SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL

Trabalho Prático 01 - ISL

Circuitos Combinacionais

Pâmela Lúcia Lara Diniz - 5898
Júlio César de Souza Oliveira - 5903

Florestal - MG
2024

Sumário

1. Introdução **3**

2. Organização **3**

3. Desenvolvimento **4**

3.1 Módulo verificação de paridade 4

3.2 Módulo display 4

4. Compilação e Execução **4**

5. Resultados **5**

6. Conclusão **5**

7. Referências **5**

1. Introdução

Ao trocar mensagens codificadas em binários de 5 bits, um grupo de amigos encontrou dificuldades relacionadas à entradas incorretas. Nesse contexto, foi feita a implementação de um hardware na linguagem Verilog para exibir um caractere em um display de sete segmentos caso suas especificações sejam correspondidas. O objetivo principal do projeto é garantir a identificação de erros na entrada, tanto de paridade quanto de representação de caracteres que não pertencem ao intervalo abrangido pelo display. No que se refere à paridade, será acrescentado um bit extra junto às entradas para realizar essa verificação, se estiver correta e o caractere for válido, a mensagem será exibida corretamente no display. Caso contrário, o display mostrará o número 8, indicando um erro de paridade ou se manterá apagado para representar um caractere que não pertence ao intervalo que pode ser representado.

2. Organização

Na Figura 1 é possível visualizar a organização do projeto. Para cada módulo (Módulo de Verificação de Paridade e Módulo de Mapeamento para o Display de 7 Segmentos) foi criado uma pasta para armazenar seus arquivos. Na pasta **Circuitos/** está as imagens dos Diagramas de portas lógicas do módulo. Na pasta **Codigo/** está a implementação do código em Verilog dos módulos implementados e do módulo de teste (testbench), que contém também o arquivo de simulação de ondas (.vcd). Na pasta **Tabela_e_mapas_karnaugh/** está as imagens da Tabela verdade e dos Mapas de Karnaugh com suas expressões booleanas simplificadas.

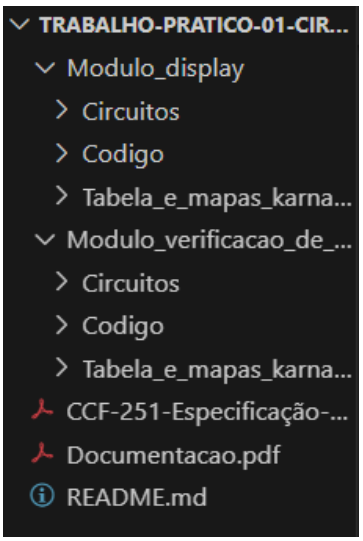

Figura 1 - Repositório do projeto.

Além disso, dentro do repositório encontram-se as especificações do trabalho e a Documentação.

3. Desenvolvimento

Nesta seção, serão destacados os principais detalhes das implementações do projeto, enfatizando-se os pontos que permitiram a criação e funcionamento de cada módulo e do circuito num todo.

3.1 Módulo de verificação de paridade

Primeiramente, para a implementação desse módulo, foi feita a tabela verdade com todas as combinações possíveis para os 5 bits da mensagem mais o bit de paridade, e sua saída se deu pela comparação entre a quantidade de bits 1 contidos na mensagem e sua equivalência com o bit verificador. A partir disso, a conclusão obtida foi que seria possível realizar tal verificação fazendo o uso da porta XOR entre todas as seis entradas e a sua negação ao final do circuito, obtendo-se uma saída (1) sempre que a paridade estiver correta e (0) quando a paridade estiver errada. Com essa lógica estabelecida, segue-se para as próximas etapas do desenvolvimento, a sua representação em um esquema de portas lógicas (figura 2), seguido da implementação em uma linguagem de descrição de hardware(figura 3).

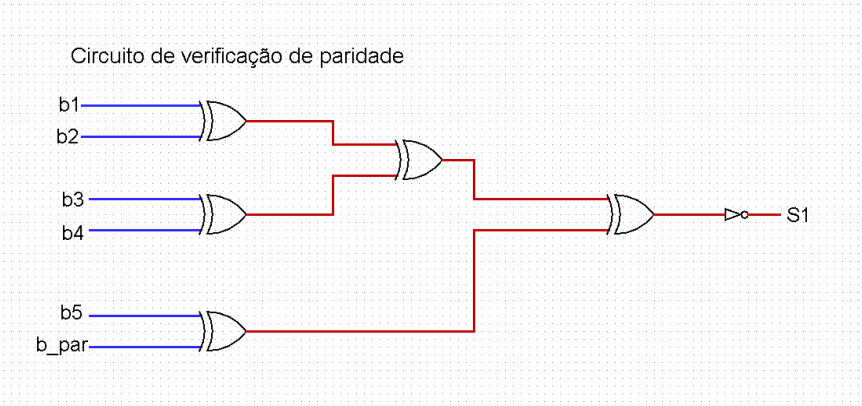

Figura 2 - Diagrama de portas lógicas do módulo de verificação de paridade.

O diagrama de portas lógicas é uma representação de operações lógicas que podem ser realizadas em circuitos digitais. No caso do módulo atual, foi retratada a combinação de portas XOR seguidas de um NOT ao final. A representação na figura 2 foi criada com o auxílio do software “Logisim”, um simulador lógico que permite o desenho e a simulação de circuitos através de uma interface gráfica.

```
module verificador_paridade (
    input wire b1,b2,b3,b4,b5,bp,
    output wire S
);
    assign S=~(b1^b2^b3^b4^b5^bp);

endmodule
```

Figura 3 - Módulo de verificação de paridade desenvolvido na linguagem Verilog.

No que se refere ao desenvolvimento do código, a IDE Visual Studio Code foi a ferramenta utilizada para a implementação, na linguagem Verilog, da lógica relacionada ao verificador de paridade. No início do módulo foi feita a declaração das entradas e da saída e em seguida foi realizada a atribuição do resultado da expressão booleana obtida de análises anteriores.

Por conseguinte, foi desenvolvido um arquivo testbench para testar e simular o comportamento do circuito em todos os casos de entrada representados na tabela verdade, o que confirmou a efetividade da lógica utilizada e possibilitou o avanço para a etapa seguinte, a implementação do display de sete segmentos.

3.2 Módulo display

Para o funcionamento desse módulo, foi necessário implementar a tabela verdade com sete saídas, cada uma representando um segmento do display. A

partir daí foram registradas as 20 combinações de bits de entrada (figura 4) que poderiam representar um caractere válido e feita a sua configuração para que a saída representasse corretamente os segmentos a serem ativados ou apagados no display no que se refere a cada dígito. Essa lógica foi implementada levando em consideração apenas os 32 casos em que a paridade da entrada esteja correta, uma vez que, caso isso não se concretize, o circuito exibe imediatamente o caractere 8 (erro de paridade), dispensando verificações adicionais.

Bits	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000	10001	10010	10011
Char	V	B	E	1	7	R	P	Y	0	K	M	H	S	Q	C	G	T	N	W	F

Figura 4 - Caracteres que podem ser representados pelo display e suas codificações binárias.

Posteriormente, utilizando-se das informações organizadas na tabela verdade, foi possível representar a saída de cada segmento do display a partir de uma expressão booleana simplificada, obtida com o uso do método de Karnaugh. Essa simplificação possibilitou a criação dos esquemas de portas lógicas para cada segmento e também facilitou a implementação da lógica na linguagem de descrição de hardware, Verilog (figura 5).

```
verificador_paridade verifica(b1,b2,b3,b4,b5,b_par,S);
always@(*) begin
    if(S == 1'b0)begin
        A <= 1'b1;
        B <= 1'b1;
        C <= 1'b1;
        D <= 1'b1;
        E <= 1'b1;
        F <= 1'b1;
        G <= 1'b1;

    end
    else if ((b1 == 1'b1 & b2 == 1'b1) | b1 == 1'b1 & b3 == 1'b1)begin
        A <= 1'b0;
        B <= 1'b0;
        C <= 1'b0;
        D <= 1'b0;
        E <= 1'b0;
        F <= 1'b0;
        G <= 1'b0;
    end
    else if(S == 1'b1)begin
        A <= (((~b1) & b4 & (~b5)) | ((~b1) & b2 & b3) | ((~b1) & b3 & (~b5)) | ((~b1) & b2 & (~b5)));
        B <= (((~b1) & b4 & b5) | ((~b1) & b2 & b5) | ((~b1) & (~b2) & b3 & (~b5)) | (~b1) & b2 & (~b3) & (~b4) | (b1 & (~b2) & (~b3) & b4 & (~b5)));
        C <= (((~b1) & (~b4) & (~b5)) | ((~b1) & b4 & b5) | ((~b1) & b2 & (~b3)) | ((~b1) & b2 & (~b4)) | ((~b2) & (~b3) & (~b4) & b5));
        D <= (((~b2) & (~b3) & (~b5)) | ((~b1) & (~b2) & (~b3) & (~b4)) | ((~b1) & b3 & b4 & b5) | ((~b1) & b2 & b3 & b4) | ((~b1) & b2 & (~b4) & (~b5)));;
        E <= (((~b1) & b2 & (~b3)) | ((~b1) & b4 & (~b5)) | ((~b2) & (~b3) & (~b4)) | ((~b1) & (~b2) & (~b4) & b5) | (b1 & (~b2) & (~b3) & b5));
        F <= (((~b1) & b2 & (~b4)) | ((~b1) & b2 & b5) | ((~b1) & b3 & b4) | ((~b1) & (~b2) & b4 & (~b5)) | ((~b1) & (~b3) & (~b4) & b5) | (b1 & (~b2) & (~b3) & (~b5)
        G <= (((~b1) & (~b4) & b5) | ((~b1) & b3 & b5) | ((~b1) & b2 & b5) | ((~b1) & (~b2) & b4 & (~b5)) | ((~b1) & b2 & b3 &(~b4)) | (b1 & (~b2) & (~b3) & (~b4)));
    end
end
```

Figura 5 - Lógica do display implementada na linguagem Verilog.

Na estruturação do código, a primeira verificação realizada pelo bloco “if” é responsável por identificar problemas de paridade na entrada. Caso aconteça algum erro, o sistema atribui aos segmentos do display o caractere relacionado “8” (A,B,C,D,E,F,G igual a 1) e encerra o funcionamento do circuito. Caso contrário, considera-se que a paridade é válida e é realizada uma segunda verificação que faz uso de algumas combinações de bits que não abrange nenhum caractere do intervalo, conforme observado a partir da tabela verdade. Dessa forma, é possível

reconhecer imediatamente um elemento inválido e atribuir um sinal para que o display permaneça apagado (A,B,C,D,E,F,G igual a 0).

Ademais, caso não sejam identificados erros nas etapas anteriores, o último bloco condicional calcula a saída de cada segmento do display com base nas entradas e nas expressões booleanas simplificadas, possibilitando assim, a efetiva exibição da mensagem codificada.

Por fim, foi desenvolvido um arquivo testbench para testar e simular a execução de todo o circuito, o que permitiu a obtenção de resultados satisfatórios de acordo com a especificação fornecida, que podem ser observados no capítulo 5 deste documento.

4. Compilação e Execução

Para compilar os arquivos é necessário que você esteja na pasta onde o arquivo está localizado, .\Modulo_display\Codigo por exemplo, e utilize o comando “icarusverilog -o nomeParaOexe.vvp nomeDoArq.v”. Para executar utilize o comando “vvp nomeParaOexe.vvp” . Se você compilou e executou o módulo de teste (testbench) será gerado um arquivo.vcd, com ele é possível abrir o gtkwave e visualizar a forma de ondas com o comando “gtkwave arquivo.vcd”.

Esses comandos foram executados a partir do Sistema Operacional windows, utilizando o compilador icarus verilog.

5. Resultados

Destarte, uma vez que os módulos sejam compilados é possível executá-los. No entanto, ao executar apenas os módulos de síntese não ocorrerá nenhuma interação com o usuário, já que esse código é usado pela FPGA, o qual ainda não temos acesso para esse trabalho. Por outro lado, ao executar o módulo de teste (testbench) dos módulos será gerado uma saída via terminal, como na imagem 7 e na imagem 9, e um arquivo .vcd para visualizar a forma de ondas do módulo, como na imagem 6 e imagem 8.

Ao executar o testbench do módulo de verificação de paridade será exibido estímolos gerados pelo testbench e suas respectivas saídas. As entradas

b1,b2,b3,b4,b5 são os bits da mensagem e o bp é o bit de paridade. Uma vez que comparado o bit de paridade (bp) com a paridade da mensagem (b1,b2,b3,b4,b5) a saída S será 1 para paridade correta e 0 para paridade incorreta.

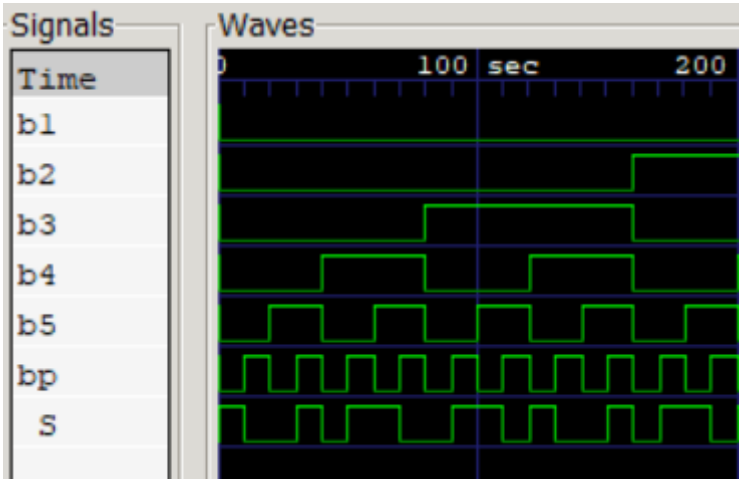

Figura 6 - Recorte da forma de ondas do módulo de paridade

```
VCD info: dumpfile v_paridade.vcd opened for output.
b1 b2 b3 b4 b5 bp - S
0  0  0  0  0  0  1
0  0  0  0  0  1  0
0  0  0  0  1  0  0
0  0  0  0  1  1  1
0  0  0  1  0  0  0
0  0  0  1  0  1  1
0  0  0  1  1  0  1
0  0  0  1  1  1  0
0  0  1  0  0  0  0
0  0  1  0  0  1  1
0  0  1  0  1  0  1
0  0  1  0  1  1  0
0  0  1  1  0  0  1
0  0  1  1  0  1  0
0  0  1  1  1  0  0
0  0  1  1  1  1  1
0  1  0  0  0  0  0
0  1  0  0  0  1  1
0  1  0  0  1  0  1
0  1  0  0  1  1  0
0  1  0  1  0  0  1
```

Figura 7 - Parte da saída pelo terminal do testbench do módulo de paridade

De forma similar, ao executar o testbench do módulo de Mapeamento para o Display de 7 Segmentos será exibido possíveis entradas e suas respectivas saídas. As entradas b1,b2,b3,b4,b5 são os bits da mensagem e o bit b_par é o bit de paridade. Assim, verifica se a paridade está correta pois caso não esteja o display irá exibir o caractere especial. Por outro lado, se a paridade está correta será verificado se os

bits compõem uma mensagem dentro do intervalo dos 20 símbolos, pois se estiverem fora do intervalo e com paridade correta o display deve permanecer apagado (as saídas A,B,C,D,E,F,G serão 0). De forma similar, se a mensagem está dentro do intervalo e sua paridade está correta as saídas A,B,C,D,E,F,G são configuradas (0 ou 1) de forma a constituir o símbolo da mensagem (isto é, uma string de bits correspondente ao símbolo uma vez que não temos a FPGA). Porém quando a mensagem estiver dentro do intervalo e sua paridade não for válida aparecerá o caractere especial escolhido, o caractere “8” (A,B,C,D,E,F,G igual a 1).

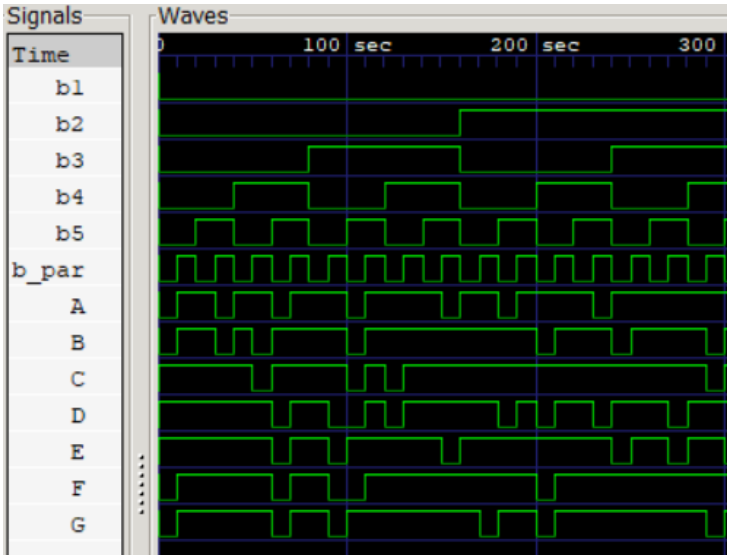

Figura 8 - Recorte da Forma de ondas

```
VCD info: dumpfile display.vcd opened for output.
b1 b2 b3 b4 b5 b_par-A B C D E F G
0  0  0  0  0  0     0 0 1 1 1 0 0
0  0  0  0  0  1     1 1 1 1 1 1 1
0  0  0  0  1  0     1 1 1 1 1 1 1
0  0  0  0  1  1     0 0 1 1 1 1 1
0  0  0  1  0  0     1 1 1 1 1 1 1
0  0  0  1  0  1     1 0 0 1 1 1 1
0  0  0  1  1  0     0 1 1 0 0 0 0
0  0  0  1  1  1     1 1 1 1 1 1 1
0  0  1  0  0  0     1 1 1 1 1 1 1
0  0  1  0  0  1     1 1 1 0 0 0 0
```

Figura 9 - Parte da Saída pelo terminal do testbench do módulo do display

6. Conclusão

O código desenvolvido neste projeto teve como foco a implementação de um display de sete segmentos para exibir mensagens codificadas em binário, garantindo a identificação de possíveis erros. Neste processo, foram utilizados conceitos de lógica matemática e computacional integrados em circuitos combinacionais com o auxílio da linguagem de descrição de hardware Verilog, para implementar e processar toda a lógica necessária ao funcionamento do circuito.

Os resultados obtidos foram satisfatórios e coerentes com a especificação fornecida. O sistema conseguiu processar corretamente as entradas, identificar problemas de paridade ou caracteres errados e informar uma saída correta.

Dessa forma, o desenvolvimento do projeto não apenas atendeu aos requisitos estabelecidos, mas também possibilitou a aplicação prática dos conceitos aprendidos na disciplina de Introdução aos Sistemas Lógicos, caracterizando uma experiência enriquecedora por meio da interação com problemas reais.

7. Referências

[1] Github. Disponível em: <https://github.com/> Último acesso em: 25 de novembro de 2024.

[2] IDE Visual Studio Code. Disponível em: <https://code.visualstudio.com/> Último acesso em: 25 de novembro de 2024.

[3] Logisim. Disponível em: <http://www.cburch.com/logisim/> Último acesso em: 25 de novembro de 2024.

[4] Chipverify. Disponível em: <https://www.chipverify.com/tutorials/verilog> Último acesso em: 25 de novembro de 2024.