Universidade Federal do Rio Grande do Norte- UFRN
Bacharelado em Ciências e Tecnologia - C&T

Disciplina: DCA0212.0 Circuitos Digitais - Teoria
Professor: Tiago Tavares Leite Barros
Acadêmicos: Igor Sérgio de França Correia
Neuman Fabricio de Oliveira Fernandes
Thiago Theiry de Oliveira

Projeto 1 - marcapasso

RESUMO

Este relatório tem como objetivo implementar um controlador de uma máquina de estados finita responsável pelo funcionamento de um marcapasso atrioventricular, seguindo os conceitos apresentados na disciplina de Circuitos Digitais.

Palavras-chave: Marcapasso**;** máquina de estados finita; lógica combinacional.

1. INTRODUÇÃO

Um marcapasso é um dispositivo eletrônico capaz de regular batimentos cardíacos através de impulsos elétricos quando o número de batimentos em um certo intervalo de tempo está abaixo do normal. Por meio de uma máquina de estados finita pode-se estabelecer o comportamento do dispositivo em questão. Para isso, faz-se necessário capturar o comportamento do controlador e convertê-lo para circuito — definindo uma arquitetura, codificando os estados, convertendo para uma tabela verdade e por fim implementando a combinação lógica.

2. DESENVOLVIMENTO

2.1 Captura do comportamento da máquina de estados finita

Como base para o projeto, foi fornecida a máquina de estados finita que descreve o funcionamento do marcapasso.

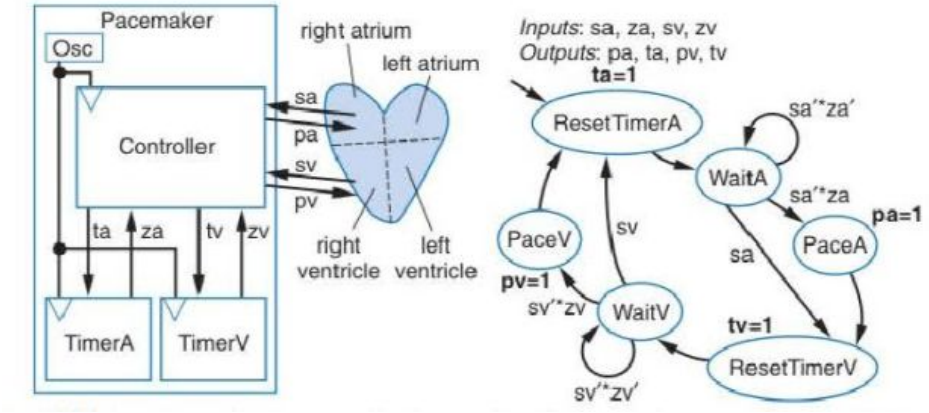


Figura 1: Máquina de estados finita do marcapasso atrioventricular.

Assim, por meio da figura 1, verifica-se que o marcapasso possui dois temporizadores, um para o átrio direito (TimerA) e outro para o ventrículo direito (TimerV). O controlador define inicialmente o TimerA no estado ResetTimerA, e em seguida, aguarda uma contração atrial natural ou o cronômetro

chegar a 0. Se o controlador detecta uma contração atrial natural (sa), então o controlador pula sua estimulação do átrio. Por outro lado, se o TimerA atingir o primeiro, o controlador vai para o estado PaceA, que causa uma contração no átrio ao definir pa = l. Após uma contração atrial (natural ou estimulada), o controlador define TimerV no estado ResetTimerV e, em seguida, espera para uma contração ventricular natural ou para o cronômetro atingir 0. Se uma contração natural ocorrer no ventrículo, o controlador pula a estimulação do mesmo. Por outro lado, se TimerV atinge o primeiro, então o controlador vai para o estado PaceV, o que causa uma contração no ventrículo, definindo pv = 1. Isso define o fim de um ciclo do dispositivo. Ao fim dessa última etapa no estado PaceV, o TimerA é redefinido novamente ao estado ResetTimerA e todo o processo citado anteriormente é repetido.

2.2 Definição da arquitetura

Para que fosse possível representar cada um dos seis estados, foi necessário utilizar 3 bits no registrador de estados, sabido que 2 bits suportam somente quatro estados no total. Portanto, tem-se como entradas a contração atrial natural (sa), contração ventricular natural (sv), indicador do temporizador A (za), indicador do temporizador V (zv), assim como os bits s0, s1 e s2 provenientes do registrador de estados; Como saída, estão presentes a contração atrial artificial (pa), contração ventricular artificial (pv), e os resets de temporizadores A (ta) e V (tv).

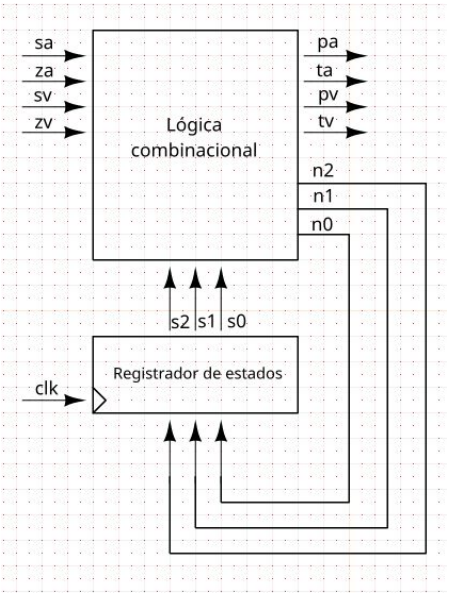


Figura 2: Arquitetura do projeto.

2.3 Codificação dos estados

Utilizando um código de 3 bits único para cada estado, foi utilizada uma contagem binária crescente partindo de 000 até 101, em que ResetTimerA: 000, WaitA: 001, PaceA: 010, ResetTimerV: 011, WaitV: 100 e PaceV: 101.

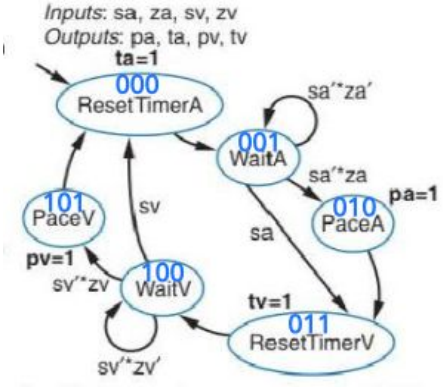


Figura 3: Codificação dos estados

2.4 Conversão para tabela verdade

Ao construir a tabela de modo tradicional, devido às 7 entradas, foram necessárias $2^7 = 128$ linhas para representar todas as possibilidades, dessa forma, tornando custoso o processo de obtenção e simplificação das equações. Assim sendo, foi decidido utilizar a notação *don't care*, que possibilitou reduzir o número de linhas.

states	input							output						
	s2	s1	s0	sa	sv	za	zv	n2	n1	n0	pa	ta	pv	tv
ResetTimerA	0	0	0	x	x	x	x	0	0	1	0	1	0	0
WaitA	0	0	1	0	0	x	x	0	0	1	0	0	0	0
	0	0	1	0	1	x	x	0	1	0	0	0	0	0
	0	0	1	1	x	x	x	0	1	1	0	0	0	0
PaceA	0	1	0	x	x	x	x	0	1	1	1	0	0	0
ResetTimerV	0	1	1	x	x	x	x	1	0	0	0	0	0	1
WaitV	1	0	0	x	x	0	0	1	0	0	0	0	0	0
	1	0	0	x	x	1	x	0	0	0	0	0	0	0
	1	0	0	x	x	0	1	1	0	1	0	0	0	0
PaceV	1	0	1	x	x	x	x	0	0	0	0	0	1	0

Figura 4: Tabela verdade

2.5 Conversão para equações booleanas

Partindo da tabela verdade, é possível encontrar as equações booleanas que descrevem o comportamento de cada saída:

n2= s2'*s1*s0 + s2*s1'*s0'*za'*zv' + s2*s1'*s0'+za'+zv ;
n1= s2'*s1'*s0*sa'*sv + s2'*s1'*s0*sa + s2'*s1*s0' ;
n0= s2'*s1'*s0'+ s2'*s1'*s0*sa'*sv' + s2'*s1'*s0*sa+ s2'*s1*s0'+ s2*s1'*s0'*za'*zv ;
pa= s2'*s1*s0' ;
ta= s2'*s1'*s0' ;
pv= s2*s1'*s0 ;
tv= s2'*s1*s0 ;

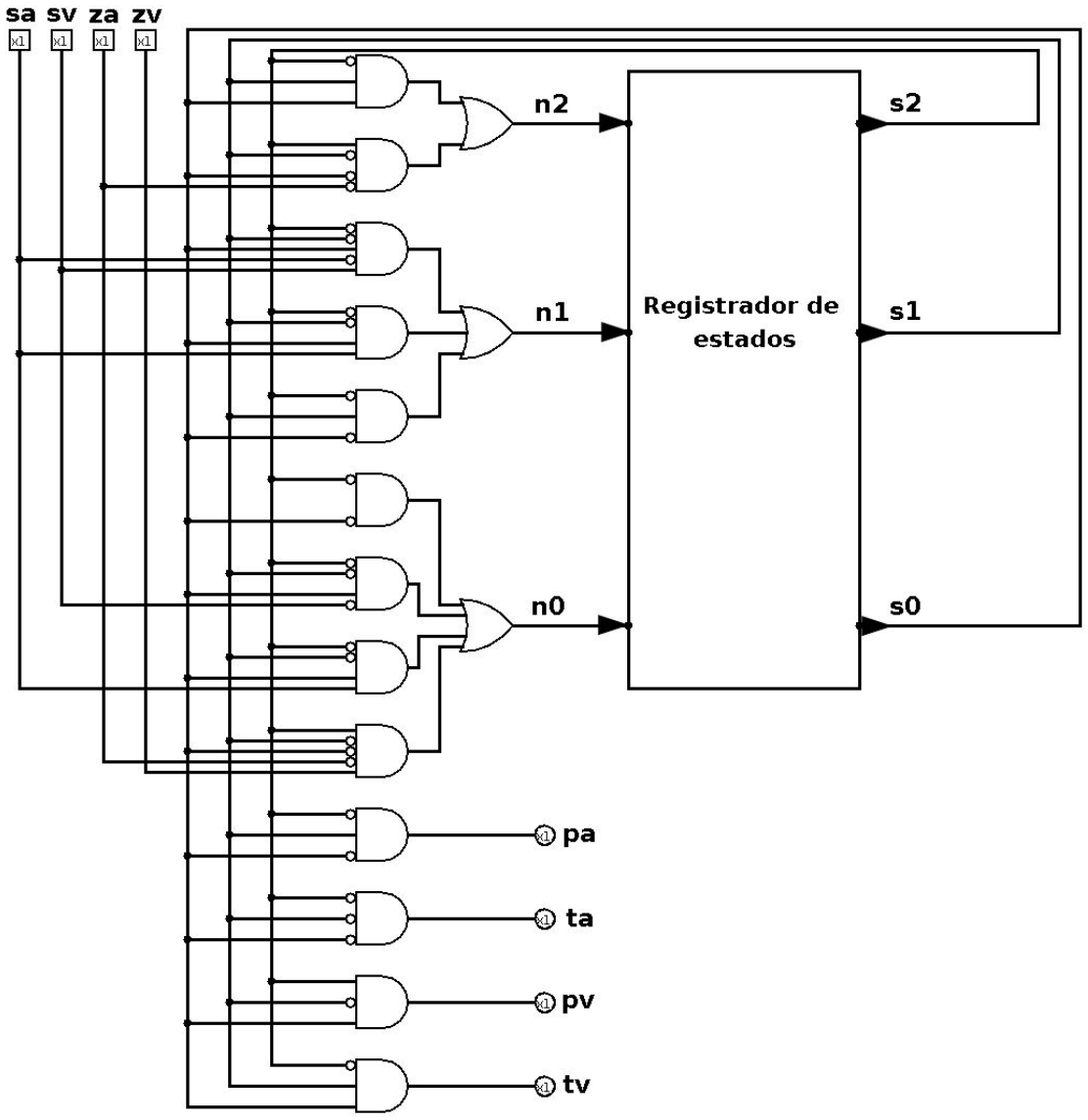
Apenas as equações de n2 e n0 possuem simplificações cabíveis, sendo:

n2= s2'*s1*s0 + s2*s1'*s0'*za'*(zv' + zv) , pela propriedade distributiva;
n2= s2'*s1*s0 + s2*s1'*s0'*za'*1 , pela propriedade do complemento;
n2= s2'*s1*s0 + s2*s1'*s0'*za' , pela propriedade da identidade;

n0= s2'*s0' *(s1' + s1) + s2'*s1'*s0*sa'*sv' + s2'*s1'*s0*sa+ s2*s1'*s0'*za'*zv , pela propriedade distributiva;
n0= s2'*s0'*1 + s2'*s1'*s0*sa'*sv' + s2'*s1'*s0*sa+ s2*s1'*s0'*za'*zv , pela propriedade do complemento;
n0= s2'*s0' + s2'*s1'*s0*sa'*sv' + s2'*s1'*s0*sa+ s2*s1'*s0'*za'*zv , pela propriedade da identidade;
n0= s2'*s0' + s2'*s1'*s0* (sa'*sv' + sa) + s2*s1'*s0'*za'*zv , pela propriedade distributiva;
n0= s2'*s0' + s2'*s1'*s0* (sv' + sa) + s2*s1'*s0'*za'*zv , pela lei da absorção;
n0= s2'*s0' + s2'*s1'*s0*sv' + s2'*s1'*s0*sa+ s2*s1'*s0'*za'*zv , distribuindo;

2.6 Implementação da lógica combinacional

A partir das equações simplificadas e com os recursos presentes no software *Logisim*, foi possível representar o circuito por meio de portas lógicas e de um registrador de estados, utilizando as entradas, e disponibilizando as saídas.



3. DIFICULDADES E CONCLUSÕES

A única dificuldade a ser citada foi durante a abordagem de construção da tabela verdade de forma tradicional. O extenso tamanho das equações booleanas geradas pôde ser simplificado por meio da notação *don't care (já* citada anteriormente). Com as devidas abordagens para cada etapa do projeto e seguindo o conteúdo que foi apresentado em aula, o projeto foi realizado de forma metódica e cuidadosa, analisando e revisando possíveis erros e chegando à implementação da lógica combinacional que, por fim, pode ser usada na máquina de estados.

4. REFERÊNCIAS

1. FRANK, Vahid. Digital Design with RTL Design, VHDL and Verilog. John Wiley & Sons, inc., 2011.