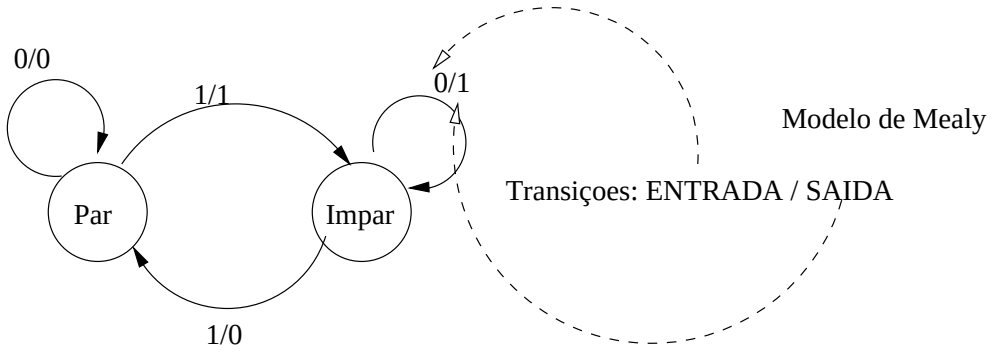
1 Introdução

O objetivo desta apostila é introduzir o projeto de máquinas de estados finitos. Alguns exemplos didáticos serão utilizados: Paridade, sequência, máquina de vendas, sinal de trânsito. As ferramentas Diglog, SIS e Hades serão utilizadas.

2 Detector de Paridade

A paridade de um código binário é definida como: seja N um código binário, N tem paridade igual a 1 se o número de bits iguais a 1 for ímpar, casocontrário a paridade é igual a zero. Por exemplo, seja N= 10011 e M=110011, N tem 3 bits iguais a 1, ou seja tem paridade 1, e M tem 4 bits iguais a 1, ou seja paridade 0. Para descrever um detector de paridade podemos usar o modelo de máquina de estados, ilustrado na figura abaixo:



resultando na tabela verdade:

entrada	estado atual	saída	próximo estado
0	par	0	par
1	par	1	impar
0	impar	1	impar
1	impar	0	par

Se codificarmos o estado par com o código 0 e o estado impar com o código 1, teremos a seguinte tabela:

entrada - I	estado atual - E	saída - S	próximo estado -PE
0	0 (par)	0	0 (par)
1	0 (par)	1	1 (ímpar)
0	1 (impar)	1	1 (ímpar)
1	1 (impar)	0	0 (par)

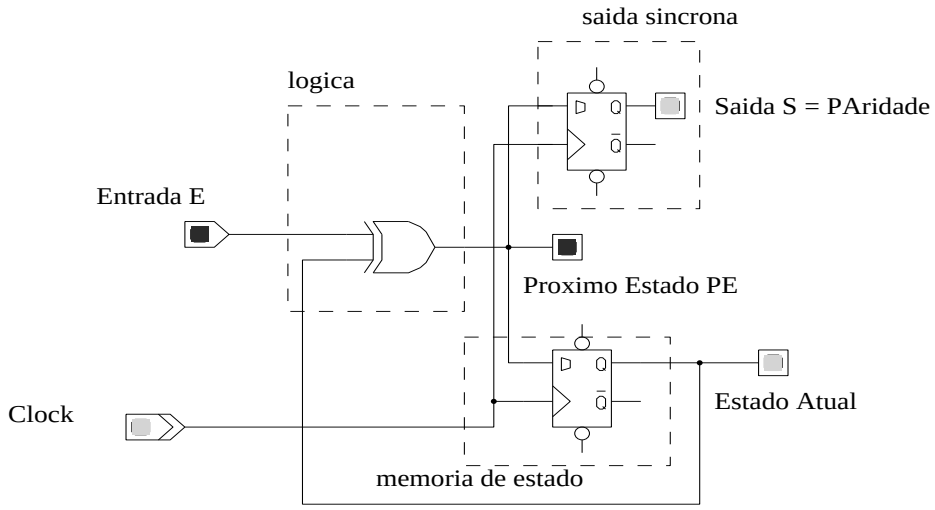
portanto a função do próximo estado PE será:

$$PE = I \oplus E$$

e a função de saída também será:

$$S = I \oplus E$$

Para implementar o circuito no Diglog (arquivo: macro_paridadade.log), podemos propor



onde a parte de lógica implementa as funções de Proximo estado (PE) e saida (S). Um flip-flop tipo D é usado para armazenar o estado atual e outro flip-flop tipo D para armazenar a saída do circuito que é sincronizada com o clock. Para testar o detector de paridade (arquivo: usa_paridade.log), podemos montar um circuito com uma entrada paralela de 8 bits (2 digitos), dois registrador de deslocamento de 4 bits e um gerador de clock. A Figura 1 mostra o circuito de teste. O sinal *carga* serve para iniciar o processo, os bits vão sendo deslocados a cada passo de clock, a máquina de estados processa um bit.

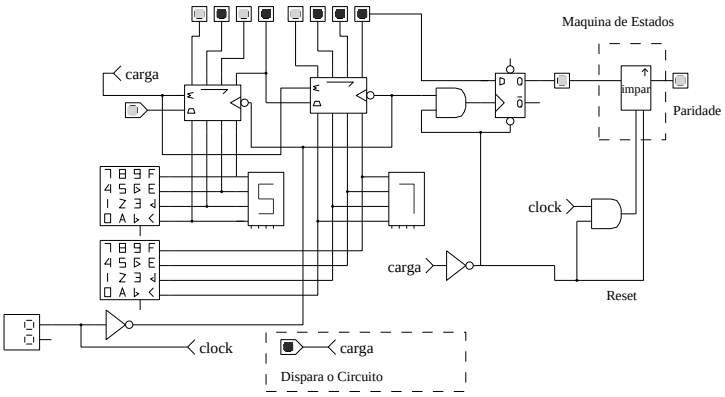


Figura 1: Circuito de Teste para o detector de paridade

O circuito pode ser simulado no software SIS. A descrição da máquina de estados seria:

```
.model
```

```
.inputs I
.outputs S
.start_kiss
.i 1
.o 1
0 par  par 0
1 par  impar  1
0 impar impar  1
1 impar par 0
.end_kiss
```

onde os nomes das variáveis de entrada e saída são listadas após o *.inputs* e *.outputs*, respectivamente. O número de entradas e saídas é especificado pelas diretivas *.i* e *.o*, respectivamente. O nome dos estados pode ser simbólico que o próprio SIS irá atribuir os códigos. A simulação pode ser feita simbólicamente. Este formato de arquivo é conhecido por KISS. Para rodar o SIS basta executar o seguinte roteiro:

```
Primeiro escrever o arquivo formato KISS, depois rodar o SIS, ler o arquivo

> read_kiss paridade.kiss

Para simular, basta digitar simulate e o valor(es) atual(is) da(s) entrada(s)

sis> simulate 0

STG simulation:
Outputs: 0
Next state: par ((null))

Neste exemplo so tem uma entrada. A saida recebe o valor 0 e o próximo estado
é o estado  par.

Para digitar uma sequencia de valores,
execute o comando varias vezes, como por exemplo a sequencia  110:
sis> simulate 1

STG simulation:
Outputs: 1
Next state: impar ((null))

sis> simulate 1

STG simulation:
Outputs: 0
Next state: par ((null))

sis> simulate 0

STG simulation:
Outputs: 0
Next state: par ((null))
```

Após a simulação, a implementação pode ser gerada com os comandos de codificação (state_assign). Estes comandos atribuem códigos binários aos estados de forma a minimizar o circuito a ser gerado. É necessário colocar no PATH o local onde os programas jedi, nova e mustang estão instalados, pois eles são os responsáveis pela codificação.

```
Após rodar

sis> state_assign jedi
Running jedi, written by Bill Lin,  UC Berkeley


para ver a codificacao

sis> print_state
Network state: 1

STG state: par (1)

neste caso, o estado par recebeu o codigo 1.
para ver os resultados no formato slif (equacoes)
----------------------------------------
 WARNING:
....
.model paridade.kiss;
.inputs IN_0;
.outputs OUT_0;
....
LatchOut_v1 = @D([0], NIL);
[0] = IN_0 LatchOut_v1' + IN_0' LatchOut_v1;
OUT_0 = IN_0' LatchOut_v1' + IN_0 LatchOut_v1;
.endmodel paridade.kiss;
----------------------------------------
Onde  as entradas e sadias  foram renomeadas para IN_0 e
OUT_0, o(s) flip-flop(s) de estado e o latchOut_v1 que recebe
a equacao [0] = IN_0 LatchOut_v1' + IN_0' LatchOut_v1;
e a saida OUT_0 recebe IN_0' LatchOut_v1' + IN_0 LatchOut_v1;
Note que a variavel LatchOUT_v1 representa o estado atual, e
a variavel temporaria [0] representa o proximo estado.

Se quiser uma notacao para compacta, pode-se usar o formato eqn

sis> write_eqn
Warning: only combinational portion is being written.
INORDER = IN_0 LatchOut_v1;
OUTORDER = [0] OUT_0;
[0] = IN_0*!LatchOut_v1 + !IN_0*LatchOut_v1;
OUT_0 = !IN_0*!LatchOut_v1 + IN_0*LatchOut_v1;

O significado é semelhante.
```

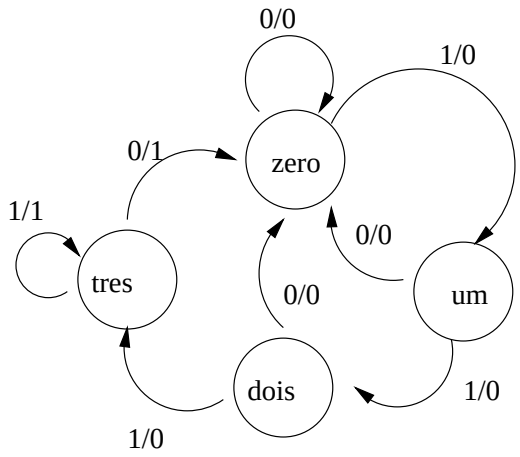
Uma terceira opção de ferramenta é o HADES. O hades possui um editor de máquina de estados, e permite a entrada através do diagrama de estados. O circuito de paridade é ilustrado na Figura 2.

Entretanto o Hades não codifica, deixando a máquina simbólica. Para codificar podemos usar o método manual (tabela -> equações) ou automático (sis), e implementar o circuito.

Figura 2: Detector de paridade no HADES

3 Exemplo detector de tripla 1´s

Nesta seção iremos ver o circuito detector de triplas (exemplo Prof. R. Jacobi). A idéia é projetar uma máquina de estado que detecte uma sequência de 3 bits iguais a 1., ou seja, Saída S(t) = 1 se E(t-1)=E(t-2)=E(t-3)=1. Pode-se propor uma máquina com 4 estados: zero, um,dois e três, que correspondem ao número de bits iguais a 1 dentre os 3 últimos bits recebidos. A máquina de estado terá o seguinte diagrama:



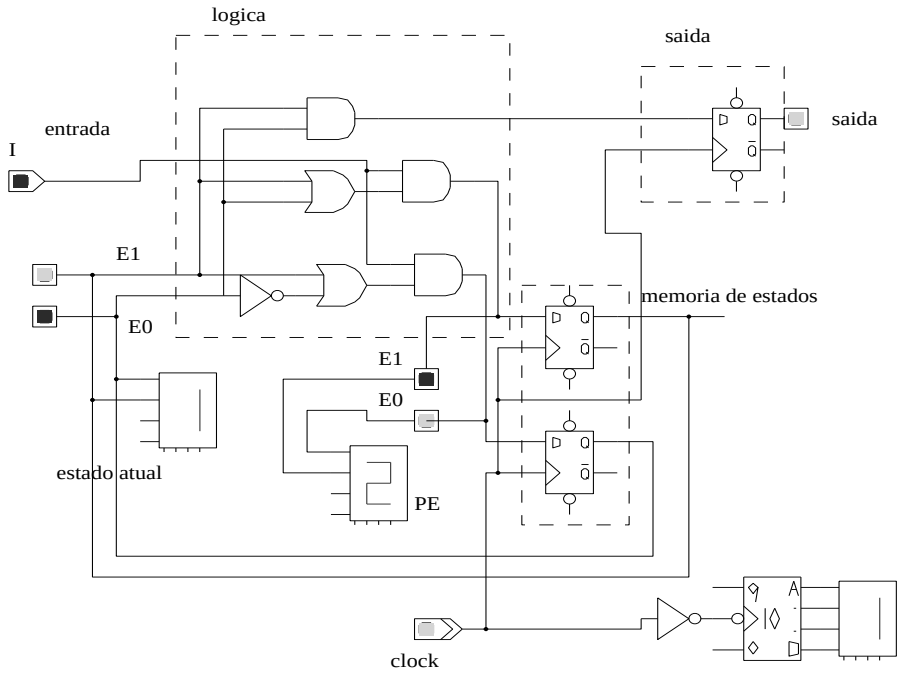
Para codificar 4 estados são necessários 2 bits no mínimo. Podemos propor a codificação natural estado zero = 00, estado um = 01, estado dois = 10 e estado tres = 11. Portanto obteremos a seguinte tabela:

estado atual	entrada (E)	saída (S)	próximo estado (P)
00 (zero)	0	0	00
00 (zero)	1	0	01
01 (um)	0	0	00
01 (um)	1	0	10
10 (dois)	0	0	00
10 (dois)	1	0	11
11 (tres)	0	1	00
11 (tres)	1	1	11

Utilizando mapa de karnaugh obtemos as seguintes expressões para o próximo estado (P) e para a saída (S) em função da entrada (I) e do estado atual (E) :

$S = E_1E_0$,$P_1 = I \cdot E_0 + I \cdot E_1 = I(E_0 + E_1)$,$P_2 = I \cdot E_1 + I \cdot \overline{E_0} = I(E_1 + \overline{E_0})$

que pode ser implementado pelo seguinte circuito (arquivo: tripla.log):



O arquivo simbólico no SIS ficaria:

```
.model
.inputs I
.outputs S
.start_kiss
.i 1
.o 1
0 zero  zero 0
1 zero um 0
0 um zero 0
1 um  dois 0
0 dois zero 0
1 dois tres 0
0 tres zero 1
1 tres tres 1
.end_kiss
```

A saida gerada será no formato slif

```
LatchOut_v1 = @D([0], NIL);
LatchOut_v2 = @D([1], NIL);
[0] = LatchOut_v1' LatchOut_v2' + IN_0';
[1] = LatchOut_v1 LatchOut_v2' + LatchOut_v1' LatchOut_v2 + IN_0';
OUT_0 = LatchOut_v1' LatchOut_v2;
```

Com a codificação de estados

```
.code zero 11
.code um 00
.code dois 10
.code tres 01
```

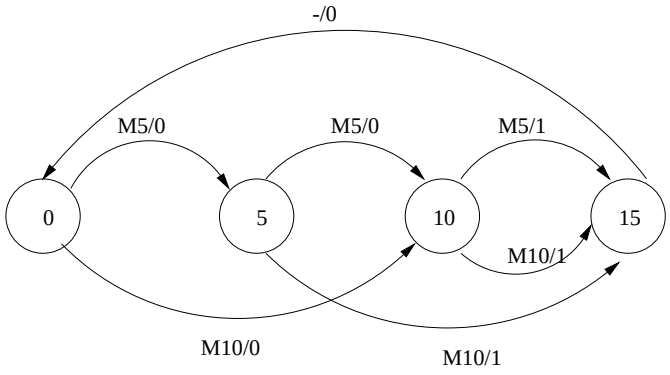
4 Exemplo da máquina de vendas

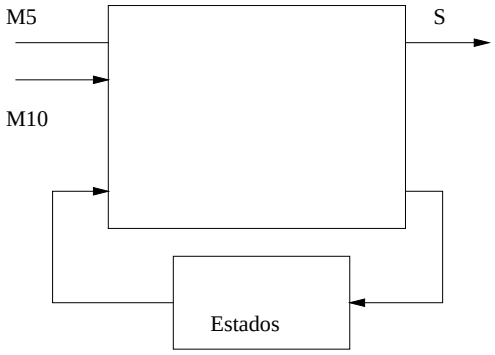
Suponha uma máquina de vendas de refrigerante. Cada refrigerante custa 15 unidades. Existem duas moedas: 5 e 10 unidades. Um sensor M5 indica a deteção de uma moeda de 5 e outro M10 da moeda de 10. A seguir iremos ilustrar passo a passo a metodologia para o projeto de uma máquina de estado. Os passos serão:

1. montar o diagrama de estados
2. montar a tabela do diagrama (com estados simbólicos)
3. codificar os estados, gerando uma tabela verdade
4. projetar as funções lógicas para as funções de saída e transição de estado

4.1 Montar o diagrama de estados

Podemos propor um estado para cada quantia detectada pela máquina: 0,5,10 e 15. Podemos ter apenas um sinal de saída S. Se S=0 o valor é ainda inferior a 15, se S=1 então o valor 15 foi alcançado. Teremos o seguinte diagrama:





4.2 Tabela de Transições Simbólica

O diagrama de estados é uma representação comapcta da máquina de estados, a tabela de transições seria outra forma enumerar todas as transições.

Entradas		estado	Proximo		linha
M5	M10	atual	estado	saída	
0	0	0	0	0	0
1	0	0	5	0	1
0	1	0	10	0	2
1	1	x	x	x	3
0	0	5	5	0	4
1	0	5	10	0	5
0	1	5	15	1	6
0	0	10	10	0	7
1	0	10	15	1	8
0	1	10	15	1	9
x	x	15	0	0	10

Na linha 3 podemos observar que os detectores M5 e M10 não podem ser acionados ao mesmo tempo, portanto é uma condição de entrada que nunca acontecera. E na linha 10, quando a máquina está no estado 15, não importa a entrada gerada pelos detectores, a máquina volta ao estado 0. Outro detalhe, a máquina não devolve troco.

4.3 Codificação dos estados e tabela verdade

Para implementar o circuito é necessário atribuir códigos binários aos estados. Primeiro é necessário determinar quantos estados existem. No nosso exemplo são 4 estados. Para codificar n estados são necessários no mínimo $\log_2 n$ bits. No nosso exemplo 2 bits são suficientes. Determinar qual é a melhor codificação é um problema complexo, tende que se tentar todas as possibilidades. Entretanto, isso pode ser caro (muito calculo) e muitas vezes inviável de se calcular para máquinas com muitos estados. Podemos adotar uma codificação simples ou utilizar um software para resolver o problema.

Um solução rápida seria atribuir os códigos:

código	estado
00	0
01	5
10	10
11	15

Portanto a tabela verdade gerada a partir da tabela de transições simbólicas seria:

Variáveis de entrada				Funções de Saída			linha
entradas		estado atual		proximo estado		saída	
m10	m5	e1	e0	pe1	pe0	s	
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	1
0	0	1	0	1	0	0	2
x	x	1	1	0	0	0	3
0	1	0	0	0	1	0	4
0	1	0	1	1	0	0	5
0	1	1	0	1	1	1	6
1	0	0	0	1	0	0	7
1	0	0	1	1	1	1	8
1	0	1	0	1	1	1	9
1	1	x	x	x	x	x	10

Os detectores são representados por M5 e M10. O estado atual pelas variáveis e_1, e_0. Portanto temos 4 variáveis que são entradas para a máquina de estados, 2 dos detectores e duas da codificação dos estados. Como funções de saída para a máquina temos as funções de próximo estado, codificando pe_1, pe_0e a saída do circuito S que libera o refrigerante.

4.4 Implementação

Para implementar o circuito pode-se usar diversos métodos, como por exemplo: propor um circuito na forma soma de produtos usando mapa de karnaugh. Iremos utilizar o programa espressoaque minimiza de forma eficiente uma soma de produtos. O formato de entrada do arquivo será :

```
.i 4

.o 3

.ilb m10 m5 e1 e0

.ob pe1 pe0 s

0000 000
```

```
0001 010

0010 100

--11 000

0100 010

0101 100

0110 111

1000 100

1001 111

1010 111

11-- ---

.e
```

A saída do programa (usando o comando espresso do SIS) gera uma soma de produtos com 34 literais. Podemos otimizar o circuito transformando em uma forma multinível através dos script de otimização algebrica e booleana : script.algebraic e script.boolean. Após executar o script obtemos uma forma com 19 literais:

$$pe_1 = m_5 \cdot a + e_1\overline{e_0} + m_{10}\overline{e_1}$$

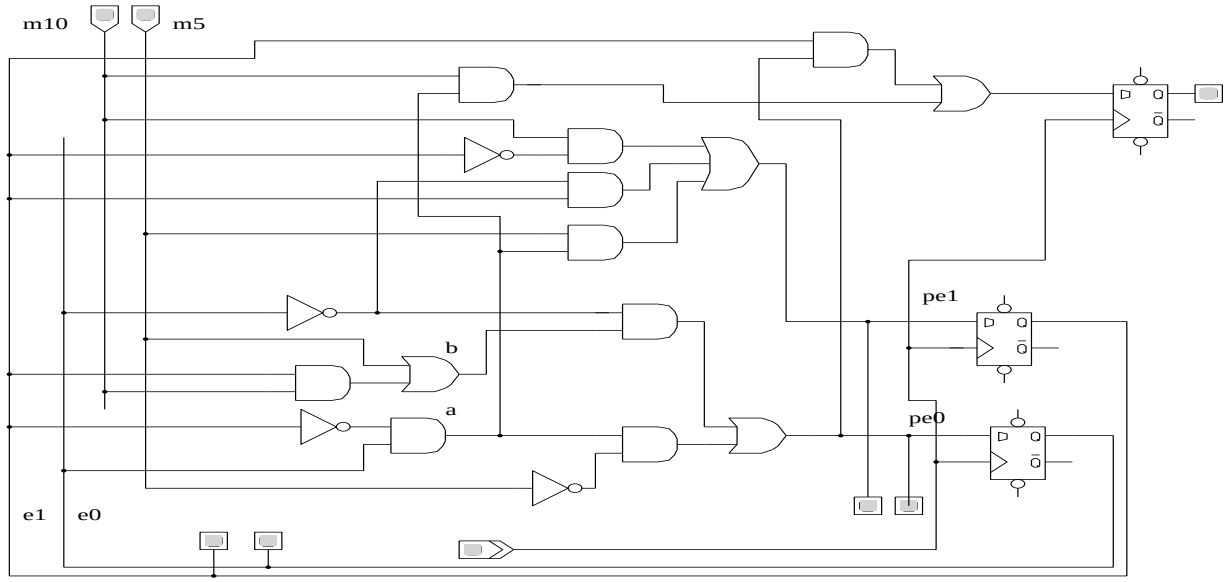
$$pe_0 = \overline{e_0}b + \overline{m_5}a$$

$$s = m_{10}a + e_1 pe_0$$

$$a = \overline{e_1}e_0$$

$$b = m_{10}e_1 + m_5$$

Ou descrito através de porta lógica teremos (arquivo: vendapla.log):



5 Exemplo Sinal de Trânsito

Suponha um cruzamento de uma estrada movimentada com um pequena estrada rural com pouco movimento. A estrada principal tem a preferência e o sinal de transito fica a maior parte do tempo verde. Somente quando um carro aparecer na estrada da fazenda, um detector sinaliza a presença do carro (C), que o sinal da estrada principal poderá fechar. Para temporizar os sinais, dois tempos serão utilizados: tempo curto (TS) e tempo longo (TL). O tempo curto é usado para temporizar a transição do sinal amarelo. O tempo longo é usado para marcar o tempo mínimo que um sinal da fazenda pode ficar aberto (verde) ou que o sinal da estrada principal pode ficar fechado (vermelho).

O funcionamento do sinal será: enquanto não tiver carro ($\overline{C}$) na estrada da fazenda o sinal dela fica fechado. Se tiver um carro (C), porém o tempo longo não terminou ($C \cdot \overline{TL}$), o carro espera até o tempo longo terminar. O sinal principal muda para amarelo e mantém em amarelo durante o tempo curto ($\overline{TC}$). O sinal muda para vermelho e o sinal da fazenda passa para verde, permanecendo enquanto houver carros ou até o tempo longo acabar. A Figura **??** ilustra o diagrama de estados. A saída não é mostrada, mas a cada transição os contadores de tempo (curto e longo, responsáveis pelo tempo de sinal amarelo e vermelho, respectivamente) são disparados.

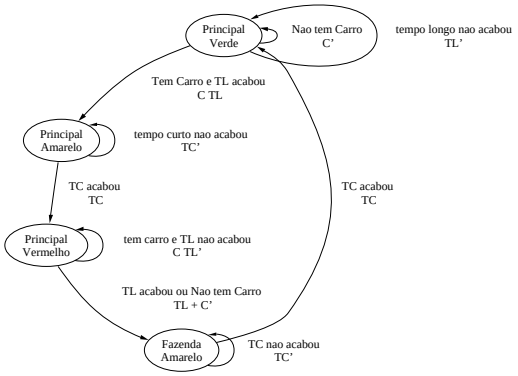


Figura 3: Diagrama de estados do problema do sinal

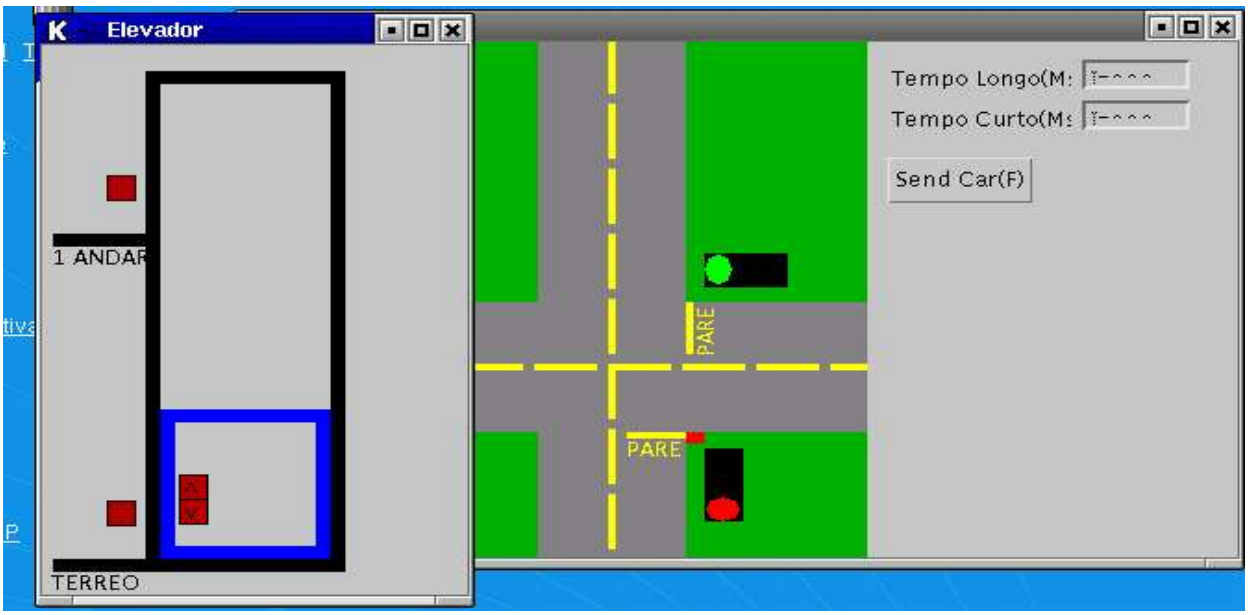
O arquivo no formato KISS poderia ser o seguinte

```
.model
.inputs C TC TL
.outputs DC
.start_kiss
.i 3
.o 1
0--  PVFF PVFF 0
--0  PVFF PVFF 0
1-1  PVFF PAFF 1
-0-  PAFF PAFF 0
-1-  PAFF PFFV 1
1-0  PFFV PFFV 0
0--  PFFV PFFA 1
--1  PFFV PFFA 1
-0-  PFFA PFFA 0
-1-  PFFA PVFF 1
.end_kiss
```

Onde PVFF significa Principal Verde e Fazenda Fechado, o A significa amarelo. C, TC e TL sao carro, tempo curto e tempo longo. E a saida DC significa dispara contador.

O HADES possui um componente para simular o problema do sinal de trânsito, que já inclui o temporizador para tempo longo e curto, além da animação e o sensor de carro.

O problema que também possui um componente para simulação é o elevador de 2 andares (terreo e 1 andar). Como ilustrado em aula, o elevador recebe dois comandos: liga/desliga e sentido (sobe/desce). Como saidas, o elevador fornece os sensores de andar e os botões externos e internos para acioná-lo.



6 Exercícios

1. Implemente o circuito de paridade gerado pelo SIS no Diglog e compare com o circuito feito pelo método manual. Faça o mesmo para o circuito do detector de Tripla.

2. Monte o arquivo de entrada no formato SIS para o problema da máquina de vendas. Gere a codificação no SIS e o circuito. Compare com o método manual. Use o comando "source script.rugged", para minimizar

o circuito após a codificação. Monte o circuito no HADES (sem utilizar o editor de máquina de estados), usando somente portas e flip-flops.

3. Resolva o problema do elevador e do sinal de trânsito usando o SIS, gere o circuito e implemente no HADES para testes dos componentes ELEVADOR e SINALTRANSITO.

4. Modifique o detector de tripla para fazer uma fechadura eletrônica com 3 dígitos. Um teclado serve para entrar com os números. A cada sinal de clock um número é testado, pode-se usar uma tecla enter para ser o clock (acionado após cada dígito). Para a máquina de estados pode-se usar o editor do HADES, não é necessário implementá-la a nível de circuito com o SIS.

5. Modifique o exemplo do relógio digital do HADES para incluir uma máquina de estados e um alarme. Quando o tempo do relógio for igual ao tempo do alarme, uma luz deve piscar. Não é necessário fazer a implementação da máquina de estados a nível de circuito, pode ser feita pelo editor do hades.

6. Monte o circuito de um somador de ponto flutuante na representação IEEE com 8 dígitos de expoente e 23 de mantissa. Considere apenas números positivos. Só é permitido usar um somador/subtrator de 8 bits e um deslocador de 8 bits, além de barramentos, multiplexadores e a máquina de estados. Monte uma máquina de estados para sincornizar o uso dos recursos.