [3:0] num0, num1, num2, num3, num4, num5, num6, num7;
9     output wire [6:0]HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7;
10
11    //Estados:
12    reg [2:0]State;
13
14    //Parametros
15    parameter soma = 3'b000,
16    subtracao = 3'b001,
17    multiplicacao = 3'b010,
18    divisao = 3'b011,
19    zerado = 3'b100;
```

Os valores de entrada ‘’A’’ e ‘’B’ serão números de 8 bits, enquanto que a saída S pode chegar até 27 bits’. No entanto, devido ao uso de 8 displays, qualquer valor de operação

que exceda ao valor máximo possível (99.999.999) será tratado mais adiante.Em seguida, é criado uma variável de 2 bits chamada ‘’op’’ em que será responsável pela escolha da operação a ser realizada, com 00 para soma, 01 para subtração, 10 para multiplicação e 11 para divisão. No fpga, os valores de ’’A’’ ‘‘B’’ e ‘’op’ serão introduzidos a partir dos 18 slides switches presentes no DE2-115. Destarte, os primeiros 16 switches serão destinados aos valores dos operandos, enquanto que os dois últimos relativos à operação.

As variáveis ‘’clk’’ e “rst”representarão clock e reset respectivamente, enquanto que ‘’State’’ simboliza o estado atual do circuito. A funcionalidade do reset será implementada em um push-button.

Ainda neste módulo, utilizando do estilo comportamental do verilog, serão recebidos os valores de input “A” e “B” e, relativo a operação “op”, ocorrerão as devidas operações aritméticas.

```
12  always@(A,B,op,State)
13    begin
14      case(State)
15        soma: begin
16          S = A + B;
17          if (S > 99999999)
18            S = 0;
19        end
20        subtracao:begin
21          if(A < B)begin
22            S = (B - A);
23            if (S > 99999999)
24            S = 0;
25            end
26          else begin
27            S = A - B;
28            if (S > 99999999)
29            S = 0;
30          end
31        end
32        multiplicacao:begin
33          S = A*B;
34          if (S > 99999999)
35            S = 0;
36        end
37        divisao:begin
38          S = A/B;
39          if (S > 99999999)
40            S = 0;
41        end
42
43      endcase
44    end
```

Portanto, dependendo de qual for o estado atual, será retornado à variável de saída ‘’S’’ os valores referentes à operação em questão. É importante salientar que esta calculadora não retornará resultados negativos e, sendo assim, se subtrairmos um número menor por um número maior o resultado será positivo. Ademais, em divisões não exatas sempre será arredondado para o número menor, ou seja, em uma divisão de 9/10 o arredondamento será para 0 e não para 1. Como a divisão por 0 não está definida, o seu resultado será ‘’X’’. Em todos os casos há a presença do if que excluirá a possibilidade do resultado ser maior que o valor máximo capaz de ser exibido em 8 displays de 7 segmentos utilizados neste projeto.

Por fim, é configurado o reset e o clock para o bom funcionamento do projeto.

```
51  always@(posedge clk)
52    begin
53        if(rst)
54            begin
55            State <= zerado;
56            end
57        else  case(op)
58          soma: State <= soma;
59          subtracao: State <= subtracao;
60          multiplicacao: State <= multiplicacao;
61          divisao: State <= divisao;
62          default: State <= soma;
63        endcase
64    end
```

Sempre que houver clear o valor referente à saída é zerado, enquanto que, em borda positiva de clock, o valor da variável de operação é passada como estado atual, o que permite o controle de todo projeto.

Display(parte 1)

O display é uma das partes mais importantes na implementação de um projeto em uma placa FPGA. É nele em que será plotado o resultado de todas operações da calculadora descrita. Devido a saída ser um número de 27 bits, limitaremos para variar de 0 à 99999999,e, por conseguinte, precisará de 8 displays de 7 segmentos para exibir o

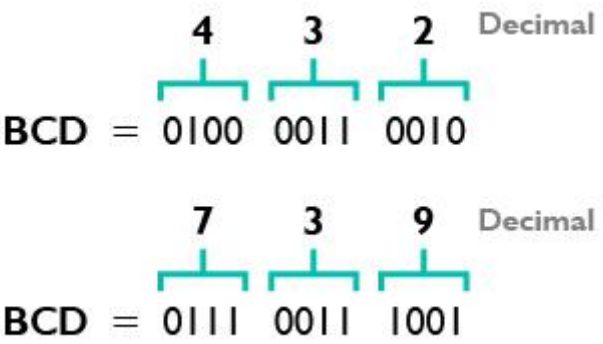
resultado por completo, o que não será um problema na placa DE2-115 utilizada no projeto. Destarte, conforme o manual da placa, os segmentos leds do display são ativos em baixo, e portanto, com ajuda do site “verilogguide.readthedocs.io” foi utilizado o seguinte módulo para tradução do resultado em dígitos no display.

```
3   input[3:0] S;
4   output reg [6:0]DATA_OUT;
5
6   always@(S)
7     begin
8     case(S)
9         0: DATA_OUT = 7'b1000000;
10        1: DATA_OUT = 7'b1111001;
11        2: DATA_OUT = 7'b0100100;
12        3: DATA_OUT = 7'b0110000;
13        4: DATA_OUT = 7'b0011001;
14        5: DATA_OUT = 7'b0010010;
15        6: DATA_OUT = 7'b0000010;
16        7: DATA_OUT = 7'b1111000;
17        8: DATA_OUT = 7'b0000000;
18        9: DATA_OUT = 7'b0010000;
19        default: DATA_OUT = 7'1111111;
20    endcase
21    end
22      endmodule
23
```

Conforme o valor de saída ‘’S’’ for variando, é perceptível que existe a combinação que exibirá esse número no display. O padrão de ordenamento de bits utilizado é o Little Endian e os leds são ordenados por “gfedcba’’. Observa-se, no entanto, que o conversor em display apenas aceita números de 4 bits e retorna o display de 7 segmentos. Isto posto, o próximo módulo será essencial para o funcionamento do código, ele fará uma ponte entre o resultado do cálculo e o display.

Conversor binário-bcd

Como supracitado, o display consegue plotar apenas dígitos de 0 a 9 e, portanto, números de, no mínimo, 4 bits. No entanto, nosso módulo da calculadora pode gerar resultados “S” de até 27 bits. É preciso, então, uma solução que sirva de ponte para esses dois módulos. Nesse sentido, no livro “Sistemas digitais: princípios e aplicações” o autor elabora o código BCD como: “Se cada dígito de um número decimal for representado por seu equivalente em binário, o resultado será um código denominado decimal codificado em binário (BCD)”. Ou seja, no código BCD, a cada 4 bits é representado um dígito, semelhante ao decimal.



Desse modo, é necessário implementar um módulo que converta o resultado “S” em BCD para exibir no display. Essa solução deve ser capaz de transformar um número binário em BCD e retornar os valores de unidade, dezenas, centenas e milhares.
Para tanto, é utilizado o algoritmo ‘’Double Dabble’’, também conhecido como ‘’shift-and-add-3 algorithm’’ que permite realizar a conversão de binário para bcd. Primeiramente, é criado um registrador de tamanho ‘’w’’ tal que permite colocar o valor binário em extenso junto ao valor bcd à ser gerado. Em vista disso, é utilizado a fórmula $n + 4\times ceil(n/3)$ visando descobrir a extensão desse registrador. Por exemplo, sendo o bit a ser convertido “ n “= 15 (1111), calculando na fórmula, é visto que é necessário um registrador de 12 bits (8 para bcd e 4 para o valor original).

BCD		
dezenas	unidades	original
'0000'	0000'	1111

Começando o algoritmo, sempre ocorrerá ‘’left-shift’’ até algum número BCD ser no mínimo 5(0101). Sempre que algum número for maior igual a 5, será somado 3 (0011) e dado ‘’left-shift’’ novamente. A operação irá terminar quando ocorrer ‘’n’’ left-shifts. Como no nosso exemplo n é um número de 4 bits, portanto a operação terminará no 4º left-shift.

BCD			
dezenas	unidades	original	
'0000'	0000'	1111	Início
0000'	0001'	1110	left-shift
0000'	0011'	1100	left-shift
0000'	0111'	1000	left-shift
0000'	1010'	1000	add3
0001'	0101'	0000'	left-shift
1	5		

Terminado o algoritmo, é retido um número convertido em BCD e o original zerado. Será feito, portanto, essa iteração em código verilog.

```
module bin2bcd
  #( parameter                W = 27)
  ( input      [26:0] bin   ,  //INPUT DE S;
   output reg [35:0] bcd   ); // RESULTADO EM BCD (36 bits pois cada 4 bits é um digito, e temos oito dígitos);

  integer i,j;

  always @(bin) begin
    for(i = 0; i <= 36; i = i+1) bcd[i] = 0;      // zerando valores
    bcd[26:0] = bin;                                    // passando valor original na direita

    for(i = 0; i <= W-4; i = i+1)           //Profundidade
      for(j = 0; j <= i/3; j = j+1)         // Largura
        if (bcd[W-i+4*j -: 4] > 4)                      // if > 4
          bcd[W-i+4*j -: 4] = bcd[W-i+4*j -: 4] + 4'd3; // add 3
  end

endmodule
```

Com a iteração realizada, é retornado um valor de 36 bits na variável BCD. Isso ocorre pois a cada 4 bits é representado um dígito, por se tratar de 8 dígitos são necessários 32 bits para representar o número máximo, os primeiros 4 bits servem apenas para realização do algoritmo e serão, portanto, descartados.
Subindo um pouco mais o nível e utilizando de artifícios do verilog, existe outra solução para este conversor binário-bcd. Como supramencionado, o verilog arredonda todas divisões para o número mais baixo. Sendo assim, e utilizando o operador de ‘%’ que retorna o valor do resto, temos:

```
module binario2bcd (numero,num0,num1,num2,num3,num4,num5,num6,num7);
    input [26:0]numero;
      output reg[3:0] num0, num1, num2, num3, num4, num5, num6, num7;
    reg [26:0]aux;

    always@(numero)begin

            num7= numero/10000000;
            aux = numero%10000000;
            num6 = aux/1000000;
            aux = aux%1000000;
            num5 = aux/100000;
            aux= aux%100000;
            num4 = aux/10000;
            aux= aux%10000;
            num3 = aux/1000;
            aux= aux%1000;
            num2 = aux/100;
            aux= aux%100;
            num1 = aux/10;
            aux= aux%10;
            num0 = aux;

    end
endmodule
```

Quando é dividido, por exemplo, num1 por 10, o valor do dígito correspondente a dezena é isolado, em seguida o resto é passada para um registrador auxiliar para realizar a operação com o próximo dígito. Desta forma, utilizando o método para todas as casas é possível separar os dígitos de forma mais simplificada e direta.

Display(parte 2)

Haja vista as implicações do conversor binário-bcd, é necessário fazer algumas reformulações no módulo do display.

```
module C7SEG(num0, num1, num2, num3, num4, num5, num6, num7,HEX0,
HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7);
  input[3:0] num0, num1, num2, num3, num4, num5, num6, num7; //
Dígitos/bcd

  //Cada HEX é um display
  output reg [6:0]HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7;

  //Criando parametros do display 7segmentos
  parameter zero=7'b1000000 , um = 7'b1111001, dois = 7'b0100100,
tres = 7'b0110000,quatro = 7'b0011001, cinco = 7'b0010010, seis =
7'b0000010, sete =  7'b1111000, oito =  7'b0000000, nove =
7'b0010000;
```

São criadas variáveis para cada display usado (HEX0 à HEX 7). Ademais, para facilitação, os parâmetros são criados com os valores necessários para ascender os números no display de 7 segmentos. Em seguida é implementado os cases para cada dígito bcd, passando-os para os displays:

```
9   always@(num0,num1,num2,num3,num4,num5,num6,num7)
10    begin
11     //Passando digito1 para o display1
12      case(num0)
13        0: HEX0 = zero;
14            1: HEX0 = um;
15            2: HEX0 = dois;
16            3: HEX0 = tres;
17            4: HEX0 = quatro;
18            5: HEX0 = cinco;
19            6: HEX0 = seis;
20            7: HEX0 = sete;
21            8: HEX0 = oito;
22            9: HEX0 = nove;
23        default: HEX0 = zero;
24      endcase
```

```
26        case(num1)
27          0: HEX1 = zero;
28          1: HEX1 = um;
29          2: HEX1 = dois;
30          3: HEX1 = tres;
31          4: HEX1 = quatro;
32          5: HEX1 = cinco;
33          6: HEX1 = seis;
34          7: HEX1 = sete;
35          8: HEX1 = oito;
36          9: HEX1 = nove;
37            default: HEX1 = zero;
38          endcase
39
```

Todos os display devem ser atendidos, e portanto,os cases seguirão até o num7 e HEX7. Desta forma, é garantido que o display receba os valores nos conformes.

Instâncias

Como supracitado, será usado o módulo ‘’calculadora’’ para realização das instâncias dos módulos do projeto. Para sintetização é preciso de uma variável tipo NET para instanciar. É criado, portanto, registradores de num0 à num7 para armazenar os valores bcds após a conversão, e passar para o módulo do display.

```
74   binario2bcd bcd(.numero(S),.num0(num0) , .num1(num1), num2(num2), .num3(num3),
.num4(num4), .num5(num5), .num6(num6), .num7(num7));
75
76   C7SEG seg7(.num0(num0) , .num1(num1), num2(num2), .num3(num3), .num4(num4),
.num5(num5), .num6(num6), .num7(num7), .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2),
.HEX3(HEX3), .HEX4(HEX4),.HEX5(HEX5),.HEX6(HEX6), .HEX7(HEX7));
```

É passado para o conversor o valor do resultado das operações “S” e este retorna o valor S em bcd (num0 à num7). Em seguida, é passado o valor em bcd para o módulo do 7segmentos enquanto retorna os valores do display, e assim, finalizando o design do projeto.

Conclusões

Testbenchs

Para a realização dos testbenchs do projeto, foi utilizada a metodologia baseada em task e funções. Isso ocorrerá pois esse procedimento é o mais flexível, o que nos permite focar em apenas uma funcionalidade (como somar, subtrair, dividir e multiplicar) o que propicia maior facilidade na manutenção.
Isto posto, partiremos para o esboço da implementação. Será feita uma task para realização de soma no Testbench no módulo principal “calculadora”:

```
//Task para soma:
task somar(input integer num1, input integer num2);
  begin
    A = num1;
    B = num2;
    op = 00;
    toggle_clk;
  end
endtask
```

Agora podemos instanciar a task, passando assim valores aleatórios utilizando do artifício “$random” do verilog.

```
initial begin
  repeat(10)begin
    somar($random, $random);
    display;end
```

Desta forma, serão feitas 10 repetições de soma, as quais podemos validar no console:

```
# KERNEL: A: 36,B:  1 ,S  37
# KERNEL: A:  9,B: 99 ,S 108
# KERNEL: A: 13,B: 13 ,S  26
# KERNEL: A:101,B: 18 ,S 119
# KERNEL: A:  1,B: 13 ,S  14
# KERNEL: A:118,B: 61 ,S 179
# KERNEL: A:109,B: 12 ,S 121
# KERNEL: A:121,B: 70 ,S 191
# KERNEL: A: 69,B: 42 ,S 111
# KERNEL: A:101,B:119 ,S 220
```

Destarte, as task serão feitas para cada operação da nossa calculadora, particionando, assim, nosso testbench e facilitando eventuais manutenções no código.
Nesse sentido, da mesma forma será feito o verificador do algoritmo bcd e do display de 7 segmentos.

```
3  module teste2;
4    reg [26:0] numero;
5    wire [3:0] num0, num1, num2, num3, num4, num5, num6, num7;
6
7    bin2bcd conv(.numero(numero), .num0(num0), .num1(num1),
   .num2(num2), .num3(num3), .num4(num4), .num5(num5), .num6(num6),
   .num7(num7));
8    initial begin
9      repeat(10)
10       begin
11         converter($random);
12       end
13
14   end
15   task converter(input integer N); //Task para conversão
16     begin
17         numero = N; // É passado um número aleatorio para numero
18         display;
19     end
20
21   endtask
22  task display; //task para printar no console;
23    #1 $display("bin :%d, bcd:%d %d %d %d %d %d %d ",
24                numero, num7, num6, num5, num4, num3, num2, num1,
   num0);
25   endtask
26
27 endmodule
```

Referências Bibliográficas

1. PCS-EPUSP. **Calculadora Simples em VHDL**. Apostila de PCS 2308/2355, 2015. Disponível em:
2. https://www2.pcs.usp.br/~labdig/pdffiles_2015/calculadora-vhdl-semestral.pdf;
3. https://verilogguide.readthedocs.io/en/latest/verilog/vvd.html#seven-segment-display;
4. TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas digitais: princípios e aplicações.** 11ª ed., Pearson, 2011;
5. https://en.wikipedia.org/wiki/Double_dabble;
6. https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1404062209-de2-115-user-manual.pdf