Departamento de Ciência da Computação

Prof. Bruno de Abreu Silva

GCC260 – Laboratório de Circuitos Digitais

Máquinas de Estados e gráficos usando controlador VGA e memória ROM

1. Objetivos

- Aprender a implementar máquinas de estados em Verilog;
- Entender o funcionamento da saída VGA;
- Entender como objetos simples podem ser desenhados na tela;
- Aprender como controlar objetos da tela usando as entradas do kit DE10-Lite;
- Ver um exemplo de uso de memória ROM em Verilog.

2. Ideia básica do projeto

Considere um robô que está em constante movimento. Porém, é possível enviar dois comandos para controlar a sua trajetória: **virar para a direita** ou **virar para a esquerda**. Sendo assim, independentemente de qual seja a sua posição, o robô estará se movendo de quatro formas possíveis, representadas na Figura 1: para frente, para baixo, para trás e para cima.

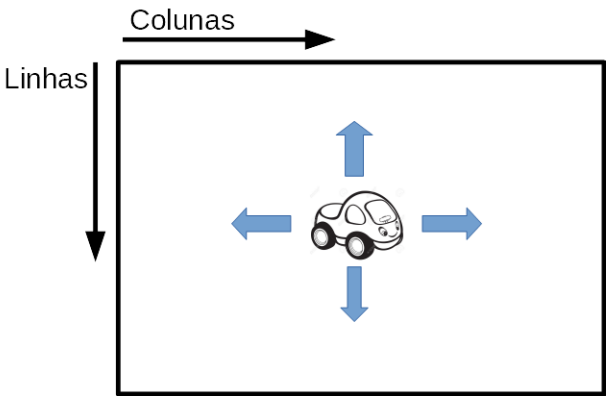

Figura 1: Movimento do robô.

Portanto, podemos representar o movimento do robô por meio do seguinte diagrama de transição de estados:

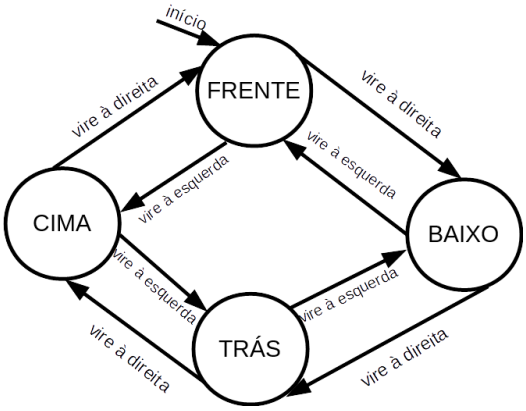

Figura 2: Diagrama de transição de estados do movimento do robô.

Em nosso projeto, o robô será representado por um quadrado na tela do monitor. A tela será representada por uma matriz de pixels, no padrão VGA com resolução de 640x480. Sendo assim, as coordenadas de cada pixel, vão de (0,0) no canto superior esquerdo até (639,479), no canto inferior direito. A partir de uma posição inicial do robô, a posição seguinte é definida de acordo com qual estado o robô está: frente, baixo, para trás ou cima. E a alteração da posição se dá pelo incremento ou decremento da linha ou coluna da coordenada atual da posição do robô. Então, os movimentos podem ser definidos da seguinte forma:

- FRENTE: Não altera a linha, incrementa a coluna;
- BAIXO: Incrementa a linha, não altera a coluna;
- TRÁS: Não altera linha, decrementa a coluna;
- CIMA: Decrementa linha, não altera coluna.

3. Atividades

As atividades práticas serão divididas em duas etapas: primeiramente será trabalhada a implementação da máquina de estados e, posteriormente, serão trabalhadas questões sobre o desenho de objetos na tela com mapeamento de objetos retangulares e mapeamento por bitmap de objetos não retangulares.

3.1. Máquina de estados modelo de Moore

Inicialmente, vamos entender como uma máquina de estados pode ser implementada em Verilog, veja o exemplo na Figura 3. Faça a leitura e entendimento do código, desenhe o diagrama de transição de estados dessa máquina e tire suas dúvidas e apresente ao Professor.

```verilog
module moore_machine(
    input wire clk, rstn,
    input wire data_in,
    output reg [1:0] data_out
);
    localparam S0 = 0, S1 = 1, S2 = 2, S3 = 3; // Declare states
    reg [1:0] state_reg, state_next; // Signal declaration

    // state register
    always@(posedge clk, negedge rstn)
       if(!rstn)
          state_reg <= S0; // Initial state
       else
          state_reg <= state_next;

    // Moore Output depends only on the state
    always @ (state_reg)
    begin
       case(state_reg)
          S0:
             data_out = 2'b01;
          S1:
             data_out = 2'b10;
          S2:
             data_out = 2'b11;
          S3:
             data_out = 2'b00;
          default:
             data_out = 2'b00;
       endcase
    end

    // Determine the next state
    always @ *
    begin
       case(state_reg)
          S0:
             state_next = S1;
          S1:
             if(data_in) state_next = S2;
             else state_next = S1;
          S2:
             if(data_in) state_next = S3;
             else state_next = S1;
          S3:
             if(data_in) state_next = S2;
             else state_next = S3;
          default:
             state_next = S0;
       endcase
    end
endmodule
```

Figura 3: Código exemplo de máquina de estados modelo de Moore em Verilog.

Boa parte do circuito necessário para o desenvolvimento desta atividade já está implementado. Faça o download do arquivo *Pratica10.qar* (https://github.com/brabreus/GCC260-UFLA/tree/main/Pratica10) e abra no software Quartus Prime. Somente um componente está incompleto e deve ser completado: a máquina de estados que controla o movimento do robô, no arquivo já criado *DirectionController.v*.

Implemente uma máquina de estados em Verilog, usando o modelo de Moore, que seja capaz de controlar o movimento do robô. Esse módulo deve possuir uma entrada de *clock (clk)*, uma entrada para receber o comando para virar à direita (*turn_right*), uma entrada para receber o comando para virar à esquerda (*turn_left*), uma entrada de *reset ativo em 0 (rstn)* e uma saída de 4 bits (*data_out*), conforme a Figura 4:

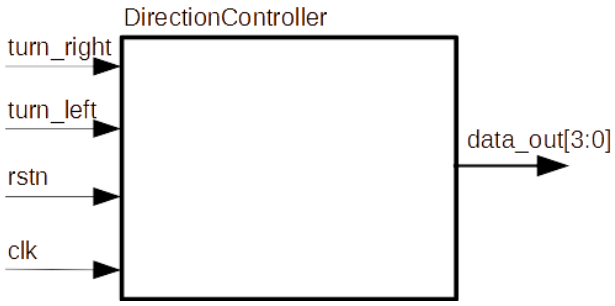

Figura 4: Diagrama de bloco da máquina de estados.

Aos quatro bits de saída *data_out* deverão ser atribuídos os seguintes valores, de acordo com o estado atual da máquina:

- 4'b0011: FRENTE - Ativa incremento da coluna, desativa a linha;
- 4'b1100: BAIXO - Desativa coluna, ativa incremento da linha;
- 4'b0001: TRÁS - Ativa decremento da coluna, desativa linha;
- 4'b0100: CIMA - Desativa coluna, ativa decremento da linha.

Para projetar a máquina, parta do código inicial fornecido no arquivo *DirectionController.v* e use como exemplo o código da Figura 3, fazendo as devidas alterações para que represente o funcionamento adequado.

Após ter concluído as modificações no arquivo *DirectionController.v*, recompile o projeto e acesse Tools->Netlist Viewers->RTL Viewer e clique duas vezes no componente *DirectionController,* que está dentro do componente *Square*, dentro de *PixGen*. Logo depois, clique em *state_reg* (bloco amarelo) até visualizar o diagrama de transição de estados implementado. Feito isso, compile o projeto e chame o professor para receber instruções sobre como verificar se o circuito está correto e também como ele será executado.

DICA: Implemente primeiro somente os movimentos para a direita (usando *if*). Depois, implemente os movimentos para a esquerda (usando *else if*). Use o *else* para manter o estado atual da máquina.

3.2. Desenho de objetos retangulares e não retangulares

O arquivo *Square.v* implementa um quadrado por meio da verificação se a coordenada atual do pixel está dentro da região definida por uma expressão. Veja na linha 26 o comando que define a região do quadrado:

assign square_on = (x_count < x) && (x < x_count + SQUARE_SIZE) && (y_count < y) && (y < y_count + SQUARE_SIZE);

Dessa maneira, é possível desenhar objetos simples, tipicamente, com formas retangulares. Tente brincar como arquivo Square.v mudando atributos do quadrado, como tamanho, cor e velocidade e verifique os resultados na placa.

O desenho de objetos não retangulares pode ser feito por meio da implementação de uma memória ROM que armazena o bitmap do objeto a ser desenhado. O código então deve verificar se o pixel atual faz parte da região delimitada para o desenho do objeto e logo em seguida realizar a leitura da posição correta na memória ROM. Caso o bit armazenado na memória ROM seja 1, o pixel correspondente será colorido.

Para verificar um exemplo de um objeto não retangular, substitua toda a linha 26 do arquivo Square.v pelo seguinte trecho de código:

```
wire [2:0] rom_addr, rom_col;
    reg [7:0] rom_data;
    wire rom_bit, sq_ball_on;

    // round ball image ROM
    always@*
    case (rom_addr)
        3'h0: rom_data = 8'b00111100; //   ****
        3'h1: rom_data = 8'b01111110; //  ******
        3'h2: rom_data = 8'b11111111; // ********
        3'h3: rom_data = 8'b11111111; // ********
        3'h4: rom_data = 8'b11111111; // ********
        3'h5: rom_data = 8'b11111111; // ********
        3'h6: rom_data = 8'b01111110; //  ******
        3'h7: rom_data = 8'b00111100; //   ****
```

```
endcase

// pixel within ball
assign sq_ball_on = (x_count <= x) && (x < x_count + SQUARE_SIZE) &&
                    (y_count <= y) && (y < y_count + SQUARE_SIZE);
// map current pixel location to ROM addr/col
assign rom_addr = y[2:0] - y_count[2:0];
assign rom_col = x[2:0] - x_count[2:0];
assign rom_bit = rom_data[rom_col];
// pixel within ball
assign square_on = sq_ball_on & rom_bit;
```

Redefina o valor de SQUARE_SIZE para 8 na linha 14, pois a memória ROM é de tamanho 8x8, conforme abaixo:

localparam SQUARE_SIZE = 8;

O restante do código deverá ser mantido. Recompile o projeto e verifique seu funcionamento na placa.

Por fim, apresente ao Professor e envie a pasta compactada para o Campus Virtual.