



Universidade Federal de São Carlos - UFSCar

Lorhan Sohaky de Oliveira Duda Kondo 740951

Experimento 05 - Implementação de um círcuito sequencial utilizando Verilog

São Carlos - SP

2017

Universidade Federal de São Carlos - UFSCar

Lorhan Sohaky de Oliveira Duda Kondo 740951

Experimento 05 - Implementação de um circuito sequencial utilizando Verilog

Orientador: Fredy João Valente

Universidade Federal de São Carlos - UFSCar

Departamento de Computação

Ciência da Computação

Laboratório de Circuitos Digitais

São Carlos - SP

2017

Listas de ilustrações

Figura 1 – Ilustração da máquina de estado do problema da garagem.	9
Figura 2 – Resultado da compilação do código da máquina de estado do problema da garagem.	12
Figura 3 – <i>Test bench Wave</i> do código da máquina.	14
Figura 4 – <i>Test bench Transcript</i> do código da máquina.	14
Figura 5 – Teste do circuito rodando na placa.	15
Figura 6 – Máquina de estado do circuito adicional.	17
Figura 7 – Compilação da máquina de estado do circuito adicional.	18
Figura 8 – <i>Test bench Wave</i> da máquina de estado adicional.	19
Figura 9 – <i>Test bench Transcript</i> da máquina de estado adicional.	20

Lista de tabelas

Lista de quadros

Quadro 1 – Lista das entradas da máquina de estado do problema da garagem. . .	8
Quadro 2 – Significados dos estados relacionando com os estados reais do problema proposto.	9

Lista de abreviaturas e siglas

FPGA *Field Programmable Gate Array* - Arranjo de Portas Programáveis em Campo

LED *Light Emitting Diode* - Diodo emissor de luz

Sumário

1	RESUMO	7
2	DESCRIÇÃO DA EXECUÇÃO DO EXPERIMENTO	8
2.1	Passo 1 – Desenhar a máquina de estado	8
2.2	Passo 2 - Escrever um código Verilog para a máquina de estado	9
3	AVALIAÇÃO DOS RESULTADOS DO EXPERIMENTO	14
4	ANÁLISE CRÍTICA E DISCUSSÃO	16
5	OUTRAS INFORMAÇÕES	17

1 Resumo

A ideia deste experimento é implementar uma máquina de estado utilizando, a linguagem de descrição de *hardware*, Verilog. A máquina tenta representar uma situação do mundo real de um portão de garagem, conforme o cenário proposto abaixo:

Considere o cenário de um controle para um portão de garagem. Em um estado inicial, o portão está fechado. Caso um acionador externo seja selecionado, o portão abre. Caso o portão esteja aberto e o acionador externo seja selecionado, o portão fecha. O portão nunca abre e fecha ao mesmo tempo. O trilho no qual o portão se desloca é equipado com dois sensores que indicam quando o portão está completamente aberto e quando está completamente fechado. O motor não deve tentar abrir o portão quando esse estiver aberto e nem deve fechá-lo quando este já estiver fechado.

Para maior segurança dos usuários, o motor está equipado com um aviso luminoso que deve ser acesso quando o portão se desloca.

Deve-se assumir que não é possível parar o portão enquanto ele estiver abrindo ou fechando, mas é possível que o usuário aperte o acionador externo enquanto o portão estiver se deslocando. Nesse caso, se o portão estiver abrindo, ele deve passar a fechar e vice-versa.

Para solucionar tal problema e facilitar sua resolução, dividiu-se o processo em três passos:

1. Desenhar a máquina de estado para o cenário em questão;
2. Escrever um código Verilog para a máquina de estado no passo anterior;
3. Executar o código na *Field Programmable Gate Array* - Arranjo de Portas Programáveis em Campo (FPGA) e simulação.

Além disso, escolheu-se uma máquina de estado qualquer para estudar como implementá-la em Verilog.

2 Descrição da execução do experimento

2.1 Passo 1 – Desenhar a máquina de estado

Com o problema em questão, tem-se em mente que trata-se de um portão de garagem que move-se horizontalmente, ou seja, da esquerda para direita e vice e versa, deste modo o portão abriria para a esquerda e fecharia para a direita. Assim, para a elaboração da máquina¹, considerou-se as entradas conforme o [Quadro 1](#).

Quadro 1 – Lista das entradas da máquina de estado do problema da garagem.

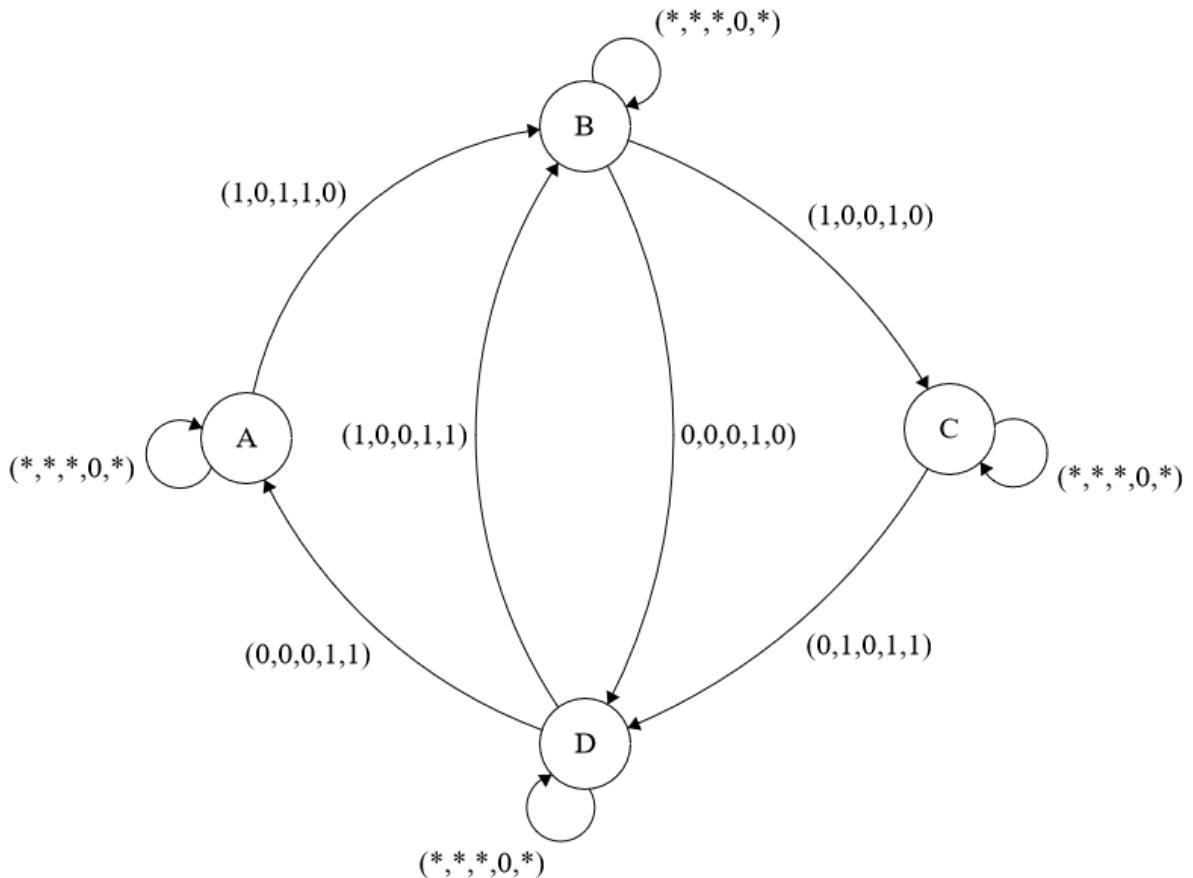
Entrada	Valor lógico 0	Valor lógico 1
Botao (b)	Abrir	Fechar
Aberto (a)	Não aberto	Totalmente aberto
Fechado (f)	Não fechado	Totalmente fechado
Motor (m)	Motor desligado	Motor ligado
Sentido (s)	Esquerda	Direita

Fonte: Próprio Autor

Com as entradas descritas no [Quadro 1](#) elaborou-se a máquina conforme a [Figura 1](#).

¹ Preferiu-se a utilização de uma máquina de Moore por haver conhecimento prévio deste tipo de máquina.

Figura 1 – Ilustração da máquina de estado do problema da garagem.



Nota: A entrada está no formato (b, a, f, m, s). Os significados dos estados estão no [Quadro 2](#).

Quadro 2 – Significados dos estados relacionando com os estados reais do problema proposto.

Estado	Significado
A	Totalmente fechado
B	Abrindo
C	Totalmente aberto
D	Fechando

Fonte: Próprio Autor

2.2 Passo 2 - Escrever um código Verilog para a máquina de estado

Para a realização deste passo, foram utilizados o programa Quartus 13.0 SP 1 e a placa FPGA Cyclone II - EP2C20F484C7.

Com a máquina de estado pronto, criou-se o código em Verilog, que no estado "totalmente aberto" apresenta A no *display*, no estado "totalmente fechado" apresenta

F no *display*, no estado "abrindo" acende um *Light Emitting Diode* - Diodo emissor de luz (LED) verde e no estado "fechando" acende um LED vermelho. Nos estados "abrindo" e "fechando" o *display* apresenta 0. O código da máquina de estado encontra-se no [Código 2.1](#) e o resultado da compilação na [Figura 2](#).

Código 2.1 – Código da máquina de estado do problema da garagem.

```

module inicial ( botao, aberto, fechado, motor, sentido, ledVerde,
    ledVermelho, display, clock );
    input botao, aberto, fechado, motor, sentido, clock;
    output ledVerde, ledVermelho;
    output [6:0] display;

    reg [1:0] estado;
    reg [4:0] entrada;

    reg [6:0] tmpDisplay;
    reg tmpLedVerde, tmpLedVermelho;

    parameter Fechado = 2'b00, Abrindo = 2'b01, Aberto = 2'b10, Fechando =
        2'b11;

    initial estado = Fechado;

    always @ (posedge clock) begin
        entrada[4] = botao;
        entrada[3] = aberto;
        entrada[2] = fechado;
        entrada[1] = motor;
        entrada[0] = sentido;

        case( estado )
            Fechado: begin
                tmpDisplay = 7'b0001110;
                tmpLedVerde = 0;
                tmpLedVermelho = 0;

                if( entrada == 5'b10110 ) // botao = 1 & aberto = 0
                    & fechado = 1 & motor = 1 & sentido = 0
                    estado = Abrindo;
            end

            Abrindo: begin
                tmpDisplay = 7'b1000000;
                tmpLedVerde = 1;
                tmpLedVermelho = 0;

                if( entrada == 5'b10010 ) // botao = 1 & aberto = 0
                    & fechado = 0 & motor = 0 & sentido = 1
                    estado = Fechando;
            end
        endcase
    end
endmodule

```

```

    & fechado = 0 && motor = 1 & sentido = 0
        estado = Aberto;
        if( entrada == 5'b00010 ) // botao = 0 & aberto = 0
    & fechado = 0 & motor = 1 & sentido == 0
        estado = Fechando;
    end

    Aberto: begin
        tmpDisplay = 7'b0001000;
        tmpLedVerde = 0;
        tmpLedVermelho = 0;

        if( entrada == 5'b01011 ) // botao = 0 & aberto = 1
    & fechado = 0 & motor = 1 & sentido = 1
        estado = Fechando;
    end

    Fechando: begin
        tmpDisplay = 7'b1000000;
        tmpLedVerde = 0;
        tmpLedVermelho = 1;

        if( entrada == 5'b10011 ) // botao = 1 & aberto = 0
    & fechado = 0 & motor = 1 & sentido = 1
            estado = Abrindo;
        if( entrada == 5'b00011 ) // botao = 0 & aberto = 0
    & fechado = 0 & motor = 1 & sentido = 1
            estado = Fechado;
    end

    default: estado = Fechado;

endcase
end

assign display= tmpDisplay;
assign ledVerde = tmpLedVerde;
assign ledVermelho = tmpLedVermelho;

endmodule

module maquina( SW, LEDG, LEDR, HEXO, CLK );
    input [4:0] SW;
    input CLK;
    output [0:0] LEDG, LEDR;
    output [6:0] HEXO;

```

```

inicial a( SW[4] , SW[3] , SW[2] , SW[1] , SW[0] , LEDG[0] , LEDR[0] , HEXO ,
CLK);
endmodule

```

Nota: Para as entradas não mapeadas, a máquina mantém-se no estado atual.

Figura 2 – Resultado da compilação do código da máquina de estado do problema da garagem.

Flow Summary	
Flow Status	Successful - Thu Dec 07 16:56:53 2017
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	maquina
Top-level Entity Name	maquina
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	20 / 18,752 (< 1 %)
Total combinational functions	20 / 18,752 (< 1 %)
Dedicated logic registers	10 / 18,752 (< 1 %)
Total registers	10
Total pins	15 / 315 (5 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

Para a realização dos testes, criou-se um *test bench* conforme o [Código 2.2](#). Além disto, fez-se o *deploy* na FPGA².

Código 2.2 – Código de *test bench* da máquina do problema da garagem.

```

module maquina_TB;
    integer a;

    wire ledVerde, ledVermelho;
    wire [6:0] HEX;
    reg [4:0] val;
    reg clock;

    inicial i( a[4] , a[3] , a[2] , a[1] , a[0] , ledVerde , ledVermelho , HEX ,
clock);

    //Alterando clock
    always begin
        clock <= 0;
        #25;
        clock <= 1;
    end

```

² Os resultados dos testes e simulações encontram-se na Avaliação dos resultados.

```

#25;
end

initial begin
$display("\tSistema Porta\n");
$display(" Estado | Entrada | LG | LR | HEX");
$display("-----");

//Estado Fechado
#50
a=0;#100 // a = 00000
$write( " Fechado | %x%x%x%x | %x | %x | ", a[4], a[3], a[2], a[1], a[0], ledVerde, ledVermelho);
if( HEX == 7'b0001110 ) $display( " F" ); if( HEX == 7'b1000000 )
$display (" 0" ); if( HEX == 7'b0001000 ) $display( " A" );

//Estado Abrindo
a=22;#100 // a = 10110
$write( " Abrindo | %x%x%x%x | %x | %x | ", a[4], a[3], a[2], a[1], a[0], ledVerde, ledVermelho);
if( HEX == 7'b0001110 ) $display( " F" ); if( HEX == 7'b1000000 )
$display (" 0" ); if( HEX == 7'b0001000 ) $display( " A" );

//Estado Aberto
a=18;#100 // a = 10010
$write( " Aberto | %x%x%x%x | %x | %x | ", a[4], a[3], a[2], a[1], a[0], ledVerde, ledVermelho);
if( HEX == 7'b0001110 ) $display( " F" ); if( HEX == 7'b1000000 )
$display (" 0" ); if( HEX == 7'b0001000 ) $display( " A" );

//Estado Fechando
a=11;#100 // a = 01011
$write( " Fechando | %x%x%x%x | %x | %x | ", a[4], a[3], a[2], a[1], a[0], ledVerde, ledVermelho);
if( HEX == 7'b0001110 ) $display( " F" ); if( HEX == 7'b1000000 )
$display (" 0" ); if( HEX == 7'b0001000 ) $display( " A" );

//Estado Fechado
#50
a=3;#100 // a = 00011
$write( " Fechado | %x%x%x%x | %x | %x | ", a[4], a[3], a[2], a[1], a[0], ledVerde, ledVermelho);
if( HEX == 7'b0001110 ) $display( " F" ); if( HEX == 7'b1000000 )
$display (" 0" ); if( HEX == 7'b0001000 ) $display( " A" );
end
endmodule

```

3 Avaliação dos resultados do experimento

Ao executar o *test bench* obteve-se os resultados conforme as Figura 3 e Figura 4.

Figura 3 – *Test bench Wave* do código da máquina.

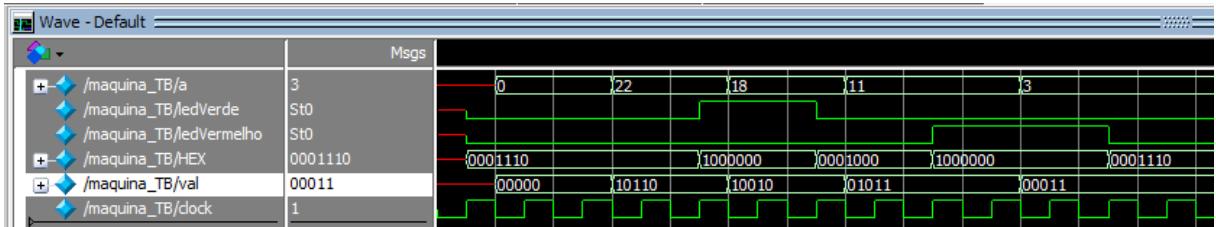


Figura 4 – *Test bench Transcript* do código da máquina.

```
#      Sistema Porta
#
# Estado | Entrada | LG | LR | HEX
# -----
# Fechado | 00000 | 0 | 0 | F
# Abrindo | 10110 | 1 | 0 | O
# Aberto | 10010 | 0 | 0 | A
# Fechando | 01011 | 0 | 1 | O
# Fechado | 00011 | 0 | 0 | F
```

Como resultado do *deploy* na placa, obteve o resultado conforme a Figura 5

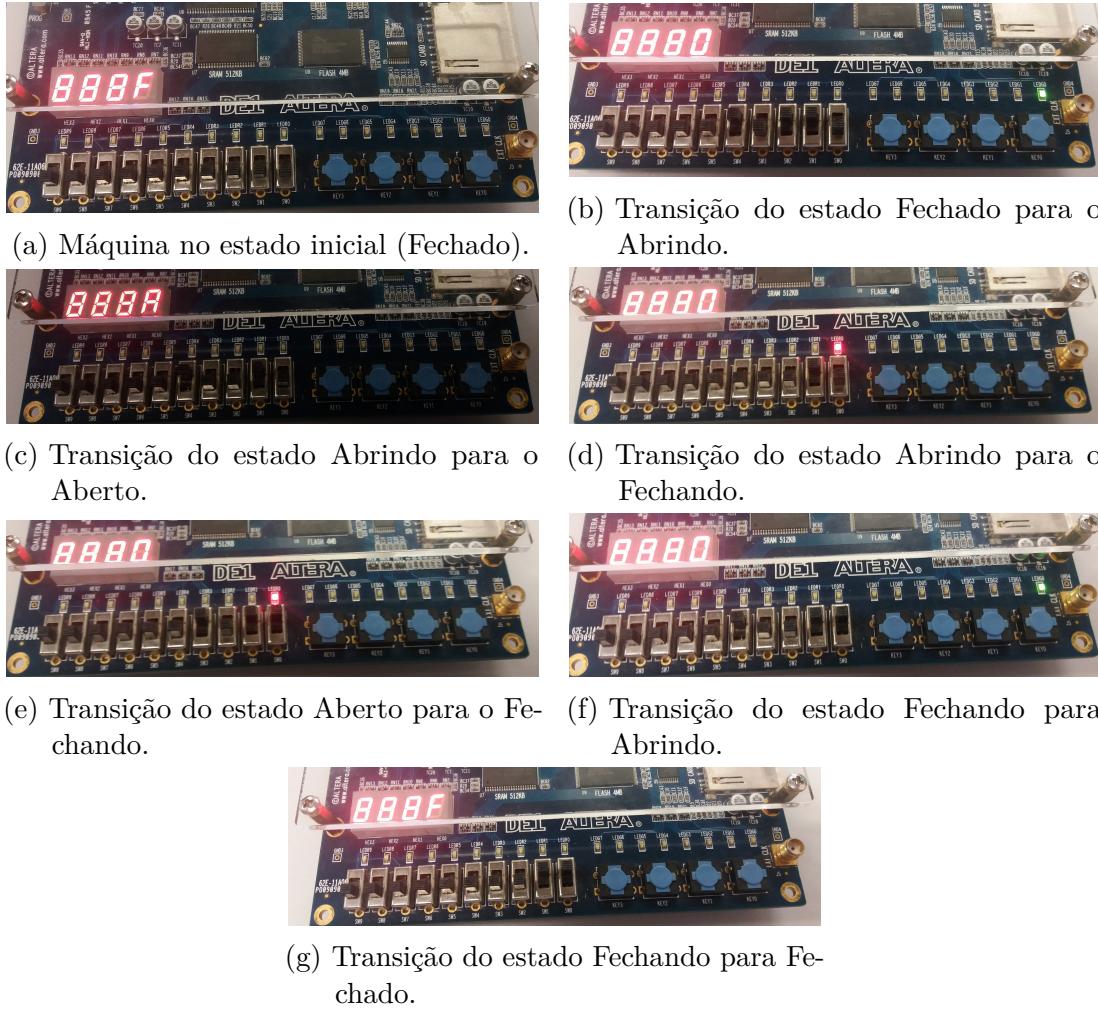


Figura 5 – Teste do circuito rodando na placa.

4 Análise crítica e discussão

O Código 2.1 engloba todos as entradas, todos os estados e suas saídas, assim, se fosse um circuito sequêncial, englobaria o circuito de excitação, circuito de memória e o circuito de saída, ou seja, não foi necessário criar um módulo para cada parte da máquina.

Constatou-se que para as entradas que não aparecem no Código 2.1, a máquina mantém-se no estado atual, isso era o esperado para uma máquina de estado.

Teve-se dificuldade de pensar em quais seriam as entradas necessárias para resolver o problema da garagem, entender como passar a ideia de máquina de estado para um código em Verilog, principalmente quando não estava-se sendo utilizado o sinal de *clock*, assim sendo necessário refazer o código mais de 5 (cinco) vezes.

5 Outras informações

Para estudar como passar a ideia da máquina de estado para um código em Verilog, pensou-se numa máquina simples, que tem como estado inicial o estado A, que vai para um estado B e o B vai para o A. Assim, para determinar em que estado a máquina encontra-se, estipulou-se LEDs para representar o estado atual, LED verde para o estado A e para B um LED vermelho.

O desenho da máquina encontra-se na [Figura 6](#), o Verilog no [Código 5.1](#), a compilação na [Figura 7](#), código do teste no [Código 5.2](#) e os testes nas [Figura 8](#) e [Figura 9](#).

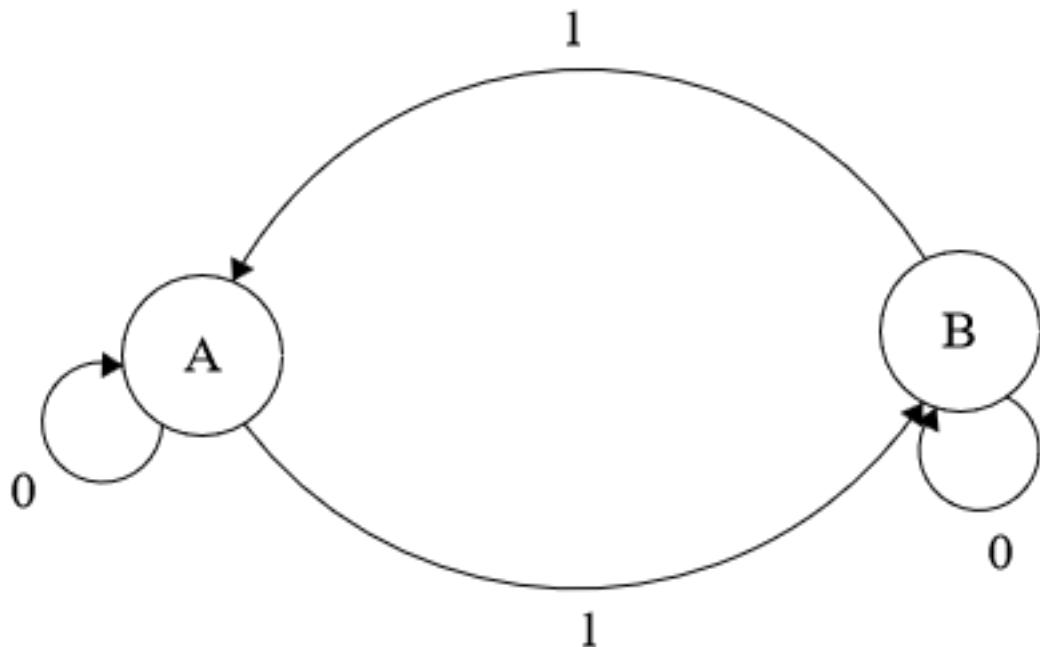


Figura 6 – Máquina de estado do circuito adicional.

Código 5.1 – Código da máquina de estado adicional.

```

module inicial( botao, ledVerde, ledVermelho, clock);
    input botao, clock;
    output ledVerde, ledVermelho;

    reg tmpLedVerde, tmpLedVermelho;

    parameter A = 1'b0, B = 1'b1;

    reg estado = A;

```

```

always @ (posedge clock) begin
    case( estado )
        A: begin
            tmpLedVerde = 1;
            tmpLedVermelho = 0;

            if( botao )
                estado = B;
        end

        B: begin
            tmpLedVerde = 0;
            tmpLedVermelho = 1;

            if( botao )
                estado = A;
        end

        default: estado = A;
    endcase
end

assign ledVerde = tmpLedVerde;
assign ledVermelho = tmpLedVermelho;
endmodule

module circuitoAdicional ( KEY , LEDG , LEDR , CLK );
    input [0:0] KEY;
    input CLK;
    output [0:0] LEDG , LEDR;

    inicial i( KEY[0] , LEDG , LEDR , CLK );
endmodule

```

Flow Summary	
Flow Status	Successful - Fri Dec 08 15:20:51 2017
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	circuitoAdicional
Top-level Entity Name	circuitoAdicional
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	3 / 18,752 (< 1 %)
Total combinational functions	2 / 18,752 (< 1 %)
Dedicated logic registers	3 / 18,752 (< 1 %)
Total registers	3
Total pins	4 / 315 (1 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 7 – Compilação da máquina de estado do circuito adicional.

Código 5.2 – Código do *test bench* da máquina de estado adicional.

```

module circuitoAdicional_TB;

    wire ledVerde, ledVermelho;
    reg clock, botao;

    inicial i( botao, ledVerde, ledVermelho, clock);

    //Alterando clock
    always begin
        clock <= 0;
        #25;
        clock <= 1;
        #25;
    end

    initial begin
        $display("\tSistema Porta\n");
        $display("Estado | Entrada | LG | LR");
        $display("-----");
    end

    //Estado A
    #50
    botao = 0;#100
    $display( " A | %x | %x | %x ", botao, ledVerde,
    ledVermelho);

    //Estado B
    botao = 1;#100
    $display( " B | %x | %x | %x ", botao, ledVerde,
    ledVermelho);

    //Estado A
    botao = 1;#150
    $display( " A | %x | %x | %x ", botao, ledVerde,
    ledVermelho);

    end
endmodule

```

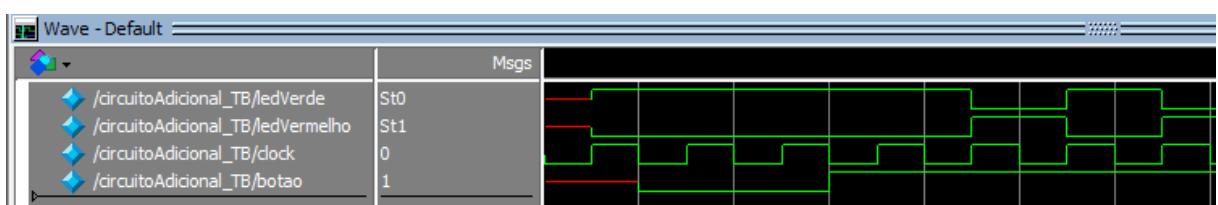


Figura 8 – *Test bench Wave* da máquina de estado adicional.

```
#      Sistema Porta
#
# Estado | Entrada | LG | LR
# -----
#   A   |    0    |  1 |  0
#   B   |    1    |  0 |  1
#   A   |    1    |  1 |  0
```

Figura 9 – *Test bench Transcript* da máquina de estado adicional.