



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

Trabalho 2 - ISL

Circuitos Sequenciais

Pedro Henrique Dias Quintão [5916]

Tariky Rodrigues [5758]

Florestal - MG

2025

Sumário

1. Introdução	3
2. Desenvolvimento	4
2.1 Montagem da Máquina de estados	4
2.2 Código em Verilog	
2.3 Fpga	
2.4 GtkWave	4
3. Conclusão	4
4. Referências	5

1. Introdução

Este projeto, desenvolvido para a competição de robôs na UFV-Florestal, tem como objetivo criar uma Máquina de Estados Finita (FSM) em Verilog para controlar o trajeto de um robô em uma pista numérica personalizada. A FSM deve verificar o trajeto correto, sinalizar erros e indicar estados de Sucesso Total, Sucesso Parcial ou Falha, utilizando um display de sete segmentos e LEDs.

O desenvolvimento inclui a modelagem do diagrama de estados (com JFlap ou similar), implementação e simulação em Verilog, e integração no kit FPGA DE2-115. Decisões sobre entradas inválidas e outros ajustes serão documentados.

Este trabalho integra teoria e prática em sistemas lógicos, abordando conceitos de FSM, programação em Verilog e uso de hardware reconfigurável, consolidando o aprendizado de ISL

2. Desenvolvimento.

2.1 Montagem da Máquina de estados

Primeiramente focamos em montar a máquina de estados, para que possamos ter mais tranquilidade nos próximos passos. Nós montamos a máquina através do site Jflap que é especializado para esse tipo de montagem, a nossa máquina de estados é de moore ou seja a saída é determinada pelo próprio estado, a representação dela se encontra na figura 1.

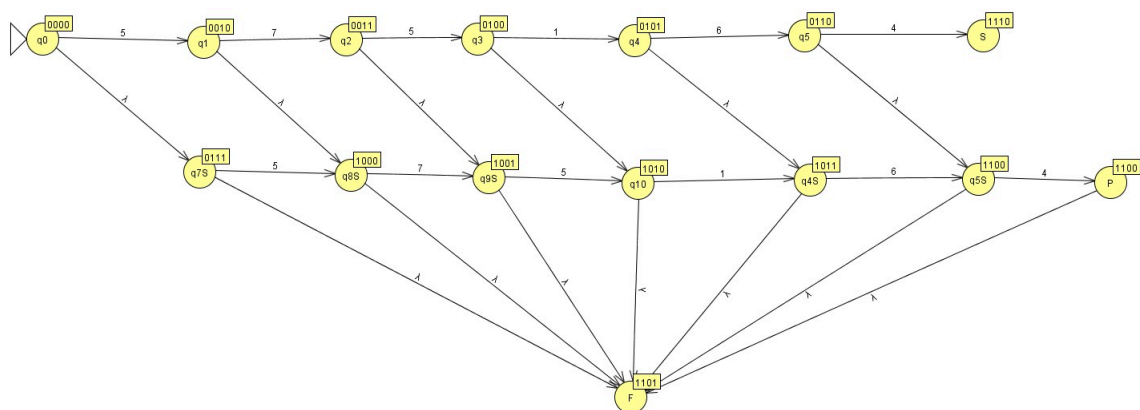


Figura 1: Máquina de estados no Jflap

2.2 Código em Verilog

Logo em seguida fizemos o código em verilog e o testbench para testar a ideia da nossa máquina de estados antes de usá-la na fpga, O código foi montado no vscode e a partir dos valores colocados no testbench ele indica se o robô acertou completamente, errou parcialmente ou errou completamente. O código final foi enviado junto desta documentação.

```
parameter q0 = 4'b0000, q1 = 4'b0001, q2 = 4'b0010, q3 = 4'b0011,
q4 = 4'b0100, q5 = 4'b0101, q0S = 4'b0110, q1S = 4'b0111,
q2S = 4'b1000, q3S = 4'b1001, q4S = 4'b1010, q5S = 4'b1011,
sucessoparcial = 4'b1100, falha = 4'b1101, sucessolindo = 4'b1110;

parameter [3:0] N0 = 4'd5, N1 = 4'd7, N2 = 4'd5, N3 = 4'd1, N4 = 4'd6,
N5 = 4'd4;

always @(*) begin

    case (estado)
        sucessolindo: display_num = 7'b0010010;
        sucessoparcial: display_num = 7'b0001100;
        falha: display_num = 7'b0001110;
        default: display_num = 7'b1111111;
    endcase

    if (estado != sucessolindo && estado != sucessoparcial && estado !=
falha) begin
        case (numero)
            4'b0000: display_num = 7'b1000000;
            4'b0001: display_num = 7'b1111001;
            4'b0010: display_num = 7'b0100100;
            4'b0011: display_num = 7'b0110000;
            4'b0100: display_num = 7'b0011001;
            4'b0101: display_num = 7'b0010010;
            4'b0110: display_num = 7'b0000010;
            4'b0111: display_num = 7'b1111000;
            4'b1000: display_num = 7'b0000000;
            4'b1001: display_num = 7'b0010000;
            default: display_num = 7'b1111111;
        endcase
    end
end
```

Figura 2: onde definimos os estados e um case para mostrar no display

A lógica progride entre os estados (q0, q1, q2, etc.), verificando condições sobre o valor de número. Dependendo das condições, o sistema transita para um próximo estado de sucesso, retorna ao inicial ou permanece no atual, indicando o progresso do processo. O uso de error_led também é essencial para indicar se uma condição incorreta foi encontrada. Essa parte do código é de extrema importância, pois garante o controle do fluxo lógico da máquina de estados, validando os dados e determinando o comportamento do sistema de forma organizada e previsível.

```
always @(posedge clock or posedge reset) begin
    if (reset) begin
        state <= q0;
    end else begin
        state <= next_state;
    end
end

always @(posedge clock or negedge insere) begin
    if (insere == 0) begin
        case (state)
            q0: begin
                if ((number == N0)&&(number<=9)) begin
                    next_state = q1;
                    error_led = 1;
                end else if (number != N0) begin
                    next_state = q0S;
                    error_led = 0;
                end else begin
                    next_state = q0;
                end
            end
            q1: begin
                if ((number == N1)&&(number<=9)) begin
                    next_state = q2;
                    error_led = 1;
                end else if (number != N1) begin
                    next_state = q1S;
                    error_led = 0;
                end else begin
                    next_state = q1;
                end
            end
            q2: begin
                if ((number == N2)&&(number<=9)) begin
                    next_state = q3;
                    error_led = 1;
                end else if (number != N2) begin
                    next_state = q2S;
                    error_led = 0;
                end else begin
                    next_state = q2;
                end
            end
        end
    end
end
```

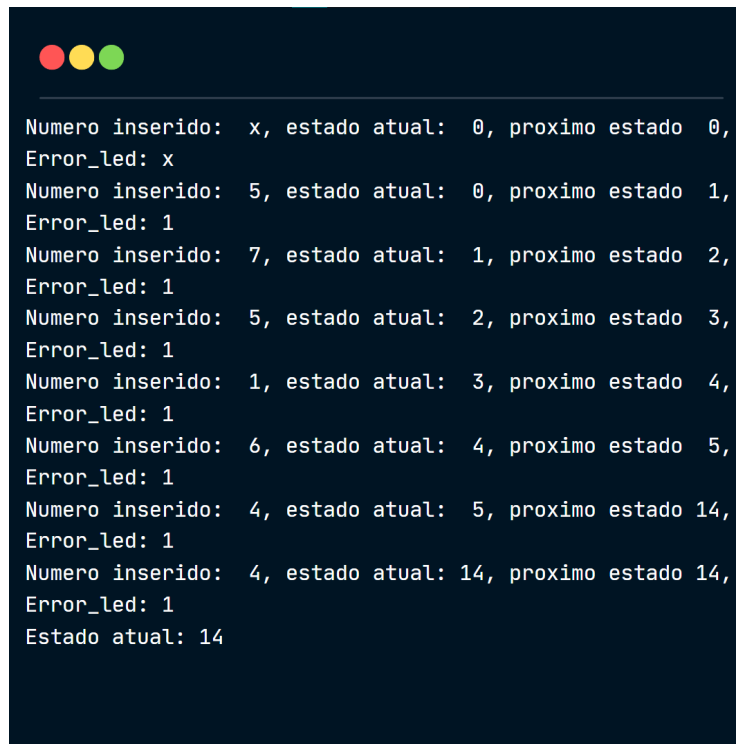
Figura 3: Amostra dos estados.

Agora as figuras abaixo mostra os resultados obtidos no testbench.

Figura 4: ESTADO SUCESSO TOTAL = 14;

Figura 5: ESTADO ESTADO PARCIAL = 12;

Figura 6: ESTADO FALHA = 13;



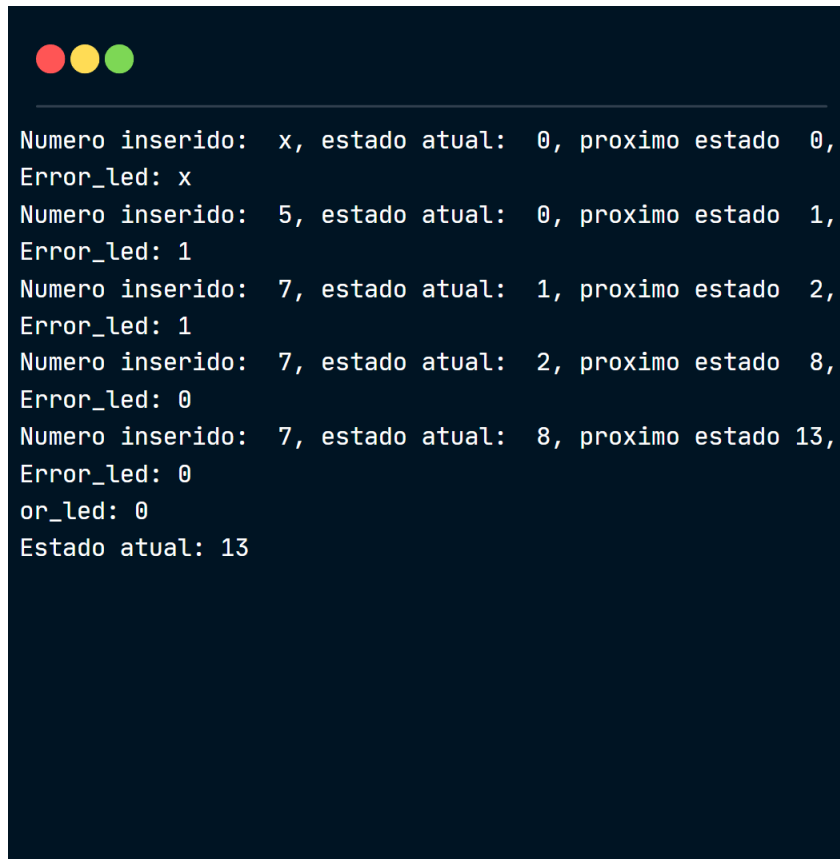
```
Numero inserido: x, estado atual: 0, proximo estado 0,
Error_led: x
Numero inserido: 5, estado atual: 0, proximo estado 1,
Error_led: 1
Numero inserido: 7, estado atual: 1, proximo estado 2,
Error_led: 1
Numero inserido: 5, estado atual: 2, proximo estado 3,
Error_led: 1
Numero inserido: 1, estado atual: 3, proximo estado 4,
Error_led: 1
Numero inserido: 6, estado atual: 4, proximo estado 5,
Error_led: 1
Numero inserido: 4, estado atual: 5, proximo estado 14,
Error_led: 1
Numero inserido: 4, estado atual: 14, proximo estado 14,
Error_led: 1
Estado atual: 14
```

Figura 3: sequência correta vscode



```
Numero inserido: x, estado atual: 0, proximo estado 0,
Error_led: x
Numero inserido: 5, estado atual: 0, proximo estado 1,
Error_led: 1
Numero inserido: 7, estado atual: 1, proximo estado 2,
Error_led: 1
Numero inserido: 7, estado atual: 2, proximo estado 8,
Error_led: 0
Numero inserido: 5, estado atual: 8, proximo estado 9,
Error_led: 0
Numero inserido: 1, estado atual: 9, proximo estado 10,
Error_led: 0
Numero inserido: 6, estado atual: 10, proximo estado 11,
Error_led: 0
Numero inserido: 4, estado atual: 11, proximo estado 12,
Error_led: 0
Numero inserido: 4, estado atual: 12, proximo estado 12,
Error_led: 0
Estado atual: 12
```

Figura 4: Sequência parcial obtida.



```
Numero inserido: x, estado atual: 0, proximo estado 0,
Error_led: x
Numero inserido: 5, estado atual: 0, proximo estado 1,
Error_led: 1
Numero inserido: 7, estado atual: 1, proximo estado 2,
Error_led: 1
Numero inserido: 7, estado atual: 2, proximo estado 8,
Error_led: 0
Numero inserido: 7, estado atual: 8, proximo estado 13,
Error_led: 0
or_led: 0
Estado atual: 13
```

Figura 5: Sequência totalmente errada

2.3 Fpga

Um FPGA é um dispositivo semicondutor que é largamente utilizado para o processamento de informações digitais

Na Fpga é onde acontece a parte mais importante desse trabalho, onde usamos o código feito no Vscode para testar com uma entrada e um clk da própria placa para mostrar o funcionamento e a execução da nossa máquina de estados.

Para o funcionamento da Fpga precisamos configurar corretamente os pinos da placa, já que sem configurá-los a placa não iria funcionar. Após configurá-los podemos testar o código junto da placa e ver os resultados nos displays e leds.

O código utilizado na fpga está presente nas figuras 5 e 6 desta documentação.

2.4 GtkWave

O gtkwave é um visualizador de formas de onda que permite visualizar as formas de onda geradas pelo circuito, neste trabalho usamos ele para confirmar o funcionamento do circuito da fpga, a foto das ondas do gtkwave está presente na figura 7.

Os números inseridos (number) mudam conforme definido no testbench.

O display (display_num) reage aos números e transições de estados.

A máquina de estados está processando corretamente as entradas, mas o LED de erro será ativado se um número incorreto for detectado.

Display de número (display_num[6:0])

Este sinal representa o valor que deve aparecer no display numérico, possivelmente usando um padrão de 7 segmentos. Durante a inserção dos números:

Cada número correto faz com que display_num mude para exibir o número atual.

Exemplo:

Ao inserir o número 5, o display_num assume um padrão correspondente ao número 5.

Com o número 7, ele muda para exibir 7, e assim por diante.

A transição entre os números ocorre sincronizada com as mudanças de number.

6. LED de erro (error_led)

Durante a inserção da sequência correta 5, 7, 5, 1, 6, 4, o error_led permanece desligado(0)

Isso indica que a máquina reconheceu cada número como válido e não encontrou erros na sequência.

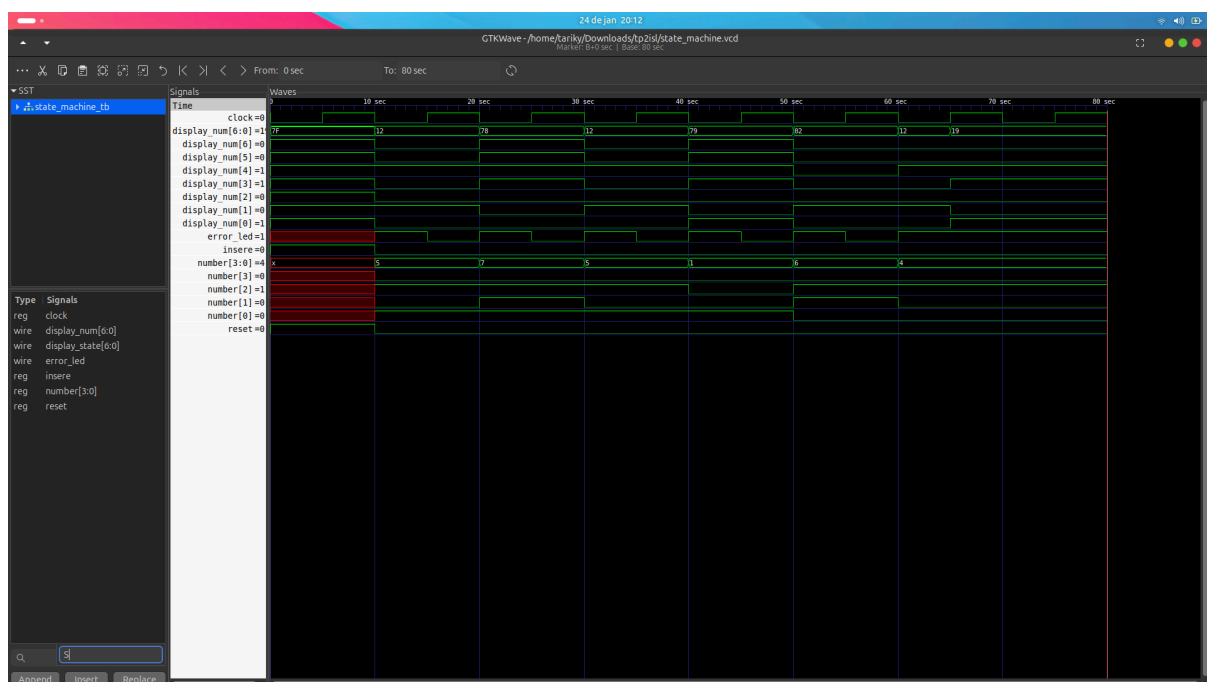


Figura 6 : Imagem GTKWave

O sistema foi projetado para controlar um robô em um ambiente com uma pista numérica. A máquina de estados lida corretamente com as transições entre os estados de sucesso, sucesso parcial e falha, e fornece feedback ao usuário por meio de displays e LEDs. A implementação foi testada e os resultados mostraram que o robô segue a sequência corretamente ou indica erro de forma apropriada.

4. Referências

- [1] Github. Disponível em: <<https://github.com/>> Último acesso em: 06 de dezembro de 2021.
- [2] introdução a verilog. Disponível em :
<<https://www.ic.unicamp.br/~rodolfo/Cursos/verilog/NocoasBasicas/>>
- [3] Video FPGA
<https://www.youtube.com/watch?v=tF3oPCiKIJ8>