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

Introdução aos sistemas lógicos - TP 02

Circuitos Sequenciais

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

Florestal - MG
2025

Sumário

1. Introdução

Em um campeonato de robôs da UFV-Florestal, cada participante deve realizar corretamente o percurso em uma pista numérica, composta por uma sequência de números de 0 a 9. Nesse contexto, os alunos da disciplina de ISL ficaram responsáveis pelo desenvolvimento da lógica que controla as movimentações dos robôs. O objetivo principal do projeto foi determinado como a implementação de um algoritmo que tem como base os conceitos de máquina de estado, em que cada posição na pista é caracterizada como um estado distinto e sua transição deve ser realizada de acordo com os sinais de controle recebidos e os parâmetros estabelecidos.

No que se refere às condições da corrida, será atribuído estado de sucesso sempre que um robô concluir seu trajeto perfeitamente. Em contrapartida, caso seja executado algum erro de percurso, o LED de erro será aceso e o usuário terá a possibilidade de corrigir o trajeto para alcançar um estado de sucesso parcial desde que nenhum outro desvio aconteça. Se o erro se repetir, alcançará o estado de falha.

A elaboração do projeto consistiu na definição de um modelo de máquina de estado para criação do algoritmo, acompanhada de sua implementação com o uso da linguagem Verilog e a realização de testes através do modo de simulação. Em seguida, foram feitas algumas adaptações no código para sua devida execução em uma FPGA (Field Programmable Gate Array) e a validação dos resultados.

2. Organização

Na figura 1 é possível visualizar a organização do projeto. Foram criadas pastas para armazenar separadamente os arquivos de cada etapa do projeto.Na pasta **codigo/** está a implementação em verilog do módulo “maquina” junto de seu

módulo testbench e o arquivo de simulação de ondas (.vcd) gerado. Na pasta **diagrama/** está contida a imagem do diagrama de estados desenvolvido e o arquivo do mesmo diagrama gerado através do Jflap. Já a pasta **projeto_quartus/** contém os arquivos necessários para a execução em uma FPGA.

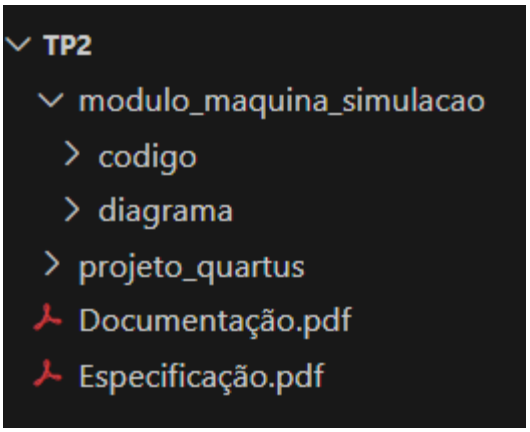


Figura 1 - Repositório do projeto.

3. Desenvolvimento

3.1 Diagrama de transição de estados

O primeiro passo executado foi a definição do modelo de máquina de estado a ser usado. Como parte da saída (display) é o estado atual da máquina, foi escolhido o modelo de Moore. Portanto, foi desenvolvido um diagrama de estados (figura 2) que reproduz os possíveis caminhos que podem ser tomados durante a execução do programa.

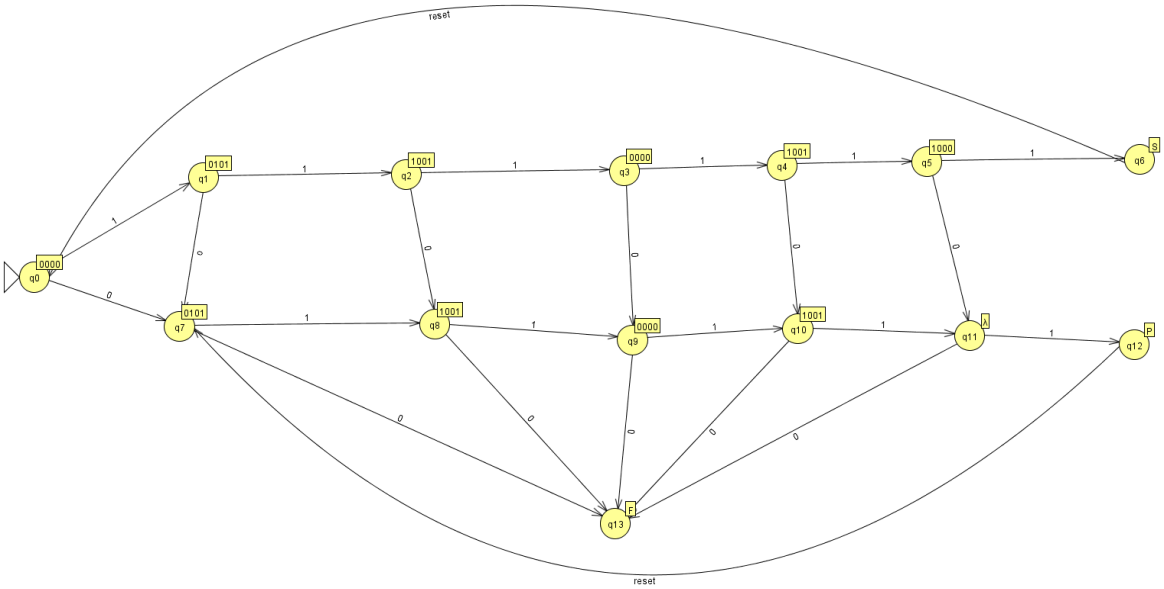


Figura 2 - Diagrama de estados

No que se refere à sequência correta de entradas, ficou estabelecido que o utilizador da máquina precisa informar os números 5,9,0,9,8 e 1 para alcançar um estado de sucesso.

Dessa forma, caso o usuário informe a sequência correta, a máquina irá sair do estado q0 e percorrer os estados q1,q2,q3,q4,q5 alcançando o estado q6, de sucesso total (S). Contudo, há a possibilidade de que apenas um erro aconteça em qualquer um dos estados q0, q1, q2, q3, q4, q5 o que leva aos estados de erro parcial q7, q8, q9, q10 ou q11, respectivamente, que culminará ao estado q6, sucesso parcial (P), ou ao estado q13, falha (F). Por exemplo, se ocorrer um erro no estado q0 a máquina irá para o estado q7, assim há somente duas possibilidades: acertar tudo a partir agora e ter um sucesso parcial ou cometer outro erro em qualquer um dos estados seguintes e falhar.

3.2 Implementação em verilog da máquina de estados finita

Uma vez construído o diagrama de estados apresentado no capítulo 3.1, iniciou-se o processo de sua implementação em um módulo de síntese em verilog para que fosse possível a criação de um módulo de simulação e visualização de sua forma de ondas. Nesse sentido, a lógica da máquina de estado estabelecida anteriormente foi implementada na linguagem de descrição de hardware Verilog.

Primeiramente, como mostra a figura 3, foram feitas as declarações iniciais das variáveis de entrada e saída, o assinalamento dos estados e as declarações das variáveis de estado. Além disso, foi estabelecido um bloco always sincronizado com uma entrada clock para retornar ao estado inicial, caso o reset seja ativado, ou, caso contrário, realizar a transição dos estados atribuindo o valor de “proximo_estado” para “estado”.

```
module maquina(
    //declaração das entradas e saídas
    input clk, reset, insere,
    input [4:1] numero,
    output reg LED =0,
    output reg A, B, C, D, E, F, G
);
//assinalamento de estados
    parameter inicial = 4'b0000;
    parameter cinco = 4'b0001;
    parameter nove = 4'b0010;
    parameter zero = 4'b0011;
    parameter nove_final= 4'b0100;
    parameter oito = 4'b0101;
    parameter um = 4'b1100;
    parameter falha = 4'b1111;

//variáveis de estado
    reg [3:0] estado;
    reg [3:0] proximo_estado;

//transição entre estados
    always @(posedge clk) begin
        if (reset == 1'b1)  begin
            estado <= inicial;
        end
        else begin
            estado <= proximo_estado;
        end
    end

```

Figura 3 - Declarações iniciais de variáveis e estados em verilog

Por conseguinte, foi implementado um bloco always sincronizado com a entrada clock que atualiza o “*proximo_estado*” se o **insere** estiver ativo ou reseta o *LED* para 0 se o **reset** estiver ativo , como mostra a figura 4. Sendo assim, o assinalamento do ”*proximo_estado*” ocorre de acordo com o estado atual, com a entrada “*numero*”, informado pelo usuário, e se o *LED* está ativo ou não para que o próximo estado seja designado. Dessa forma, surge a primeira diferença do diagrama para a implementação do módulo em verilog, pois, apesar de ter um funcionamento semelhante, a forma que isso ocorre é distinta já que no diagrama

existe um estado que indica uma falha existente e no código o *LED* é responsável por essa distinção.

Dessa forma, sempre que identificado uma entrada correta a máquina irá para o estado subsequente até que ocorra um erro ou então alcance o estado de sucesso (variável "*um*" no código, já que "1" é o último valor a ser inserido em qualquer tipo de sucesso). Por outro lado, se identificado uma entrada incorreta em qualquer estado a máquina permanece no estado atual e acende o *LED*. Assim, de forma similar ao diagrama, com uma falha há apenas duas opções: acertar todas as próximas entradas e alcançar o estado de sucesso parcial ou ir para o estado de falha se errar em qualquer um dos estados seguintes.

Em síntese, se está mostrando no display o número 5 mas o LED está apagado significa que você está no estado "q1" do diagrama de estados. Porém se o display indica 5 mas o LED está aceso, significa que você está no estado q7 do diagrama e qualquer erro leva para o estado de falha. Destarte, o LED é o que diferencia um estado com zero erros e um estado com 1 erro, o que é essencial para a distinção entre um resultado "S" ou "P". Uma vez que o LED está ativo, pode-se alcançar somente o sucesso parcial ou a falha.

```
always @(posedge clk) begin
    if (insere == 1'b0) begin
    case (estado)
        um:
            proximo_estado = um;
        falha:
            proximo_estado = falha;
        inicial:
            if (numero == 4'b0101) proximo_estado = cinco;
            else begin
                if (LED) proximo_estado = falha;
                else begin
                    proximo_estado = inicial;
                    LED = 1'b1;
                end
            end
        cinco:
            if (LED) begin
                if (numero == 4'b1001) proximo_estado = nove;
                else                   proximo_estado = falha;
            end
            else begin
                if (numero == 4'b1001) proximo_estado = nove;
                else begin
                    proximo_estado = cinco;
                    LED = 1;
                end
            end
```

Figura 4 - Bloco always que atualiza os estados e o LED de acordo com as entradas

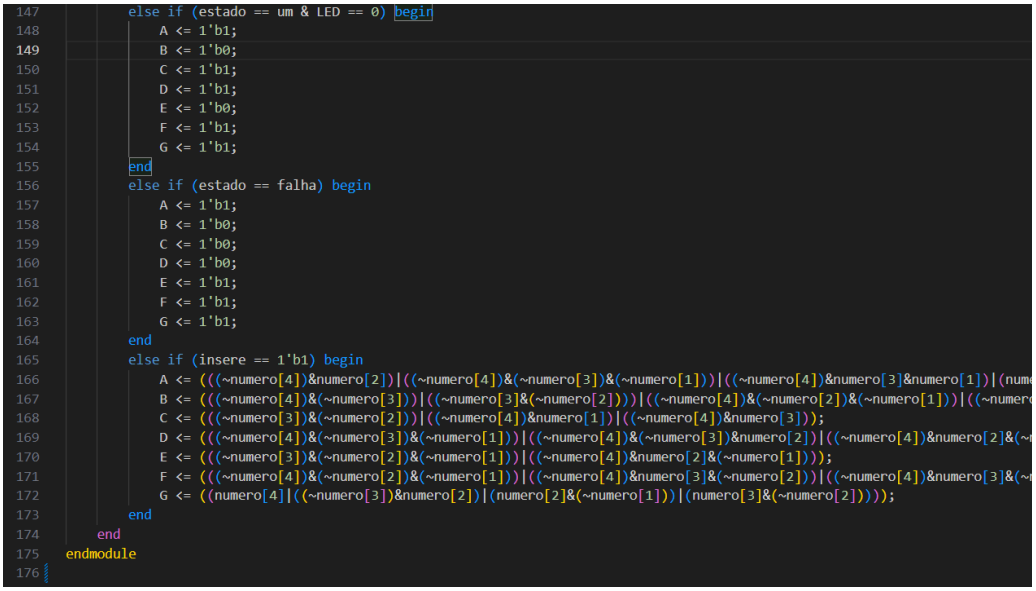
```
114
115             else begin
116           //reseta o LED quando a máquina reseta
117             if (reset == 1'b1) begin
118                     proximo_estado = inicial;
119                     LED = 0;
120               end
121             end
122       end
```

Figura 5 - Condição para levar a máquina ao estado inicial

Posteriormente, foi feita a configuração das variáveis A, B, C, D, E, F e G (cada segmento de um display), que indicam ou o último número informado ou S para estado de sucesso ou P para estado de sucesso parcial ou F para o estado “falha”. Para isso, foram utilizados alguns blocos condicionais sincronizados com o clock, que controlam o valor desses segmentos de forma que sempre mostre o último valor inserido, exceto se reset for acionado (mostra zero), se um os estados de sucesso forem alcançados (parcial (P) ou total (S)) ou se estiver no estado de falha (mostra (F)) como retratado pelas figuras 6 e 7. Para configuração dos segmentos do display em relação ao número inserido foi utilizada uma tabela verdade para obtenção de expressões booleanas a partir do mapa de Karnaugh, além disso foi definido que **entradas fora do intervalo de 0 a 9 são representadas como um “ - ”** e contam como uma falha (LED acende e apenas o segmento do meio é ativado).

```
126     always @(clk) begin
127           if (reset == 1'b1) begin
128             A <= 1'b1;
129             B <= 1'b1;
130             C <= 1'b1;
131             D <= 1'b1;
132             E <= 1'b1;
133             F <= 1'b1;
134             G <= 1'b0;
135
136           end
137         else if (estado == um & LED == 1) begin
138             A <= 1'b1;
139             B <= 1'b1;
140             C <= 1'b0;
141             D <= 1'b0;
142             E <= 1'b1;
143             F <= 1'b1;
144             G <= 1'b1;
145         end
```

Figura 6 - Blocos condicionais que podem controlar o display



```
147         else if (estado == um & LED == 0) begin
148             A <= 1'b1;
149             B <= 1'b0;
150             C <= 1'b1;
151             D <= 1'b1;
152             E <= 1'b0;
153             F <= 1'b1;
154             G <= 1'b1;
155         end
156         else if (estado == falha) begin
157             A <= 1'b1;
158             B <= 1'b0;
159             C <= 1'b0;
160             D <= 1'b0;
161             E <= 1'b1;
162             F <= 1'b1;
163             G <= 1'b1;
164         end
165         else if (insere == 1'b1) begin
166             A <= (((~numero[4])&numero[2])|((~numero[4])&(~numero[3])&(~numero[1]))|((~numero[4])&numero[3]&numero[1])|(num
167             B <= (((~numero[4])&(~numero[3]))|((~numero[3]&(~numero[2])))|((~numero[4])&(~numero[2])&(~numero[1]))|((~numer
168             C <= (((~numero[3])&(~numero[2]))|((~numero[4])&numero[1])|((~numero[4])&numero[3]));
169             D <= (((~numero[4])&(~numero[3])&(~numero[1]))|((~numero[4])&(~numero[3])&numero[2])|((~numero[4])&numero[2]&(~
170             E <= (((~numero[3])&(~numero[2])&(~numero[1]))|((~numero[4])&numero[2]&(~numero[1])));
171             F <= (((~numero[4])&(~numero[2])&(~numero[1]))|((~numero[4])&numero[3]&(~numero[2]))|((~numero[4])&numero[3]&(~
172             G <= ((numero[4]|((~numero[3])&numero[2])|(numero[2]&(~numero[1]))|(numero[3]&(~numero[2]))));
173         end
174     end
175 endmodule
176
```

Figura 7 - Blocos condicionais que podem controlar o display

Por fim, foi criado o módulo de simulação,o testbench, e a partir dele gerado o **.vcd** para análise da forma de ondas, que trouxe resultados satisfatórios e poderão ser observados mais adiante neste documento.

3.3 Implementação FPGA

Posteriormente, o código foi executado em uma FPGA (Field-Programmable Gate Arrays) e os testes realizados resultaram em alguns erros relacionados ao assinalamento de estados a partir do clock. Logo, foi necessária a alteração do bloco “always” de assinalamento de estados para que ele sincronizasse a atualização à variável de inserção do número, ao invés do clock, assim foi atribuído um “negedge insere” e a verificação da condição insere dentro do always passou a ser desnecessária e consequentemente removida (figura 8). Além disso, o controle de reset foi removido do bloco else na sequência do código.

```
always @(negedge insere) begin
    case (estado)
        um:
            proximo_estado = um;
        falha:
            proximo_estado = falha;
        inicial:
            if (numero == 4'b0101) proximo_estado = cinco;
            else begin
                if (LED) proximo_estado = falha;
```

Figura 8 - Alteração na forma com que o bloco always é atualizado

Contudo, essa atualização provocou outra alteração no funcionamento do circuito, uma vez que anteriormente o reset do LED dependia do clock, agora faz-se necessária uma atualização de “insere” para que essa função seja executada.

```
    default:
        proximo_estado = inicial;
endcase

if (reset == 1'b0) begin
        proximo_estado = inicial;
        LED = 0;
 end
```

Figura 9 - Verificação do reset dentro do bloco always, sincronizado com “insere”

Outra alteração realizada foi a inversão do reset,insere e das saídas do display de 7 segmentos, uma vez que o display da FPGA utilizada funciona/acende quando o nível lógico é baixo, ao contrário do que havia sido implementado anteriormente. Assim, com tais alterações propriamente estabelecidas, o algoritmo foi executado com êxito e realizou sua função adequadamente.

4. Compilação e Execução

4.1 Simulação

Para compilar os arquivos é necessário que você esteja na pasta onde o arquivo está localizado, .\modulo_maquina_simulacao\codigo, e utilize o comando “iverilog -o nomeParaOexe.vvp maquina.v” para compilar o módulo de síntese, e “iverilog -o nomeDoTBexe.vvp maquina_tb.v” para compilar o testbench. Para executar a simulação utilize o comando “vvp nomeDoTBexe.vvp” e aparecerá no terminal a saída desse módulo . Se você compilou e executou o módulo de teste (testbench) será gerado um arquivo.vcd “maquina.vcd”, com ele é possível abrir o gtkwave e visualizar a forma de ondas com o comando “gtkwave maquina.vcd”.

Esses comandos foram executados a partir do Sistema Operacional windows, utilizando o compilador icarus verilog e o programa gtkwave através da IDE Vs Code.

4.2 FPGA

Para a execução do programa na FPGA, é necessário utilizar um software que permite projetar FPGAs (o Quartus, por exemplo) e implementação de projetos nesse tipo de dispositivo.

Primeiramente, deve-se abrir o arquivo “FPGA_TP2.qpf” (responsável pelo código) e “FPGA_TP2.csv” (Responsável por mapear os pins da FPGA), ambos localizados dentro da pasta projeto_quartus/. Assim, já será possível realizar a conexão com o circuito integrado para a utilização do programa implementado e testar a passagens de entrada para alcançar algum dos estados referidos anteriormente. Caso o projeto não funcione corretamente, dentro da pasta “projeto_quartus/” encontra-se o arquivo top-level “maquina.v”. Assim permite-se que o usuário crie o projeto usando esse arquivo, compile e mapeie as entradas. Esse procedimento foi executado no sistema operacional Windows, a partir do software Intel Quartus Prime**.**

Por fim, quanto à utilização do programa na FPGA, é válido evidenciar novamente que para fazer uso da função *reset* é necessário atualizar seu sinal (switch ou botão) juntamente com *insere*, por motivos já explicados anteriormente.

5. Resultados

5.1 Simulação

Destarte, uma vez que os módulos sejam compilados é possível executá-los. No entanto, essa etapa de testes está limitada ao modo de simulação, uma vez que o procedimento de síntese será executado posteriormente. Sendo assim , ao executar o testbench do módulo será gerada uma saída via terminal, como na imagem 10, e um arquivo .vcd para visualização da forma de ondas do circuito, como na imagem 11. Ao executar o módulo da máquina de estado, serão exibidos estímulos gerados pelo testbench e suas respectivas saídas. Assim, “numero” se refere aos quatro bits definidos como as entradas do testbench, tais dígitos são

imediatamente processados para exibição no display, em que os segmentos representados pelas saídas A,B,C,D,E,F,G são acesos de modo a reproduzir o número decimal informado ou os estados de sucesso, sucesso parcial e falha, com a observação de que o intervalo 10 a 15 é indicado pelo caractere especial “-” e contabiliza como um erro (apenas o segmento G acende).

Paralelamente, verifica-se se a entrada condiz com a sequência pré-definida **(5,9,0,9,8,1)**, e em caso de inconsistência, é atribuído ao LED o valor 1, como pode ser observado nos casos de sucesso parcial e falha representados na figura 10. Além disso, vale ressaltar que o número que finaliza a execução da máquina nunca é exibido pelo display, haja vista que é mostrado o resultado “S”,”P” ou “F”, e assim que uma dessas situações é alcançada, também existe a possibilidade de acionar o reset (atribuindo o valor 1) para que a máquina retorne ao seu estado inicial.

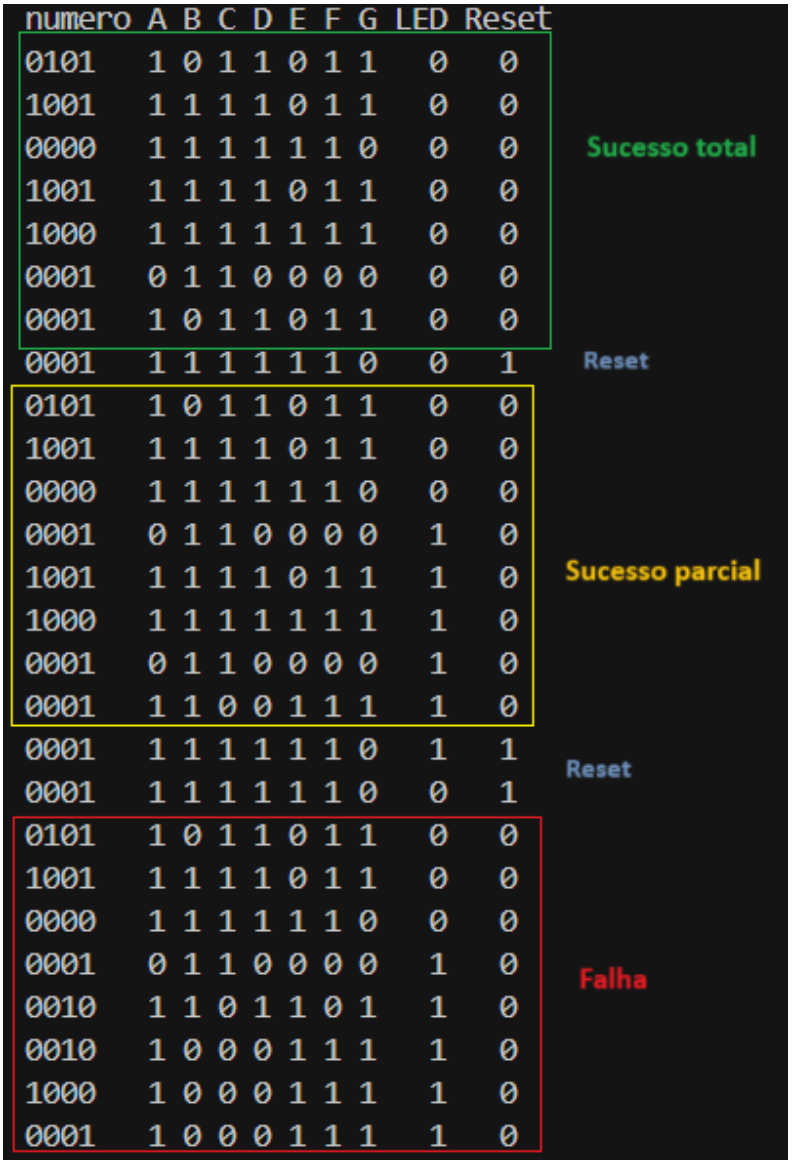


Figura 10 - Saída pelo terminal do testbench do módulo maquina

De forma similar, esse mesmo resultado da simulação pode ser visualizado na forma de ondas pelo gtkwave. No entanto, na forma de ondas os estados estão indicados na variável “estado” e parte do estado “0” (inicial) e alcançam sucesso no estado “c”, tanto para sucessos parciais (indicado pelo estado em “c” e o LED ativo) quanto para sucesso total (indicado pelo estado em “c” e o LED inativo), e a falha está representado pelo F.

Além disso, quando é desejado reiniciar a máquina o reset torna-se ativo e o estado volta para “0”, de forma que novos números podem ser inseridos e sincronizados pela borda de subida do insere, de maneira que na próxima borda de subida do clock o “proximo_estado” será atualizado de acordo com a entrada a corretude da entrada “numero” e na posterior borda de subida do clock o “estado” será atualizado de acordo “com o proximo_estado” e, por fim, no próximo clock o LED será 1 se o “numero” inserido for um erro ou continuará 0 se estiver certo.

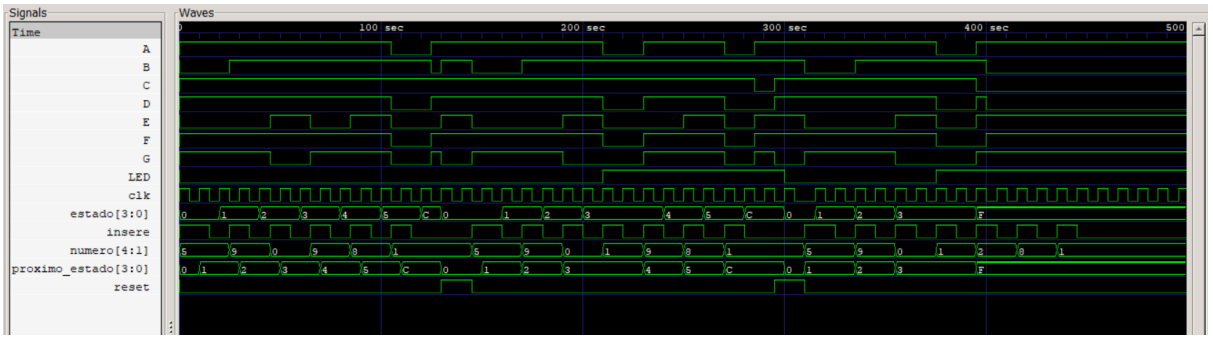


Figura 11 - Forma de ondas do testbench

5.2 FPGA

Por fim, o programa foi executado com êxito na FPGA e houveram resultados satisfatórios. Isto é, quando foi feita uma combinação de entradas que representam um número de quatro bits (0 a 15) e pressionado o botão relacionado à variável “insert”, foi realizado o processamento da entrada para definir o avanço para o estado seguinte ou a manutenção do estado, o acionamento do LED de erro ou não. Ademais, uma vez que o botão de inserção foi acionado, o display atualizou automaticamente para mostrar o número inserido se a entrada estivesse de acordo

com o intervalo, caso contrário, indicou apenas um traço e contabilizou um erro. O display também atualizou sempre que a máquina atingiu algum dos estados finais, exibindo “S”,”P” ou “F” a depender do resultado alcançado. Além disso, no caso de acionamento do reset o display mostrou o dígito zero, indicando que a máquina se encontrava em seu estado inicial.

Todas essas demonstrações são retratadas pelas figuras 12-17, referentes aos testes realizados, e uma observação mais detalhada dos resultados obtidos pode ser visualizada através do vídeo enviado.

Relembrando que a pista é composta pela sequência 590981.

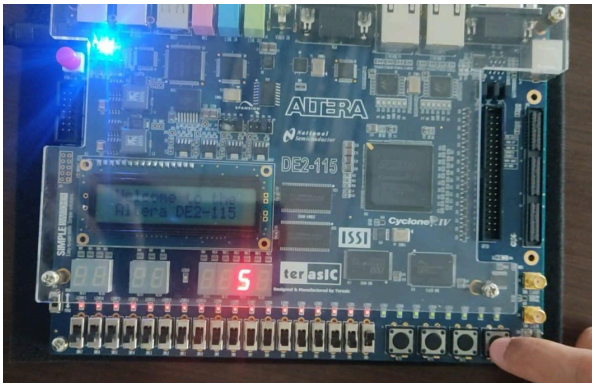


Figura 12 - Sucesso total

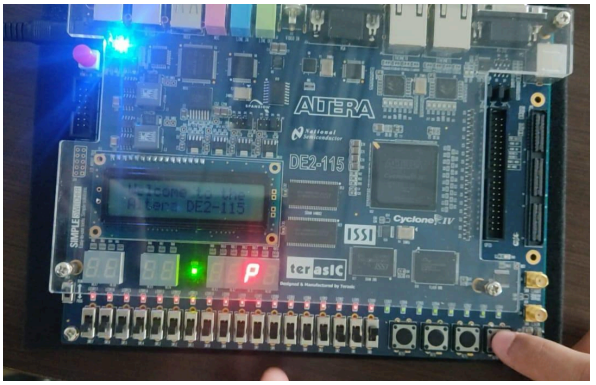


Figura 13 - Sucesso parcial

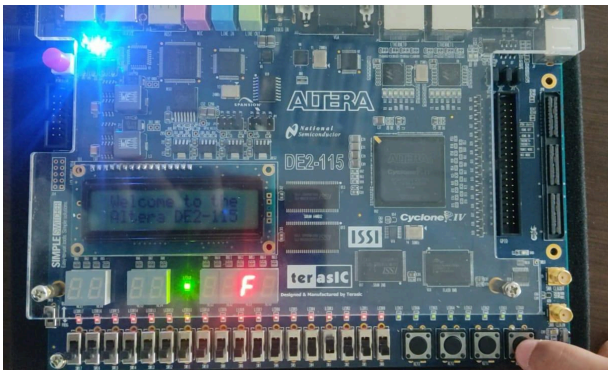


Figura 14 - Falha

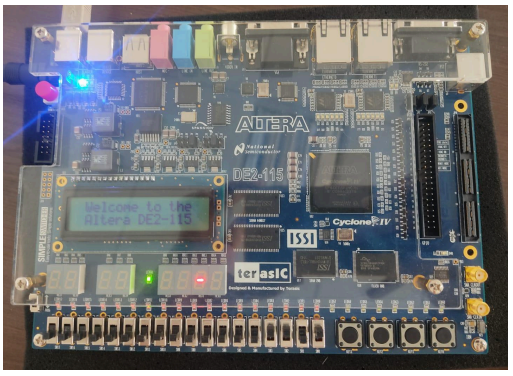


Figura 15 - Número fora do intervalo e erro contabilizado

.

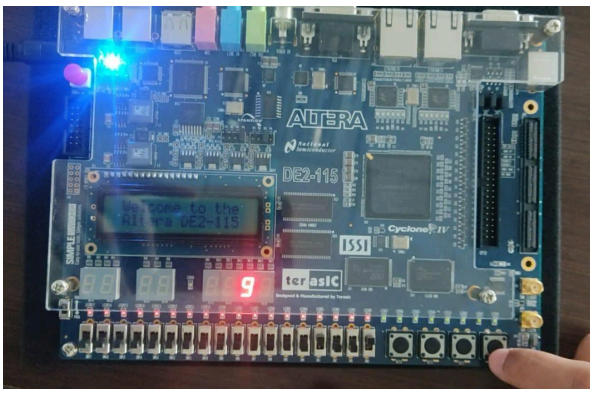


Figura 16 - Exibição do número inserido

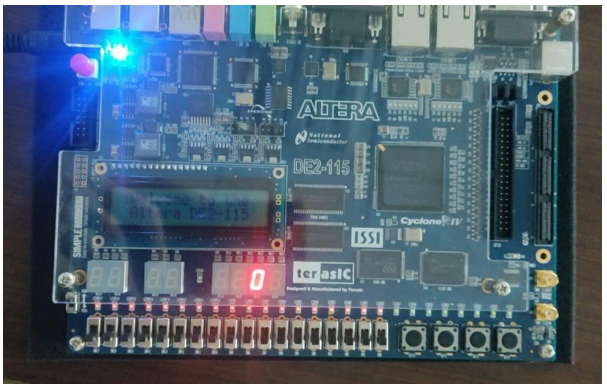


Figura 17 - Estado inicial após reset

6. Vídeo explicativo e testes na FPGA

A seguir se encontra o link de acesso ao vídeo que contém uma explicação do código implementado, juntamente com a gravação dos testes realizados na FPGA.

Vídeo explicativo + testes:
https://drive.google.com/file/d/1wL7Bb418ChpSYbAWuXQIirWYVYiGweAg/view

7. Conclusão

As etapas de desenvolvimento deste projeto tiveram como foco a implementação de um sistema capaz de controlar os avanços de um robô em uma pista caso informada uma combinação correta de números, sendo capaz de identificar entradas incorretas e informar o resultado alcançado. Durante esse processo, foram utilizados conceitos de circuitos sequenciais e máquinas de estado finito para desenvolver um código na linguagem de descrição de hardware Verilog que atendesse a todas essas condições.

Diante disso, os resultados obtidos foram satisfatórios e coerentes com a especificação fornecida. O sistema conseguiu realizar as transições de estados corretamente, identificar valores inseridos no momento errado a partir do acionamento de um led e representar no display tanto os números inseridos quanto os resultados finais, como observado no detalhamento a respeito da implementação.

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.

8. Referências

[1] Github. Disponível em: <https://github.com/> Último acesso em: 17 de janeiro de 2025;

[2] IDE Visual Studio Code. Disponível em: <https://code.visualstudio.com/> Último acesso em: 17 de janeiro de 2025;

[3] Notas de aula: “Máquinas de estado finito (Cap. 07 - Katz)” Último acesso em: 17 de janeiro de 2025;

[4] Quartus. Disponível em: <https://www.intel.com.br/content/www/br/pt/products/details/fpga/development-tools/quartus-prime.html> Último acesso em: 17 de janeiro de 2025;