Lógica Digital (1001351)

Experimento NR.3

Implementação de circuitos e *test benches* em Verilog. [1]

Aviso ENPE

Esta prática foi adaptada para ser realizada em simuladores. As instruções originais foram mantidas para que se tenha uma ideia do procendimento real. Ao escrever o relatório, deve-se destacar quando não foi possível obter a informação solicitada por limitação do simulador usado.

1 Instruções Gerais

- Grupos definidos no AVA, só incluir os nomes de quem efetivamente participou;
- Ler atentamente todo o procedimento desta experiência antes de realizá-la;

2 Objetivos da Prática

- Ambientação introdutória ao simulador EDAplayground;
- Manipulação introdutória de arquivos bssicos correspondentes a *test bench* e a código de descrição de circuito;
- Execução de esperimentos em simulador.

3 Materiais e Equipamentos

- Simulador EDAplayground.
- Exemplo de implementação/teste multiplexador;
- Exemplo de implementação/teste leds (Tutorial Vivado).

[1] Prof. Mauricio Figueiredo e Prof. Ricardo Menotti (Revisão: 1 de setembro de 2021)

4 Orientações para a prática no laboratório

1. Observe atentamente os códigos fornecidos e seu comportamento antes de alterar qualquer coisa;
2. Faça uma cópia dos códigos a serem modificados, assim você pode comparar com os originais sempre que for preciso;
3. Procure associar cada linha do código ao circuito que você está implementando/testanto;
4. Faça alterações por etapas, assim você não corre o risco de modificar muito o código de uma vez e inserar vários erros difíceis de detectar.

5 Fundamentos teóricos

Nesta seção, será descrito o básico sobre o simulador e a linguagem Verilog necessários para o seu laboratório.

5.1 EDAplayground

Esta ferramenta é na verdade uma interface para simuladores (inclusive comerciais). Na barra lateral da esquerda você pode selecionar em **Tools & Simulators** qual ferramenta deseja usar.

O principal a saber é que a interface do simulador possui dois painéis de código.

- No painel da direita está o código da sua implementação, ou seja, o circuito que você está implementando.
- No painel da esquerda está o *test bench* que serve para testar o seu circuito.

O *test bench* instancia seu circuito, gera estímulos de entrada para ele e verifica se as saídas esperadas estão corretas (ou simplesmente as exibe em um diagrama de forma de ondas (*waveform*) para você conferir visualmente).

5.2 Verilog

Como já mencionamos antes, o Verilog é apenas uma ferramenta que usamos para compreender a lógica digital e seus circuitos. Nosso objetivo não contempla compreender toda a sintaxe da linguagem durante o curso. Um subconjunto dela já é suficiente para atingir o nosso objetivo.

O exemplo do multiplexador usa entradas e saídas simples, conforme a declaração abaixo:

```
        input x1, x2, s,
        output f);
```

Já o exemplo dos leds usa barramentos (bus) como entradas e saídas, eles são como arranjos de fios simples e possuem um índice para acessar cada fio separadamente. No exemplo a seguir, tanto a entrada quanto a saída possuem 4 fios cada:

```
    input [3:0] swt,
     output [3:0] led);
```

O *test bench* precisa conhecer as saídas corretas (esperadas) e compará-las com as obtidas do circuito para saber se o circuito realmente corresponde à implementação desejada. Sendo assim, ao longo do tempo, o *test bench* aplica ao circuito cada uma das entradas possíveis, uma por vez, e compara as respectivas saídas do circuito com os seus pares esperados.

A modelagem do tempo, para efeitos da aplicação das entradas no circuito, presente no *test bench*, é obtida por meio do sinal # , usado para gerar atrasos na simulação, ou seja, significam esperas. Além disso, o *test bench* usa um sinal de *clock* (clk) que consiste em uma sequência de pulsos, construído com os atrasos gerados pelo sinal #, para sequenciar o teste. Cada vez que o *clock* alterna, um caso de teste é verificado. Na subida de cada pulso/clock (posedge) o *test bench* atribui valores à entrada do circuito e, na descida (negedge), lê a saída do circuito e compara com a desejada.

No exemplo dos leds, o *test bench* gera as saídas desejadas (corretas) a partir de uma seção de código idêntica àquela referente à implementação do circuito que se deseja verificar. Além disso, usa uma repetição do tipo *for* para avançar no tempo. A cada iteração, da repetição, *test bench* muda a entrada e confere a saída. Neste exemplo, note que o número de iterações é maior do que o de entradas possíveis, então há casos repetidos que já foram testados e não influenciarão na verificação.

Já no exemplo do multiplexador, o *test bench* usa um arquivo de texto com as entradas possíveis e as saídas esperadas, neste caso em hexadecimal. Apenas "0"ou "1"estão presentes (correspondendo a "0000"e "0001"em binário), claro, pois as saídas das portas lógicas correspondem aos valores lógicos 0 e 1. Desta forma, tomando-se valores de quatro em quatro bits (apenas os menos significativos de cada digito hexa) constroi-se a palavra binária desejada. Por exemplo, a última linha do arquivo em hexa é 111_1, a qual, se executadas as devidas substituições para um sistema binário, corresponde à linha em binário 000100010001_0001. Consequentemente são considerados os bits 0, 4, 8 e 12, no caso todos bits 1, desprezando as posições onde estão os zeros, para formar a linha de teste 111_1 em binário.

Marque a caixa **Open EPWave after run** (na barra de configurações à esquerda da janela do simulador) para exibir o diagrama de forma de ondas.

6 Procedimentos Experimentais

Passo 1: Simule o exemplo do multiplexador e observe suas saídas:

Etapas-1: Observe que as mensagens na parte inferior da interface (**Log**);

Etapas-2: Observe as saídas no diagrama de forma de ondas;

Etapas-3: Observe as diferentes formas de implementar o mesmo circuito;

Etapas-4: Observe como as entradas do *test bench* são geradas a partir do arquivo;

Etapas-5: Procure associar o arquivo `values.tv` com estas saídas.

Etapas-6: Modifique um ou mais valores na saída e veja se o *test bench* é capaz de detectar a inconssitência. Apresente suas sugestões, com comentários pertinentes, no relatório.

Passo 2: Simule o exemplo dos leds e observe suas saídas:

Etapas-1: Observe que as mensagens na parte inferior da interface (**Log**);

Etapas-2: Observe as saídas no diagrama de forma de ondas;

Etapas-3: Observe como as entradas do *test bench* são geradas;

Etapas-4: Que forma de implementação este circuito usa?

Etapas-5: Procure associar a função lógica esperada com estas saídas.

Etapas-6: Modifique a função do teste:

```
1    function [3:0] expected_led;
2       input [3:0] swt;
3    begin
4       expected_led[0] = ~swt[0];
5       expected_led[1] = swt[1] & ~swt[2];
6       expected_led[3] = swt[2] & swt[3];
7       expected_led[2] = expected_led[1] | expected_led[3];
8    //   expected_led[7:4] = swt[7:4];
9    end
10   endfunction
```

Verifique se o *test bench* é capaz de detectar a inconssitência. Apresente sua sugestão, com comentários pertinentes, no relatório.

Passo 3: Reimplemente o exemplo dos leds usando:

Etapas-1: Um *test bench* baseado em um arquivo de vetores de teste (`values.tv`);

Etapas-2: A forma de implementação estrutural;

Passo 4: Questões:

Questão-1: Com respeito ao projeto "MULTIPLEXADOR": A sentença para declaração da variável "testvectors"considera uma dimensão de capacidade muito acima da necessária: "logic [103:0] testvectors[10000:0]".

De que forma deveria ser reescrita essa sentença para que considerasse a capacidade exata para a necessidade da simulação.

Questão-2: Com respeito ao projeto "MULTIPLEXADOR": Há várias maneiras de instanciar o projeto "mux1": mux1 dut1(.x1(x1), .x2(x2), .s(s), .f(f1)). Corrija, se necessário, as seguintes sentenças alternativas:

- mux1 dut1(.x2(x2), .x1(x1), .s(s), .f(f1))
- mux1 dut1(x2, x1, s, f1)

Questão-3: Com respeito ao projeto "MULTIPLEXADOR": Caso o "testbench.sv"devesse considerar os dados no arquivo "values.tv"em binário, como as sentenças em seguida, presentes no arquivo "testbench.sv", deveriam ser reescritas (para compatibilizar as dimensões das variáveis):

```
1   // at start of test, load vectors
2  initial begin
3    $readmemh("values.tv", testvectors);
4    vectornum = 0; errors = 0;
5  end
6
7    // apply test vectors at rising edge of clock
8  always @(posedge clk)
9    begin
10     x1 = testvectors[vectornum][12];
11     x2 = testvectors[vectornum][8];
12     s = testvectors[vectornum][4];
13     f_expected = testvectors[vectornum][0];
14     display;
15   end
```

Questão-4: Com respeito ao projeto "LEDS": Houve redundância de simulação para teste do circuito? Justifique. De que forma a seção de código em seguida deve ser alterada para que essa redundância não ocorra?

```
1   begin
2             $dumpfile("dump.vcd"); $dumpvars;
3     for (i=0; i < 255; i=i+1)
4       begin
5           #50 switches=i;
6           #10 e_led = expected_led(switches);
7           if(leds == e_led)
8               $display("LED output matched at", $time);
9           else
10              $display("LED output mis-matched at ",$time,
11              ": expected: %b, actual: %b", e_led, leds);
12      end
13  end
```

Passo 5: Elaborar relatório simplificado consistindo de: página de rosto (com identificação da prática e integrantes entre outras informações, tal como no relatório padrão) e seção de resultados contendo: respostas e comentários solicitados (indicando os enunciados correspondentes) e o link da implementação no AVA.