Calculadora Verilog

Bernardo Fraga
João Terra
Raul Costa
Pedro Oliveira

Objetivo:

O objetivo desse projeto é criar uma calculadora básica, utilizando Verilog, capaz de realizar operações aritméticas básicas entre dois números - até um total de 32 bits -, sendo elas: adição, subtração e multiplicação (utilizando somas consecutivas) e o comando de igualdade, responsável por executar as operações. Também será possível limpar a calculadora e visualizar uma mensagem de erro quando o display chegar no limite de 32 bits.

Queremos fazer um arquivo de testbench, chamado tb_final.sv, para que seja possível testar a calculadora. Nele, iremos simular possíveis entradas de um usuário, testando situações incomuns e de possíveis erros. Utilizaremos a ferramenta Questa para simular o clock e o reset, assim como visualizar as ondas (sinais) subindo e descendo.

Metodologia:

Antes de começar a desenvolver toda a lógica da calculadora, desenvolvemos dois arquivos responsáveis por controlar 8 displays de 7 segmentos cadas, os quais irão ser usados para mostrar os números da calculadora. São eles o Display.sv, responsável por mostrar um dígito em um display individualmente, e o Display Ctrl.sv, responsável por usar o Display.sv para controlar as posições de cada dígito entre os 8 displays.

O módulo principal do projeto é a Calculadora_Top.sv, ela é responsável por receber 3 informações: CMD, CLOCK, RESET. O CMD é o **comando** do usuário, de 4 bits, ele é levado para o módulo Calculadora.sv. Nesse módulo é onde ocorre toda lógica da calculadora.

Primeiramente, o código dentro do módulo Calculadora.sv utiliza um bloco sequencial “always”, sensível à borda do **clock** e do **reset**, implementado uma lógica simples com if/else dentro, o qual espera um dígito de 0 à 9 ou uma operação.

Se identificar que é um dígito, ele atribui a um registrador chamado ***Reg1*** o valor desse dígito e continua atribuindo mais valores caso o usuário digite mais números. Se o programa identificar que não é um dígito, ele entende que só pode ser uma operação.

Se essa operação **não for** “igual a”, o comando do usuário é armazenado como operador e a variável ***set_op*** recebe “1”. Agora como é ***set_op*** é diferente de zero, o

primeiro *else if* é ativo, fazendo com que o comando do usuário seja armazenado no ***Reg2***. Após isso, o programa, novamente, espera outra operação. Se essa operação for "igual a", então o programa entra no bloco condicional *"else if (cmd == 4'b1110)*", onde existe um case para identificar qual operação realizar.

Conforme o valor da operação, salvo anteriormente, o programa realiza quatro possíveis ações:

- **Adição:** Faz uma soma simples entre **Reg1** e **Reg2**.
- **Subtração:** Faz uma subtração simples entre **Reg1** e **Reg2**.
- **Multiplicação:** Usa somas em sequência para realizar a multiplicação.
  - Se Reg1 e Reg2 for zero, a saída será zero.
  - Se não, o estado da calculadora passa para OCUPADO. Nesse estado, a cada ciclo do clock o valor de Reg1 é somado à saída e o contador incrementado até chegar ao valor de Reg2. Quando o contador for igual ao Reg2, o estado volta para PRONTA.

- **Backspace:** Apaga o último dígito inserido no Reg1, fazendo uma divisão por 10. Caso o valor de *set_op* já tenha mudado para 1, ele faz essa divisão no Reg2.

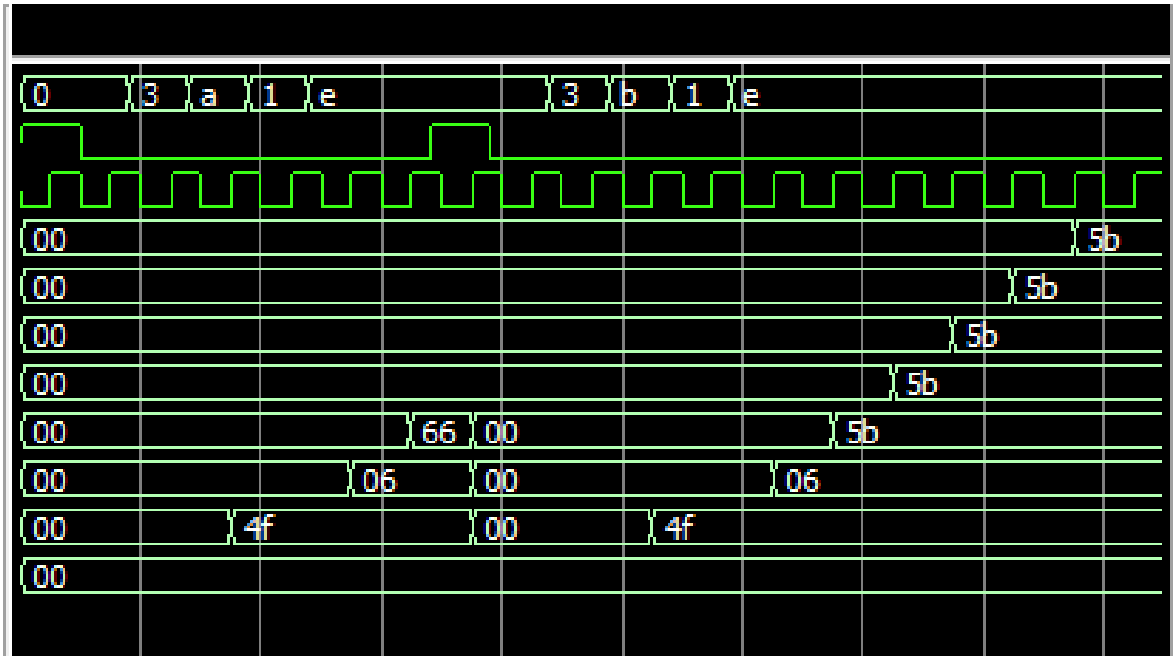
Para a lógica das posições no display, fizemos com que a cada dígito recebido, o dígito vai para o *dig* do display e a *pos* é incrementada em um, pulando assim para o próximo display.

Testagem:

Para testar o código, fizemos um simples *test bench* o qual simula entradas no comando a cada ciclo de um *clock.* Nas testagens encontramos erros, possivelmente erros de lógica na calculadora.sv. Percebemos que o primeiro display é pulado, mesmo utilizando uma lógica para que o *pos* não seja incrementado na primeira vez. Também tivemos problemas com números com mais de um algarismo, mesmo utilizando a lógica que multiplica o reg1/reg2 por 10 e em seguida acrescenta o comando do usuário. Nas testagens foi possível observar que os valores dos registradores e dos resultados só são mostrados dois clocks após a entrada simulada. Outro pequeno problema foi que, o último resultado fica se repetindo até acabar os displays, caso não recebe mais nenhum comando após ele.

```
// Geração de clock
  initial clock = 0;
  always #5 clock = ~clock;

  initial begin
    // Inicialização
```

```
        cmd   = 4'b0000;
        reset = 1;
    #10 reset = 0;

    // ======================
    // Teste 1: 3 + 1 =
    // ======================
    #8  cmd = 4'b0011;  // 3

    #10 cmd = 4'b1010;  //+

    #10 cmd = 4'b0001;  //1

    #10 cmd = 4'b1110;  // =

    #20 reset = 1;

    #10 reset = 0;

    // ======================
    // Teste 2: 3 - 1 =
    // ======================

    #10 cmd = 4'b0011;  // 3

    #10 cmd = 4'b1011;  //-

    #10 cmd = 4'b0001;  //1

    #10 cmd = 4'b1110;  // =
    #100;

    $stop;
end
```



Os números acima são representações em hexadecimal dos displays, representados em binário, por isso fizemos essa simples tabela para ajudar na compreensão:

0 - 3F
1 - 06
2 - 5B
3 - 4F
4 - 66
5 - 6D
6 - 7D
7 - 07
8 - 7F
9 - 6F

a - Soma
b - Subt
c - Mult
d - Vazio
e - Igual