Universidade Federal de São João del-Rei
Ciência da Computação – 2ºperíodo

“Circuito de controle de ocupação de um estacionamento”

Trabalho apresentado à disciplina
Introdução aos Sistemas Lógicos Digitais,
professor Hilton de Oliveira Mota

Isabella Vieira Ferreira
Lucivânia Ester da Costa
Mônica Neli de Resende

São João del-Rei, 09 de dezembro de 2011

Sumário

pág.

Lista de abreviações

MEF : Máquina de estados finitos

MUX: Multiplexador

DECODER BCD: decodificador BCD

DECODER 2X4: decodificador 2x4

FFD: *flip-flop* D

Lista de figuras

pág.

Lista de tabelas

pág.

1. Introdução

Os circuitos lógicos digitais são de extrema importância na atualidade, pois nos ajudam a controlar e organizar os dados de forma adequada e eficaz. Encontramos circuitos desde os mais “simples” até os mais “complexos” de serem projetados e são muito utilizados em aparelhos do nosso cotidiano, como por exemplo, relógios, celulares e computadores.

Um circuito eficiente e que é útil para controle de dados é o circuito de controle de fluxo de um estacionamento, que iremos descrevê-lo neste trabalho. O mesmo possui em sua estrutura um *display* digital e recebe informações de foto-sensores instalados na via de entrada do estacionamento. Dessa forma é possível visualizar que um carro tenha entrado ou saindo do estacionamento. A seguir mostraremos a descrição das estratégias para projetar esse ciruito.

2. Descrição do problema

Conforme nos foi proposto, foi implementado um circuito para monitorar o fluxo de carros em um estacionamento de uma única via de entrada e saída. Essa via possui dois pares de foto-sensores (foto-transmissor e foto-receptor) monitorando a atividade dos veículos e desprezando situações adversas, como por exemplo, pedestres ou veículos que não completaram a entrada ou a saída.

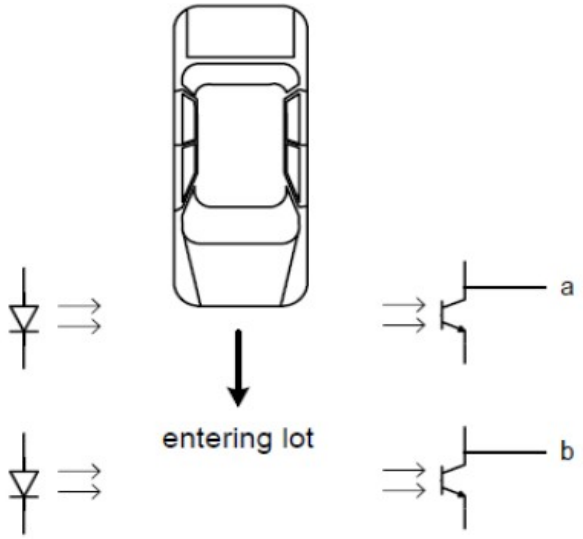

Figura 1: Simulação de entrada

Se não há nada entre o foto-transmissor e foto-receptor a saída tem nível lógico 0, e caso tenha algum objeto bloqueando a luz, a saída passa a ter nível lógico 1, sendo que as saídas dos foto-sensores são representadas como “A” e “B”. De acordo com uma sequência de ocorrências dessas saídas é verificado se o veículo está entrando ou saindo (bem como saber se é realmente um veículo que está passando). Com essa sequência o número de veículos presentes no estacionamento será controlado por meio de um contador: crescente para entrada e decrescente para saída. Como o limite máximo de vagas no estacionamento é 2500, se a quantidade de veículos exceder o intervalo $0 \leq x \leq 2500$, o circuito impede o *underflow* e *overflow*.

2.1 Sequências de ocorrência dos controles “A” e “B”

As tabelas a seguir identificam as sequências válidas de modo a detectar apenas automóveis:

- Veículo entrando no estacionamento:

Contar	
A	B
0	0
1	0
1	1
0	1
0	0

Tabela 1: Sequência válida para veículo entrando no estacionamento.

- Veículo saindo do estacionamento:

Up_down	
A	B
0	0
0	1
1	1
1	0
0	0

Tabela 2: Sequência válida para veículo saindo do estacionamento.

Qualquer situação que deixe a sequência incompleta ou que difere das mencionadas acima, significa que:

- O veículo não entrou ou saiu do estacionamento;
- Não é um veículo.

Sendo essa sequência inválida, o algoritmo não conta nem decrementa.

Com isso para o controle do fluxo de veículos no estacionamento foram utilizados as seguintes estratégias:

- Máquina de estados finitos;
- Contador BCD;
- Acionador dos displays.

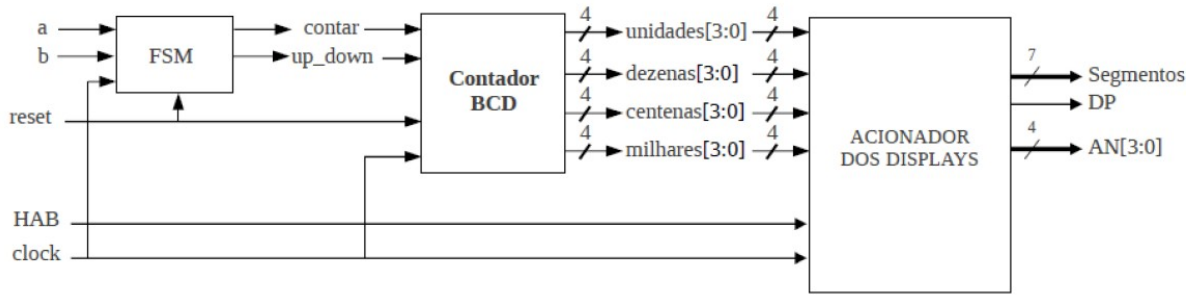

Figura2 : representação em módulos do circuito lógico.

3. Descrição da estratégia utilizada para implementar a máquina de estados finitos

Para identificar as sequências de entrada e saída de veículos utilizamos uma máquina de estados finitos, de modo que ao completar a entrada ou saída de um veículo a saída da MEF fosse 1. Para tal a implementação, a MEF foi baseada no Modelo de Moore, que monitora as saídas em função do estado atual.

Abaixo temos o diagrama de estados que representa a máquina de estados finitos:

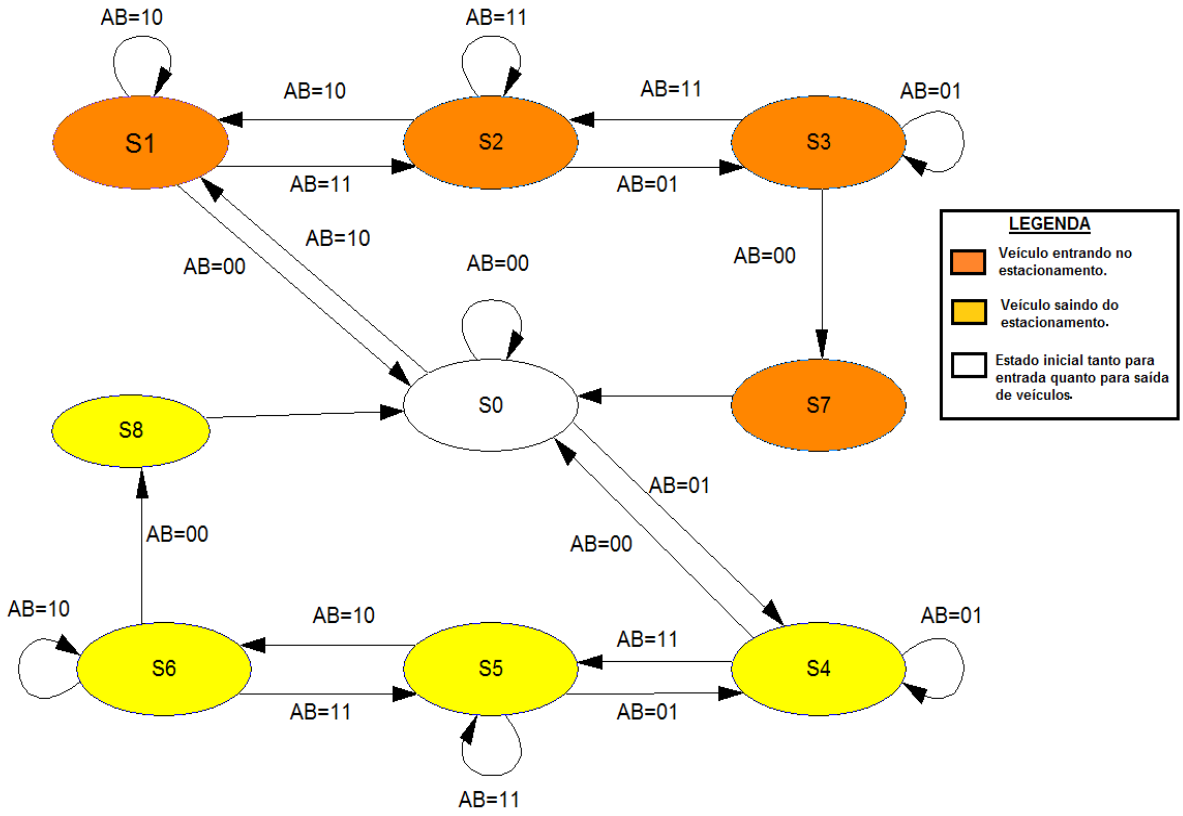

Figura 3: Diagrama de estados para a máquina de estados finitos

3.1 Descrição dos estados atuais

S0: estado *default*, quando não tem nenhum objeto entre os foto-sensores;

S1: estado que identifica uma possível entrada, bloqueando o primeiro foto-sensor;

S2: estado em que o veículo bloqueia os dois foto-sensores;

S3: estado em que o veículo desbloqueia o primeiro foto-sensor;

S7: estado em que se confirma a entrada do veículo;

S4: estado que identifica uma possível saída;

S5: estado em veículo bloqueia os dois foto-sensores;

S6: estado em que o veículo desbloqueia o segundo foto-sensor;

S8: estado em que se confirma a saída do veículo.

Para a implementação da MEF utilizamos dois blocos: um sequencial que define a sensibilidade ao *clock* e ao *reset* como parâmetros do *always*, e outro combinacional para o controle das variáveis “estado_atual” e “prox_estado” utilizando o modelo comportamental.

No bloco sequencial, caso o *reset* seja ativado, as saídas e os controles são colocados em estado inicial (zero), caso contrário o “estado_atual” é atualizado com o “prox_estado”.

```
//Controle do clock e do reset na borda de
subida  Sequencial
      always @(posedge CLK, posedge reset)
      if (reset)
      begin
            estado_atual <= s0;
      end
             else
            estado_atual <= prox_estado;
```

Figura 4 : Descrição do bloco sequêncial da MEF

No bloco combinacional utilizamos uma estrutura de controle *case* para monitorar o “estado_atual” e em cada possibilidade testamos os controles “A” e “B” a partir da mesma estrutura. Contudo, o “prox_estado” será definido de acordo com o “estado_atual” e os controles.

```
//Controle das entradas - Combinacional
always @ *
begin
case(estado_atual)
      //Estado inicial.
      S0: case({A,B})
      2'b00: prox_estado = s0;
            2'b01: prox_estado = s4;
            2'b10: prox_estado = s1;
            default: prox_estado = s0;
         endcase
                              ⋮
                    //Outros estados

     default : prox_estado = s0;
endcase
```

Figura 5: Descrição do bloco combinacional da MEF

Ao se confirmar uma entrada ou saída, a MEF gera duas saídas “contar” e “up_down” de acordo com as especificações abaixo:

Reset	**Contar**	**Up_down**	**Clock**	**Ação**
1	x	x	↑	Volta ao estado inicial
0	0	x	↑	Repouso
0	1	0	↑	Conta crescente
0	1	1	↑	Conta decrescente

Tabela 3: Especificações para implementação da MEF

- A saída “contar” é ativada quando está no estado “S7” ou “S8”;
- A saída “up_down” é ativada quando está no estado “S8”;

4. Descrição da estratégia para implementação do contador BCD.

A partir das saídas da MEF (“contar” e “up_down”) a contagem ocorre no módulo “cont_bcd” dividido em unidade, dezena, centena e milhar.

Na implementação utilizamos um bloco sequencial que define a sensibilidade ao *clock* e ao *reset* como parâmetros do *always*. Caso o *reset* seja ativado as saídas são colocadas em estado inicial (zero) utilizando as estruturas de controle *if-else*.

4.1 Contagem crescente

De acordo com as especificações para contagem crescente (tabela 3) o algoritmo verifica se o número de veículos ultrapassou 2500. Se não tiver ultrapassado soma um na unidade. Se a mesma for nove, então recebe zero e soma um na dezena fazendo as mesmas verificações até a centena. Termina somando um na casa do milhar, e caso a contagem já tenha atingido 2500 o número não é mais acrescido.

4.2 Contagem decrescente

Também de acordo com as especificações (tabela 3) o algoritmo verifica primeiramente se as casas da unidade, dezena, centena e milhar são simultaneamente zero, caso seja o número não é decrescido. Do contrário decresce um na unidade. Se a mesma for zero então recebe nove e decresce na dezena fazendo as mesmas verificações até a centena. Termina decrementando um na casa do milhar.

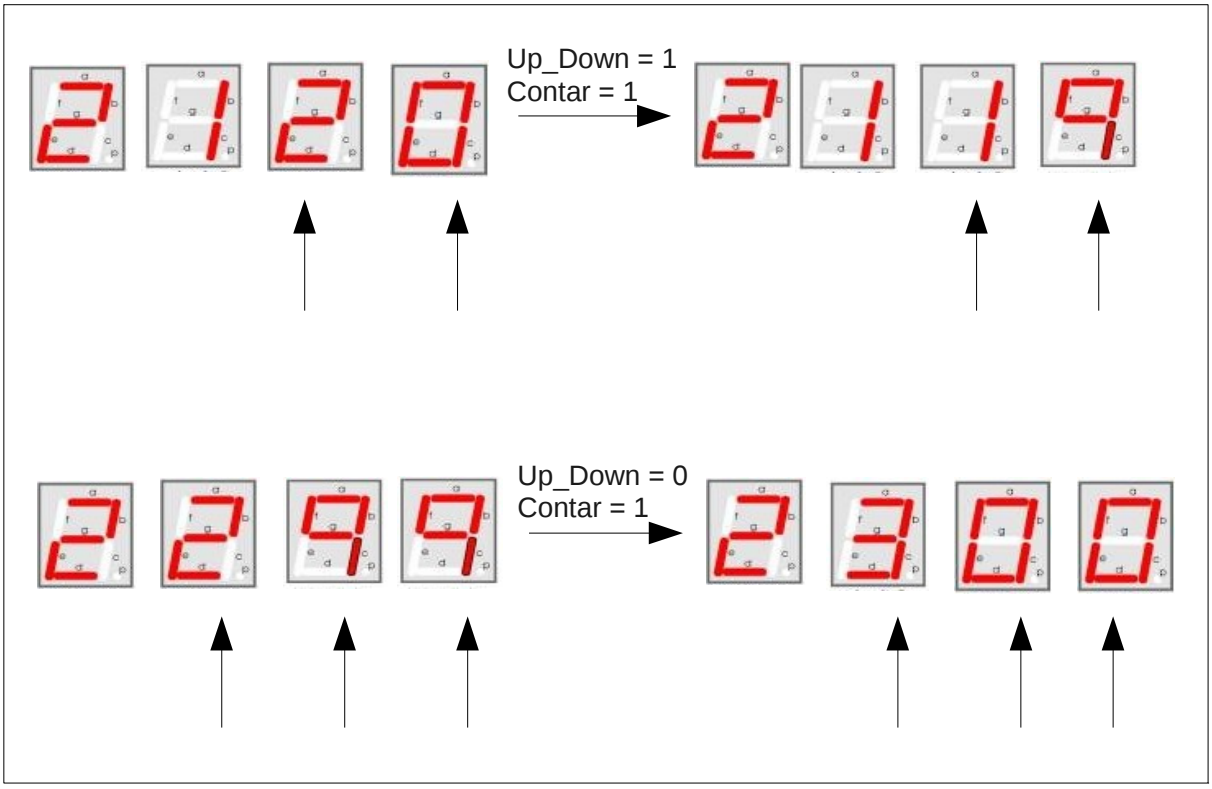

Figura 6: Demonstração dos sinais “contar” e “up_down”

5. Acionador dos *displays*

O número de veículos presentes no estacionamento deverá ser apresentado nos *displays* do módulo de desenvolvimento *Nexys* 2. Para fazer os números referentes à contagem aparecerem nesses *displays* foram utilizados as seguintes estratégias:

- Multiplexador 4x5;
- Decodificador BCD – 7 segmentos;
- Decodificador 2x4;
- Contador de 2 bits;
- Contador de 17 bits (condicionador de clock).

Cada módulo será conectado da seguinte forma:

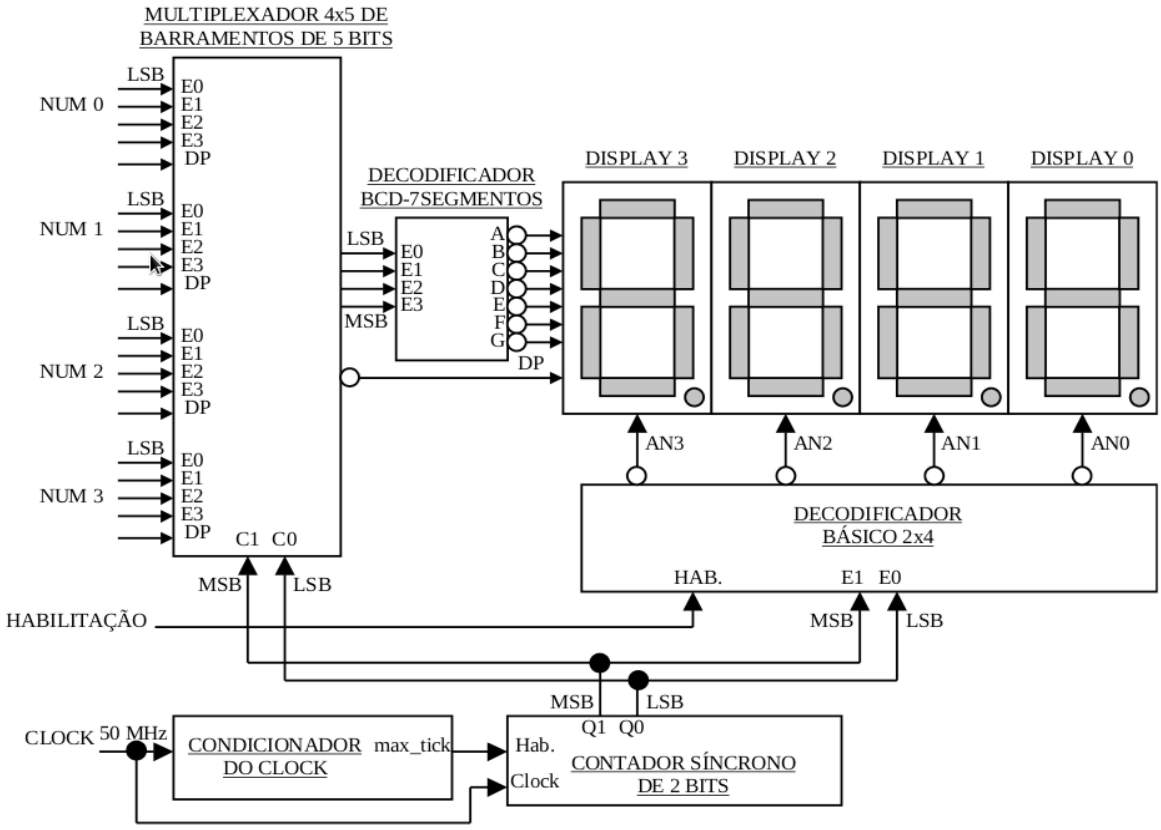

Figura 7: Módulos do acionador de *displays*.

5.1 Multiplexador 4x5

Responsável por selecionar cada número referente à unidade, dezena, centena e milhar conforme as entradas de controle “sel1” e “sel2”.

Controles		**Saída**
Sel1	**Sel2**	
0	0	Num0 (unidade)
0	1	Num1 (dezena)
1	0	Num2 (centena)
1	1	Num3 (milhar)

Tabela 4: Seleção dos números no multiplexador

Na implementação (combinacional) do módulo “mux_4x5” utilizamos o modelo comportamental com a estrutura de controle c*ase.*

5.2 Decodificador BCD – 7 segmentos

O decodificador BCD – 7 segmentos recebe como entrada o número do MUX e gera um número de 7 bits responsável por acender os *leds* dos *displays* considerando a lógica negativa dos mesmos. Na implementação (combinacional) do módulo “Decoder_BCD_7seg” utilizamos modelagem por fluxo de dados.

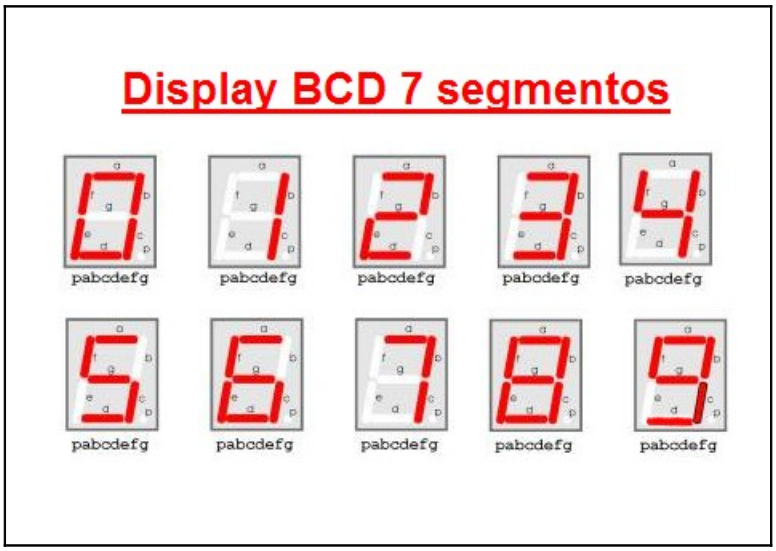

Figura 8 : Representação de cada número no *display* BCD – 7 segmentos

Segue abaixo a tabela-verdade utilizada para implementação do DECODER BCD. Como os *displays* do módulo de desenvolvimento *Nexys* 2 possuem lógica negativa por isso na tabela 0 representa onde o *display* terá que acender e 1 onde terá que apagar.

Entradas				Saídas						
E3	E2	E1	E0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0
1	0	1	0	0	1	1	0	0	0	0
1	0	1	1	0	1	1	0	0	0	0
1	1	0	0	0	1	1	0	0	0	0
1	1	0	1	0	1	1	0	0	0	0
1	1	1	0	0	1	1	0	0	0	0
1	1	1	1	0	1	1	0	0	0	0

Tabela 5: Tabela-verdade do DECODER BCD

Mapas de *Karnaugh* utilizados para simplificação das expressões das saídas:

Saída a

	~E1 & ~E0	~E1 & E0	E1 & E0	E1 & ~E0
~E3 & ~E2	0	(1)	0	0
~E3 & E2	(1)	0	0	0
E3 & E2	0	0	0	0
E3 & ~E2	0	0	0	0

Equação booleana simplificada: a= (~E3 & E2 & ~E1 & ~E0) |(~E3 & ~E2 & ~E1 & E0)

Tabela 6: Mapa de *Karnaugh* para a saída “a” do DECODER BCD.

Saída b

	~E1 & ~E0	~E1 & E0	E1 & E0	E1 & ~E0
~E3 & ~E2	0	0	0	0
~E3 & E2	0	1	0	1
E3 & E2	1	1	1	1
E3 & ~E2	0	0	1	1

Equação booleana simplificada: b = E2 & (E0 ^ E1) | (E3 & E2) | (E3 & E1)

Tabela 7: Mapa de *Karnaugh* para a saída "b" do DECODER BCD.

Saída c

	~E1 & ~E0	~E1 & E0	E1 & E0	E1 & ~E0
~E3 & ~E2	0	0	0	1
~E3 & E2	0	0	0	0
E3 & E2	1	1	1	1
E3 & ~E2	0	0	1	1

Equação booleana simplificada: c = ~E3 & E0 & (~(E2 ^ E1))

Tabela 8: Mapa de *Karnaugh* para a saída "c" do DECODER BCD.

Saída d

	~E1 & ~E0	~E1 & E0	E1 & E0	E1 & ~E0
~E3 & ~E2	0	1	0	0
~E3 & E2	1	0	1	0
E3 & E2	0	0	0	0
E3 & ~E2	0	0	0	0

Equação booleana simplificada:
d = (~E3 & E2 & ~E1 & ~E0) |(~E3 & ~E2 & ~E1 & E0) | (~E3 & E2 & E1 & E0)

Tabela 9: Mapa de *Karnaugh* para a saída "d" do DECODER BCD.

Saída e

	~E1 & ~E0	~E1 & E0	E1 & E0	E1 & ~E0
~E3 & ~E2	0	1	1	0
~E3 & E2	1	1	1	0
E3 & E2	0	0	0	0
E3 & ~E2	0	1	0	0

Equação booleana simplificada: e = (~E3 & E2 & ~E1) | (~E2 & ~ E1 & E0) | (~E3 & E0)

Tabela 10: Mapa de *Karnaugh* para a saída "e" do DECODER BCD.

Saída f

	~E1 & ~E0	~E1 & E0	E1 & E0	E1 & ~E0
~E3 & ~E2	0	1	1	1
~E3 & E2	0	0	1	0
E3 & E2	0	0	0	0
E3 & ~E2	0	0	0	0

Equação booleana simplificada: f = (~E3 & ~E2 & E0) |(~E3 & E1 & E0) | (~E3 & ~E2 & E1)

Tabela 11: Mapa de *Karnaugh* para a saída "f" do DECODER BCD.

Saída g

	~E1 & ~E0	~E1 & E0	E1 & E0	E1 & ~E0
~E3 & ~E2	1	1	0	0
~E3 & E2	0	0	1	0
E3 & E2	0	0	0	0
E3 & ~E2	0	0	0	0

Equação booleana simplificada: g = (~E3 & E2 & E1 & E0) |(~E3 & ~E2 & ~E1)

Tabela 12: Mapa de *Karnaugh* para a saída "g" do DECODER BCD.

5.3 Decodificador 2x4

O número gerado pelo DECODER BCD será mostrado no *display* selecionado e ativado pelo decodificador 2x4. A seleção e ativação é dada de acordo com as entradas de controle e a habilitação, considerando a lógica negativa dos *displays* da placa *Nexys* 2. Na implementação (combinacional) no módulo "Decoder_basic_2x4" utilizamos modelagem comportamental com estruturas de controle *if – else.*

Segue abaixo a tabela-verdade utilizada para implementação do DECODER 2x4.

Entradas			Saídas			
H	A	B	S3	S2	S1	S0
0	X	X	1	1	1	1
0	X	X	1	1	1	1
0	X	X	1	1	1	1
0	X	X	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	1	1	1	0

Tabela 13: Tabela-verdade do DECODER 2x4

5.4 Contador de 2 bits

As entradas de controle do DECODER 2x4 são geradas pelo contador de 2 bits, para que assim a seleção e ativação dos *displays* sejam alternadas com auxílio de uma entrada de habilitação. Na implementação foi utilizado modelagem comportamental utilizando *flip-flop* D.

Para modelagem do FFD utilizamos:

- diagrama de estados;
- tabela de transição de estados;
- mapas de *Karnaugh* contendo as equações simplificadas.

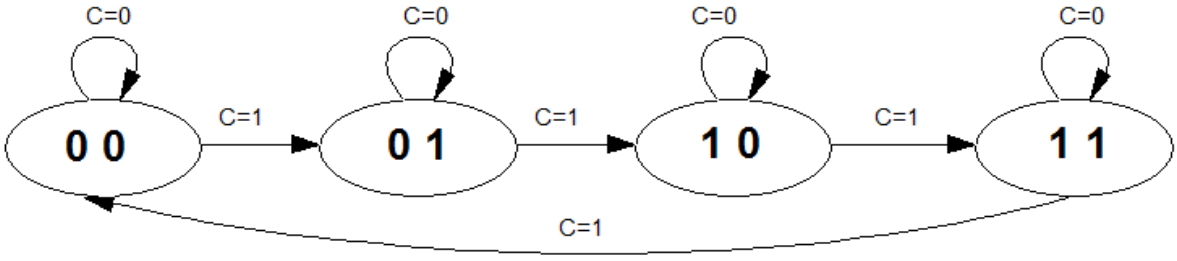

Figura 9: Diagrama de estados para o contador de 2 bits.

Entrada	Estado atual		CLOCK	Próx. Estado		Flip-flops	
C	Qa(t)	Qb(t)		Qa(t+1)	Qb(t+1)	Da	Db
0	0	0	↑	0	0	0	0
0	0	1	↑	0	1	0	1
0	1	0	↑	1	0	1	0
0	1	1	↑	1	1	1	1
1	0	0	↑	0	1	0	1
1	0	1	↑	1	0	1	0
1	1	0	↑	1	1	1	1
1	1	1	↑	0	0	0	0

Tabela 14: Tabela de transição de estados para o contador de 2 bits

Mapa do flip-flop Da

	~Qb	Qb
~C & ~Qa	0	0
~C & Qa	1	1
C & Qa	1	0
C & ~Qa	0	1

Equação booleana simplificada: (Qa & ~Qb) | (~C & Qa) | (C & ~Qa & Qb)

Tabela 15: Mapa de *Karnaugh* para o *flip-flop* D do contador de 2 bits.

Mapa do flip-flop Db

	~Qb	Qb
~C & ~Qa	0	1
~C & Qa	0	1
C & Qa	1	0
C & ~Qa	1	0

Equação booleana simplificada: C ^ Qb

Tabela 16: Mapa de *Karnaugh* para o *flip-flop* D do contador de 2 bits.

5.5 Contador 17 bits (condicionador de clock)

A frequência da placa *Nexys* 2 é de 50 MHz, ou seja, 50.000.000 ciclos por segundo. Porém não é necessário toda essa velocidade para alternar entre os *displays*, pois além de gastar energia gera um aquecimento. Devido a isso o contador de 2 bits possui uma entrada de habilitação que é ativada após uma contagem de modo a reduzir a frequência do clock. Como são quatro *displays* adotamos uma frequência de 100 Hz para cada um, logo:

$$\begin{aligned} 50\ \text{MHz} &\rightarrow 50.000.000\ \text{ciclos por segundo} \\ 400\ \text{Hz} &\rightarrow 400\ \text{ciclos por segundo} \end{aligned}$$

$$\frac{50\,000\,000}{400} = 125\,000$$

Todavia o contador deverá contar até 125.000 para reduzir a velocidade para 100 ciclos por segundo para cada *display*. A cada contagem até 125.000 o contador gera uma saída, que é a habilitação do contador de 2 bits.

Na implementação (sequencial) do módulo “contador17bits” utilizamos o modelamento comportamental com estruturas de controle *if - else.*

6. Resultados dos *testbenches*

De acordo com as implementações feitas obtivemos os seguintes resultados:

- **MEF**

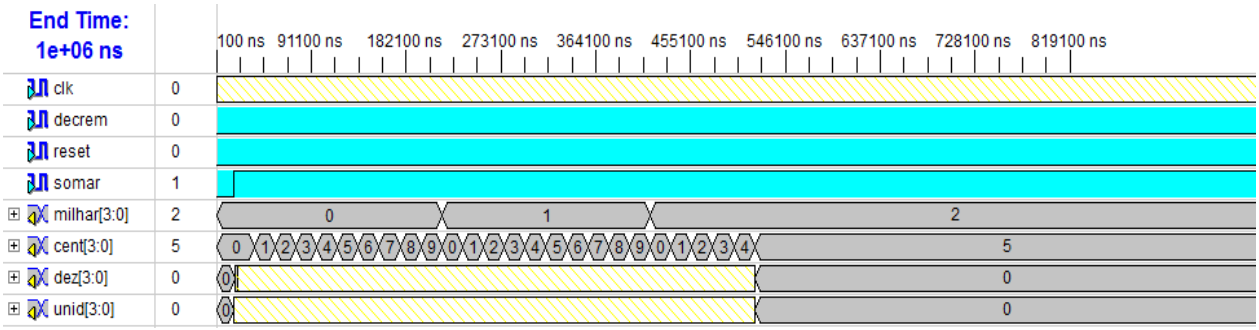

Figura 10: Testbench da máquina de estados finitos contado até seu limite máximo

Conforme o teste apresentado acima verifica-se que o circuito impede *overflow* e a realiza a contagem corretamente. Os espaços amarelos na unidade e na dezena acontecem pois o número de mudanças é tão grande que o teste não consegue apresentar.

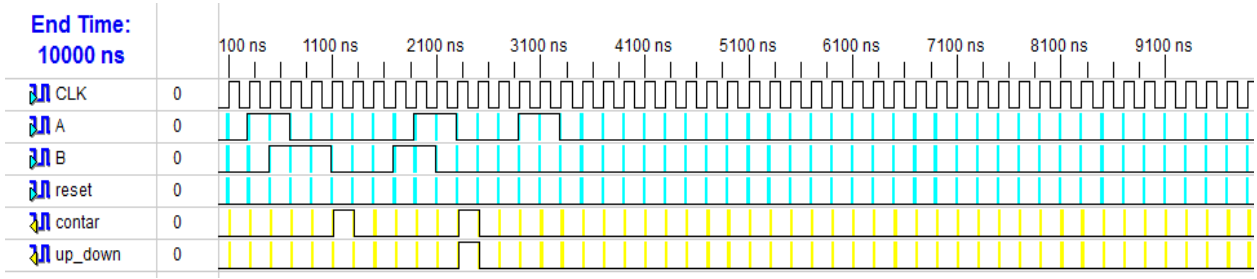

Figura 11: *Testbench* da máquina de estados finitos

O teste acima mostra a sequência de entrada, de saída e uma terceira sequência inválida. De acordo com as especificações, na sequência de entrada a saída “contar” foi
ativada e na sequência de saída “contar” e “up_down” . Como a terceira sequência não foi completada, ou seja, é inválida, o circuito não realiza nenhuma operação.

- **Contador BCD**

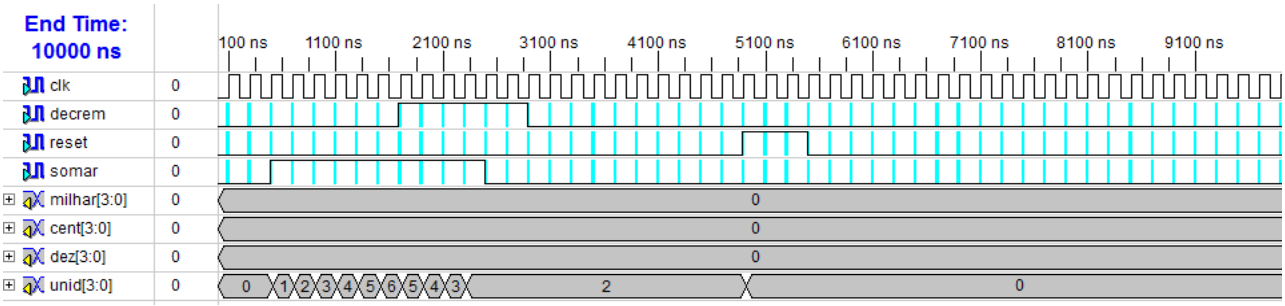

Figura 12: *Testbench* do contador BCD

O teste mostra o contador incrementado, ao ativar o "somar" e decrementando ao ativar o "decrem" e "somar" simultaneamente. Quando o *reset* é ativado todas as saídas são colocadas em nível lógico zero.

- **Multiplexador**

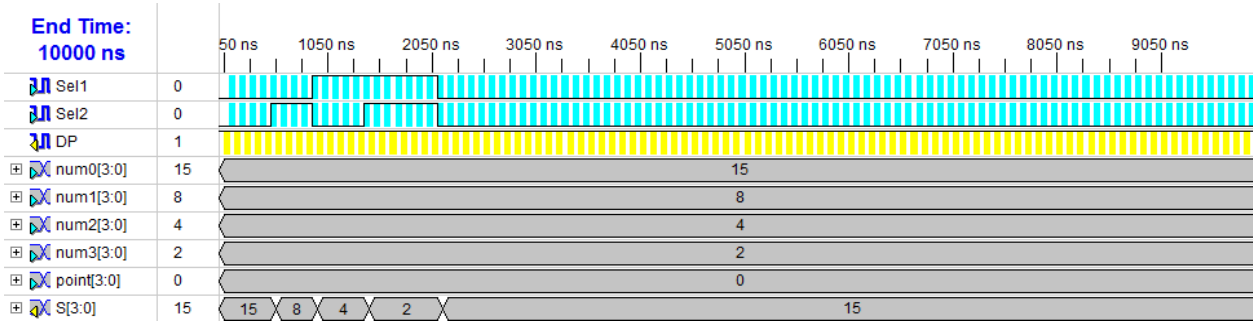

Figura 13: *Testench* do multiplexador

No teste apresentado acima atribuímos números quaisquer às variáveis "num0", "num1", "num2" e "num3" . De acordo com os controles "sel1" e "sel2" o MUX seleciona o número correspondente.

- **Contador de 2 bits**

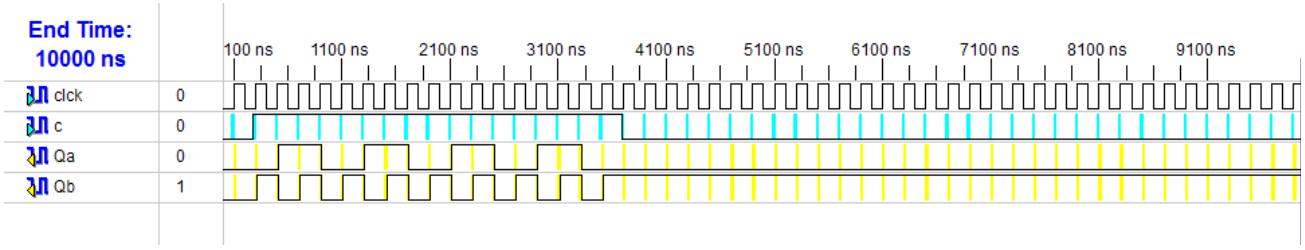

Figura 14: *Testbench* do contador de 2 bits

O teste apresenta a contagem correta de zero até três quando o controle está ativado e mantém o estado anterior caso o controle seja desativado.

- **Contador de 17 bits**

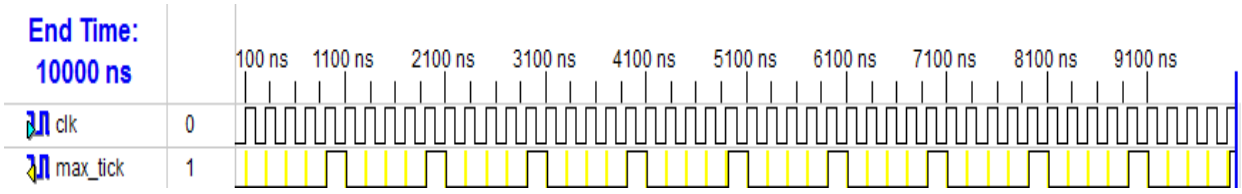

Figura 15: *Testbench* do contador de 17 bits (condicionador de clock)

Para mostar o comportamento da saída “max_tick” alteramos o contador para que seja ativada a cada contagem até cinco, pois do contrário não seria possível apresentar o resultado.

7. Diretrizes para efetuar os testes

Com as alterações abaixo o contador efetuará a soma de zero a 2500 com a habilitação ligada e se a sequência no controle “A” e “B” for correta, conforme especificado. Para zerar o contador aperte o push-button “H13”.

- **1º modo (manual):**

1. Chave de habilitação: K18;
2. Chave de controle “A”: H18;
3. Chave de controle “B”: G18;
4. *Reset: push-button* H13.

- **2 º modo (automático – contagem crescente/decrescente):**

Com as alterações abaixo o contador efetuará a soma de zero a 2500 automaticamente somente com a habilitação ligada, desprezando o controle “A” e “B”. Mantenha apertado o aperte o push-button “D18” para decrementar a partir de onde a soma parou. Para zerar o contador aperte o push-button “H13”.

No módulo principal:

1. Comente a linha 39 (“Maquina_de_estados”);
2. Descomente as linhas 24 e 53 (que possui o comentário //teste);
3. Comentar “*output* decre” na linha 25.

No módulo “implementacao.ucf”:

1. Descomentar linha 36 (#NET "decre" LOC = "D18");

No módulo “contador17bits”:

1. Comentar linhas 25, 29, 38, 43 e 51;
2. Descomentar linhas 26, 30, 39, 44 e 52;

OBS.: Se a habiliação estiver desativada a contagem será realizada, porém não será apresentada nos *displays.*

8. Conclusão

Durante a execução deste trabalho, uma das maiores dificuldades encontradas na implementação da MEF foi em como verificar se a sequência de entrada foi completada ou não. Para tal tentamos ativar a saída quando o "estado_atual" for S3 (estado em que o veículo desbloqueia o primeiro foto-sensor) e os controles "A" e "B" forem 0 e 0, de modo a ativar a saída de "contar" e "up_down" na transição do estado S3 para S0. Porém a implementação não apresentou os resultados desejados. Como solução acrescentamos mais dois estados S7 e S8 que seria seguinte aos estados S3 e S6 ,respectivamente.

Outra dificuldade foi em como desprezar situações de pedestres passando pelos foto-sensores. Para isso primeiramente utilizamos uma variável interna denominada "carro" que seria ativada no estado S2 onde os dois foto-sensores estariam ativados de modo a identificar um veículo. Porém percebemos que quando o pedestre passasse do estado S1 voltaria ao estado S0, dessa forma a variável foi removida.

Na implementação do Contador BCD em um primeiro momento fizemos os módulos de contagem separados, porém tivemos dificuldade em controlar o *underflow* e *overflow*. A solução deu-se ao perceber que seria mais adequado implementar o contador em um único módulo.

Como proposta de continuidade deste trabalho, caso seja um estacionamento rotativo, sugerimos que houvesse um sistema que capture a hora que o veículo entrou retornando o tempo que permaneceu no estacionamento quando saisse. Outra sugestão seria mostrar o número de vagas disponíveis ao invés do número de vagas ocupadas.

Dessa forma, utilizamos nossos conhecimentos para resolver um problema com aplicação no cotidiano, aperfeiçoamos nossos conhecimentos em Verilog e aprendemos a estruturar um problema proposto.

9. **Referências Bibliográficas**

MOTA, Hilton O. *Modelamento de circuitos lógicos combinacionais no nível de transferência entre registradores.* Slide apresentado na Unidade Curricular de Introdução aos Sistemas Lógicos Digitais, 2011.

MOTA, Hilton O. *Modelamento de circuitos lógicos sequenciais utilizando a HDL Verilog.* Slide apresentado na Unidade Curricular de Introdução aos Sistemas Lógicos Digitais, 2011.

MOTA, Hilton O. *Modelamento de circuitos sequenciais por RTL .* Aula prática 14 apresentada na Unidade Curricular de Introdução aos Sistemas Lógicos Digitais, 2011.