UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
INE5406 – SISTEMAS DIGITAIS

PROJETO PRÁTICO DE SISTEMAS DIGITAIS:

SISTEMA DE UM SEMÁFORO PARA CONTROLE DE UM CRUZAMENTO

Área:

Sistemas Digitais

Alan Djon Lüdke
Matheus Henrique Schaly

Florianópolis
2018/2

Alan Djon Lüdke
Matheus Henrique Schaly

PROJETO PRÁTICO DE SISTEMAS DIGITAIS:

SISTEMA DE UM SEMÁFORO PARA CONTROLE DE UM CRUZAMENTO

Trabalho da disciplina “INE5406 – Sistemas Digitais” apresentado ao curso de Ciências da Computação do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina.

Professor: Rafael Luiz Cancian, Dr. Eng.

Florianópolis
2018/2

Lista de Figuras

Sumário

1 Introdução

Este projeto prático de sistemas digitais tem como objetivo criar um sistema digital síncrono capaz de gerenciar o fluxo de veículos e pedestres em um cruzamento. Tal sistema será reproduzido em VHDL, sintetizado, simulado e prototipado em FPGA da Altera. Esse sistema corresponde a um semáforo para controle de um cruzamento que consiste de uma rua principal no sentido norte-sul, uma rua secundária no sentido leste-oeste e quatro travessias de pedestres simultâneas. O semáforo das ruas principais consiste das cores verde, amarela e vermelha, enquanto o semáforo referente aos pedestres, constitui-se apenas das luzes verde e vermelha.

2 Projeto do Sistema

O projeto do sistema digital inicia com a identificação das entradas e saída e descrição e captura do comportamento do sistema, o que é realizado nas seções seguintes.

2.1 Identificação das entradas

O semáforo proposto possui a interface indicada na figura 2.1. A entrada do sistema é composta de um relógio (“clock”) e um reset assíncrono (“reset”). As saídas do sistema são a representação das luzes do semáforo norte-sul (“*NS*”), semáforo leste-oeste (“*EW*”) e semáforo de pedestres (“*P*”) representadas respectivamente por 3, 3 e 2 bits.

Figura 2.1: Interface do semáforo proposto.

1. O sinal de entrada síncrono *reset* permite reiniciar o sistema, retornando o sistema digital ao seu estado inicial.
2. O sinal de entrada *clock* corresponde ao sinal de relógio para o sincronismo do sistema digital.
3. Os sinais de saídas *NW*, *EW* e *P* permitem ao sistema digital externalizar seu estado atual. Os 3 bits da saída *NS* são assim divididos: O primeiro bit representa a luz verde, o segundo bit a amarela e o terceiro bit a luz vermelha. A mesma ordem de luzes mantêm-se para o semáforo *EW*. Porém, no caso do semáforo dos pedestres, não há a cor amarela. Sendo assim, o primeiro bit e segundo bits correspondem respectivamente a luz verde e vermelha do semáforo dos pedestres.

2.2 Descrição e Captura do Comportamento

O funcionamento fundamental do sistema será descrito a seguir com o auxílio do algoritmo 2.1. O sistema possui um estado inicial que será executado ao iniciar ou resetar o sistema (*reset* – linhas 7 a 11). O estado inicial atribuirá ao semáforo norte-sul a cor verde (linha 9), ao semáforo leste-oeste a cor vermelha (10) e ao semáforo de pedestres também a cor vermelha (linha 11). Após a inicialização, o sistema entrará em loop, atualizando o *time* a cada segundo (linha 38). O sistema permanecerá nas cores atuais durante 45 segundos. Após a duração de 45 segundos (linha 13), o semáforo norte-sul altera sua cor para amarela (linha 14). Após 5 segundos, a variável *time* chega a 50 (linha 16) e o sistema mudará novamente de estado, ou seja, a cor do semáforo norte-sul se tornará vermelha (linha 17). Em seguida há outro aguardo de 5 segundos, mudando a variável *time* para 55 segundos (linha 19) seguido por outra mudança de estado e assim

por diante. Ao chegar em 140 segundos (linha 34) a variável *time* zera (linha 36) e o loop recomeça.

```
 1  #include <iostream>
 2  #include <string>
 3
 4  unsigned short time = 0;
 5
 6  void semaphore(std::string &NS, std::string &EW, std::string &P, unsigned short reset) {
 7      if (reset == 1) {
 8          time = 0;
 9          NS = "green";
10          EW = "red";
11          P = "red";
12      }
13      if (time == 45) {    // after 45 seconds
14          NS = "yellow";  // changes north-south traffic light to yellow
15      }
16      if (time == 50) {   // after 50 seconds
17          NS = "red";      // changes north-south traffic light to red
18      }
19      if (time == 55) {
20          EW = "green";
21      }
22      if (time == 100) {
23          EW = "yellow";
24      }
25      if (time == 105) {
26          EW = "red";
27      }
28      if (time == 110) {
29          P = "green";
30      }
31      if (time == 135) {
32          P = "red";
33      }
34      if (time == 140) {  // after 140 seconds
35          NS = "green";   // changes north-south traffic light to green
36          time = 0;          // resets timer
37      }
38      time ++;               // increases one second
39  }
```

Algoritmo 2.1: Descrição do funcionamento do semáforo.

Entretanto, ao analisar o funcionamento desse algoritmo, podemos perceber que há um pequeno melhoramento que pode ser realizado. Nota-se que os estados onde *time* é igual a 50, *time* é igual a 105 e *time* é igual a 135 são equivalentes, e podem ser rearranjados dentro de apenas uma condição, como é demonstrado no algoritmo 2.2.

```
1  #include <iostream>
2  #include <string>
3
4  unsigned short time = 0;
5
6  void semaphore(std::string &NS, std::string &EW, std::string &P, unsigned short reset) {
7      if (reset == 1) {
8          time = 0;
9          NS = "green";
10         EW = "red";
11         P = "red";
12     }
13     if (time == 45) {    // after 45 seconds
14         NS = "yellow";  // changes north-south traffic light to yellow
15     }
16     if (time == 50 || time == 105 || time == 135) {
17         NS = "red";
18         EW = "red";
19         P = "red";
20     }
21     if (time == 55) {
22         EW = "green";
23     }
24     if (time == 100) {
25         EW = "yellow";
26     }
27     if (time == 110) {
28         P = "green";
29     }
30     if (time == 140) {  // after 140 seconds
31         NS = "green";   // changes north-south traffic light to green
32         time = 0;          // resets timer
33     }
34     time ++;              // increases one second
35 }
```

Algoritmo 2.2: Descrição completa do funcionamento do semáforo com aprimoramento.

O algoritmo 2.2 já possui a vantagem de possuir menos condições de desvio.

Com o intuito de obter absoluta segurança no algoritmo que descreve o funcionamento do sistema digital, o mesmo foi executado na linguagem de programação C++, e uma série de testes foi aplicado sobre o algoritmo. Tal algoritmo é demonstrado no algoritmo 2.3.

A saída desse algoritmo de teste demonstra o real funcionamento do semáforo. O algoritmo simula 420 pulsos de clock (linhas 45 a 60), assim como o *input* reset nas linhas 46 a 51 e 54 a 59. Portanto, o algoritmo funciona como o esperado e é capaz de simular o funcionamento do semáforo em diferentes situações.

Ao analisar o algoritmo 2.2 podemos perceber que há a necessidade da existência de registradores de dados para o armazenamento de variáveis internas, podemos também deduzir as operações aritméticas, lógicas, relacionais e de transferência que são necessárias para o funcionamento do sistema digital, assim como o fluxo de controle de execução do comportamento do semáforo e a dependência da variável *time* com as condições de desvio.

```
1  void simulate() {
2      std::string NS_light = "green", EW_light = "red", P_light = "red"; // initialization
3      unsigned short reset = 0;
4      for (unsigned int i = 1; i <= 420; i++) {  // simulates 420 clock pulses
5          if (i == 65) {                         // simulates 5 consecutive resets after 65 clock pulses
6              reset = 1;
7          }
8          if (i == 200) {                        // simulates one reset after 200 clock pulses
9              reset = 1;
10         }
11         semaphore(NS_light, EW_light, P_light, reset);
12         std::cout << "Pulse " << i << ": " << NS_light << ", " << EW_light << ", " << P_light << ", " << reset << std::endl;
13         if (i == 70) {
14             reset = 0;
15         }
16         if (i == 200) {
17             reset = 0;
18         }
19     }
20 }
21 }
22
23 int main() {
24     simulate();
25     return 0;
26 }
```

Algoritmo 2.3: Programa de teste do algoritmo que descreve o comportamento do sistema digital para o funcionamento do semáforo.

Portanto, com base no algoritmo descrito acima, podemos identificar os componentes abaixo que constituirão o bloco de controle e o bloco operativo do sistema digital proposto.

I. **Registradores de Dados.** Ao observar as variáveis internas podemos extrair os registradores necessários:
   (1) time: 8 bits, inteiro sem sinal.
   (2) NS: 3 bits, logic vector.
   (3) EW: 3 bits, logic vector.
   (4) P: 2 bits, logic vector.
   Os registradores de dados serão alocados ao bloco operativo.

II. **Operações.** As operações aritméticas, lógicas são extraídas ao analisar as operações feitas pelo algoritmo e também seus operandos. No sistema digital proposto, estas são as operações necessárias:
   (1) Comparação com zero: Utilizada para realizar a funcionalidade da linha 32.
   (2) Comparação com inteiro sem sinal: Necessária para efetuar a funcionalidade das linhas 7, 13, 16, 21, 24, 27 e 30.
   (3) Atribuição: Todos os registradores terão valores alterados em certos momentos (linhas 8, 9, 10, 11, 14, 17, 18, 19, 22, 25, 28, 31 e 32). Significando que a carga de tais registradores devem ser monitoradas.
   (4) Adição inteira sem sinal: Utilizada para realizar a funcionalidade da linha 34.

III. **Fluxo de Controle.** A presença de estruturas de controle de fluxo podem ser notadas diretamente da observação do algoritmo. Nesse caso o uso de *if-else* e de inicialização de variáveis:
   (1) Inicialização de variáveis: Necessária para inicializar o sistema digital, e também corresponde ao controle do estado de reset.

(2) Desvios condicionais: Necessários para alterar o estado das lâmpadas de distintos semáforos. Podem ser vistos nas linhas 7, 13, 16, 21, 24, 27 e 30.

IV. Dependência entre Dados: As seguintes operações foram identificadas como não possuindo dependência entre si:

(1) Linhas 4 e 6: As atribuições das variáveis *time*, *NS*, *EW*, *P* e *reset* não possuem dependências entre si.

A partir desses elementos retirados do algoritmo que representa o funcionamento do sistema digital, podemos visualizar o comportamento do mesmo em uma máquina de estados de alto nível (FSMD). A figura 2.2 demonstra a FSMD do sistema digital proposto.

2.3 Projeto do Bloco Operativo

Tendo como base as operações realizadas no algoritmo, assim como os registradores de dados, podemos iniciar o projeto do bloco operativo. Na figura 2.2 abaixo será demonstrado o FSMD do sistema digital proposto no algoritmo 2.1, ou seja, do algoritmo anterior ao aprimoramento.

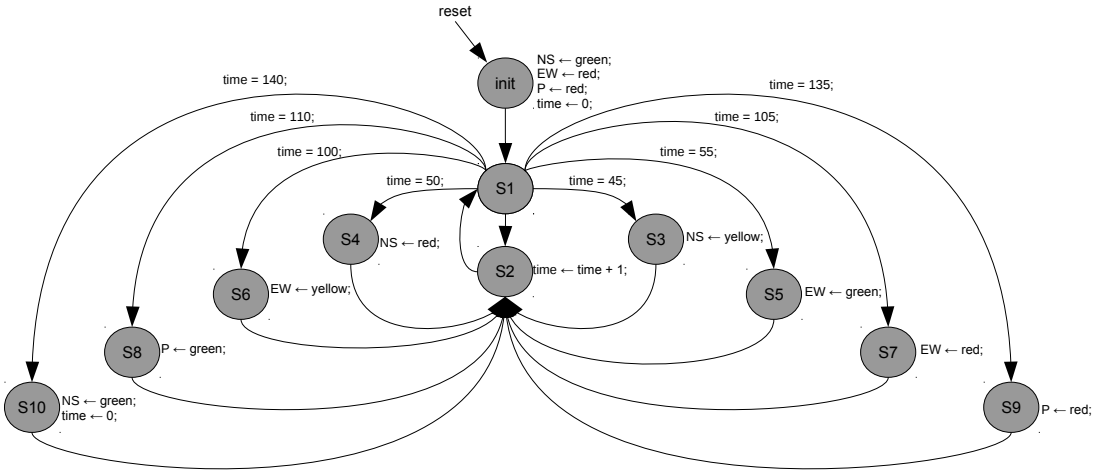

Figura 2.2: FSMD do sistema digital - A representação do comportamento da FSMD foi retirada da representação do sistema digital em algoritmo, já considerando registradores, operações e fluxos de controle.

Ao compararmos a FSMD da figura 2.2 e a FSMD da figura 2.3, notamos que há uma redução no número de estados de 11 para 9. Sendo assim, ao retirarmos os estados repetidos, diminuímos em 2 o número de estados da FSMD.

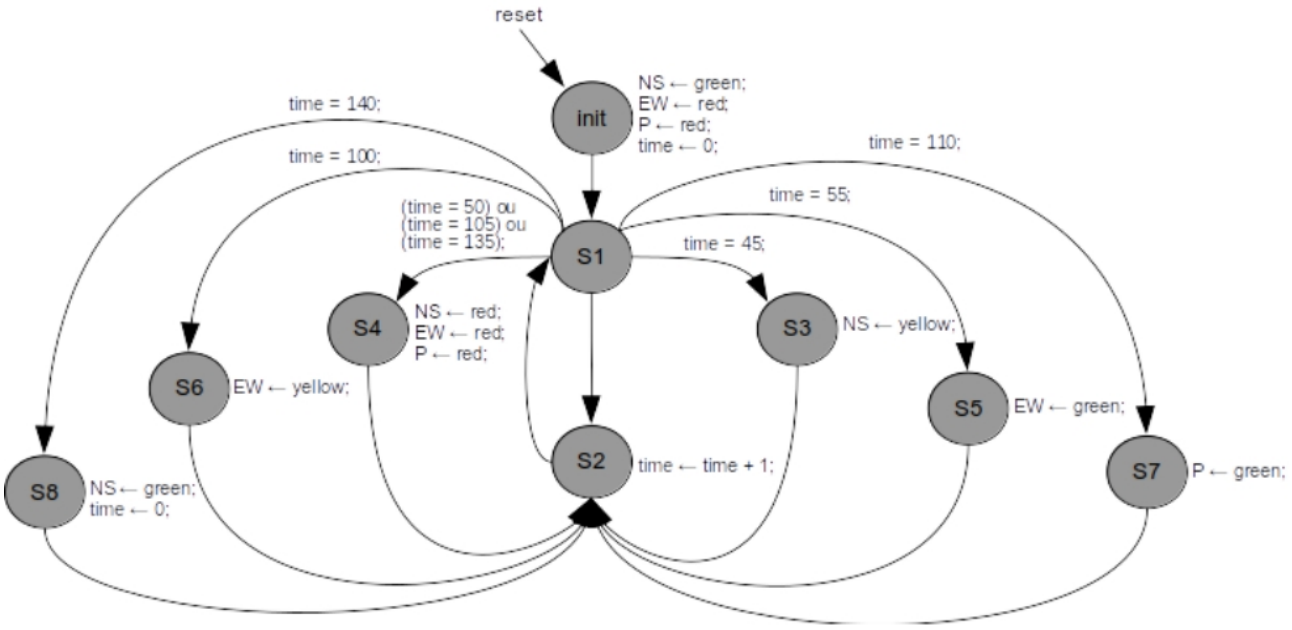

Figura 2.3: FSMD aprimorada do sistema digital proposto – Nota-se a redução de dois estados.

Abaixo é exposto, identificados com base no algoritmo, os circuitos para atribuição de cada um dos registradores. Nas figuras a seguir, os sinais de controle são representados em azul e os sinais de estado (retorno ao bloco de controle) são mostrados em vermelho.

(1) **time** = **time + 1** (linha 34). Figura 2.4.

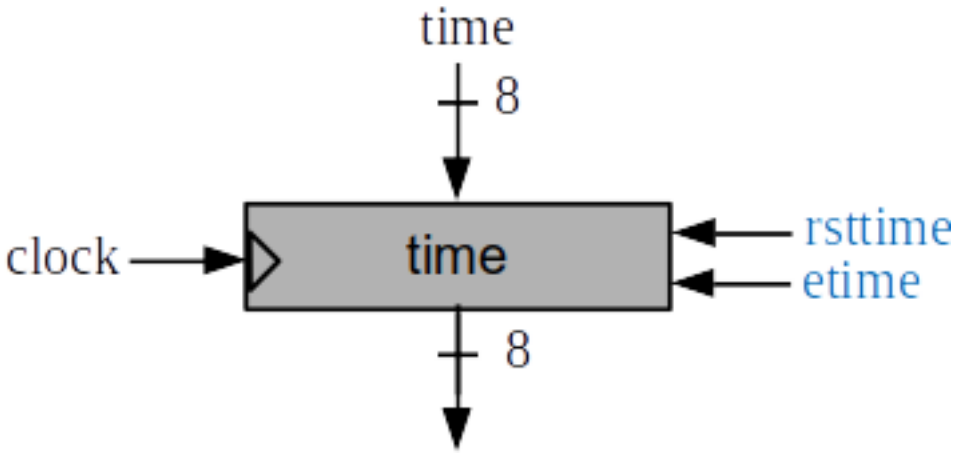

Figura 2.4: Circuito para atribuição do registrador time.

(2) **NS** = "**100" (**linha 9); **NS = "010"** (linha 14); **NS = "001"** (linha 17); **NS = "100"** (linha 31). Figura 2.5.

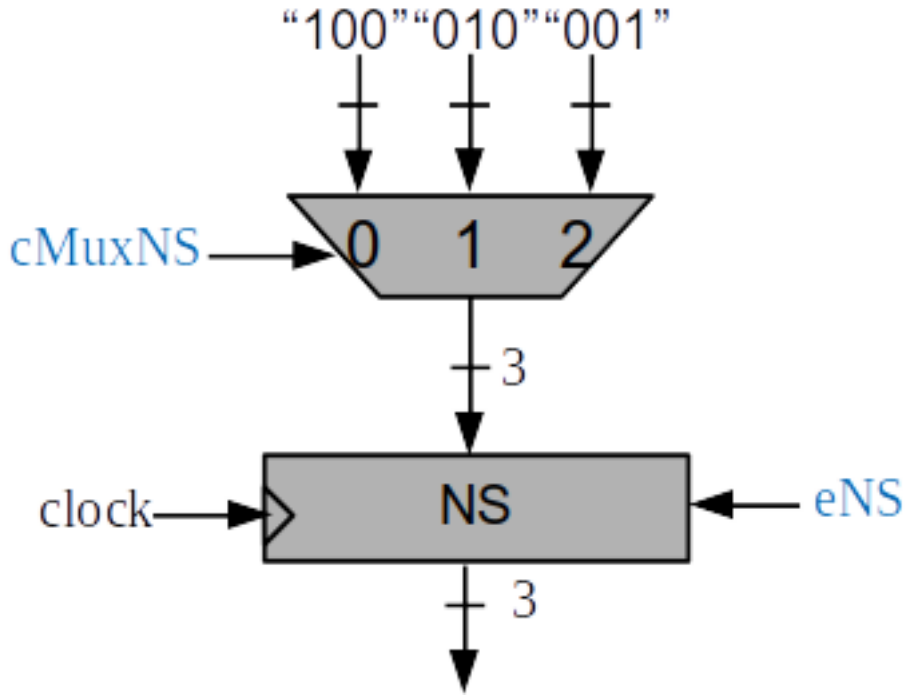

Figura 2.5: Circuito para atribuição do registrador NS.

(3) **EW** = “**001”** (linha 10); **EW = “001”** (linha 18); **EW = “100”** (linha 22); **EW = “010”**; (linha 25) Figura 2.6.

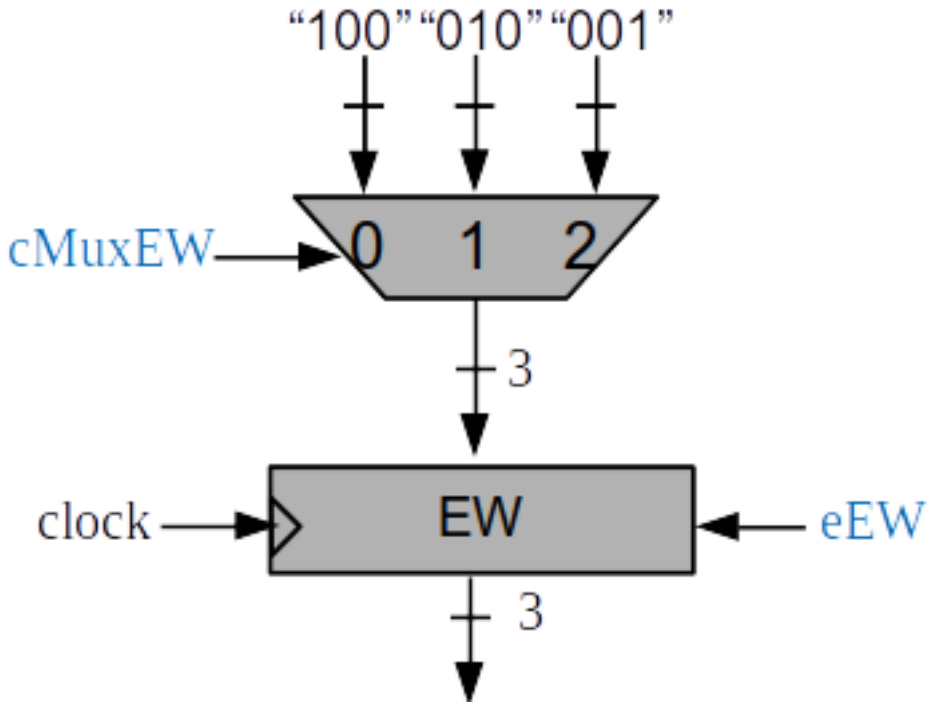

Figura 2.6: Circuito para atribuição do registrador EW.

(4) **P** = **“01”** (linha 11); **P = “01”** (linha 19); **P = “10”** (linha 28). Figura 2.7.

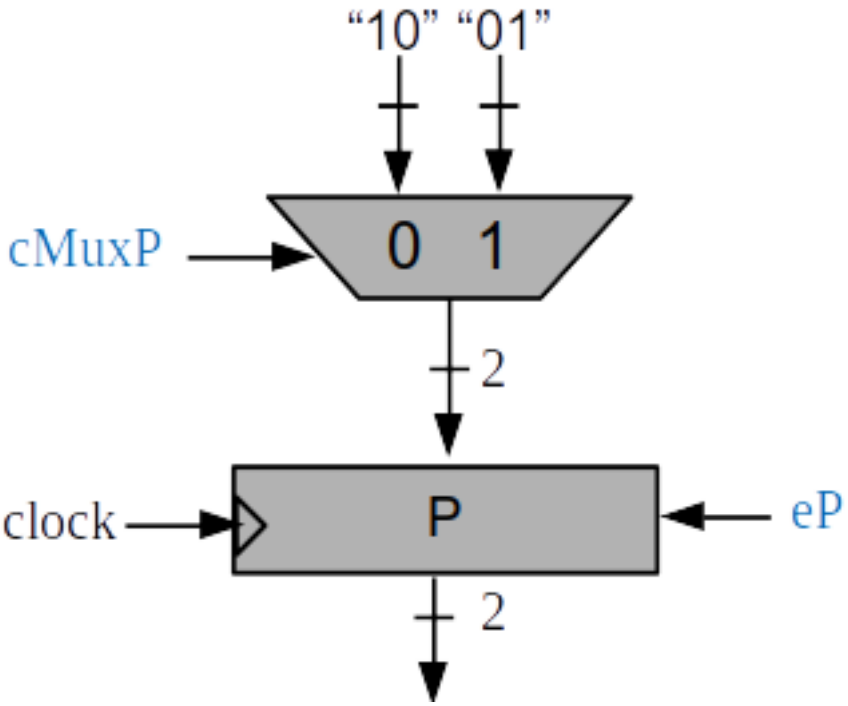

Figura 2.7: Circuito para atribuição do registrador P.

Além disso é necessário a utilização de circuitos que serão usados para controlar o fluxo da execução. Tendo como base o fluxo de controle analisado anteriormente, nota-se que os estados que serão utilizados nos controles de fluxo precisam ser efetuados:

(1) reset == 1 (Reseta o circuito – linha 7).
(2) time == 45 (Mudança de estado – linha 13).
(3) time == 50 || time == 105 || time == 135 (Mudança de estado – linha 16).
(4) time == 55 (Mudança de estado – linha 21).
(5) time == 100 (Mudança de estado – linha 24).
(6) time == 110 (Mudança de estado – linha 27).
(7) time == 140 (Mudança de estado – linha 30).

Os circuitos descritos acima são conectados para gerar o bloco operativo. O projeto do bloco operativo pode ser visto na figura 2.8.

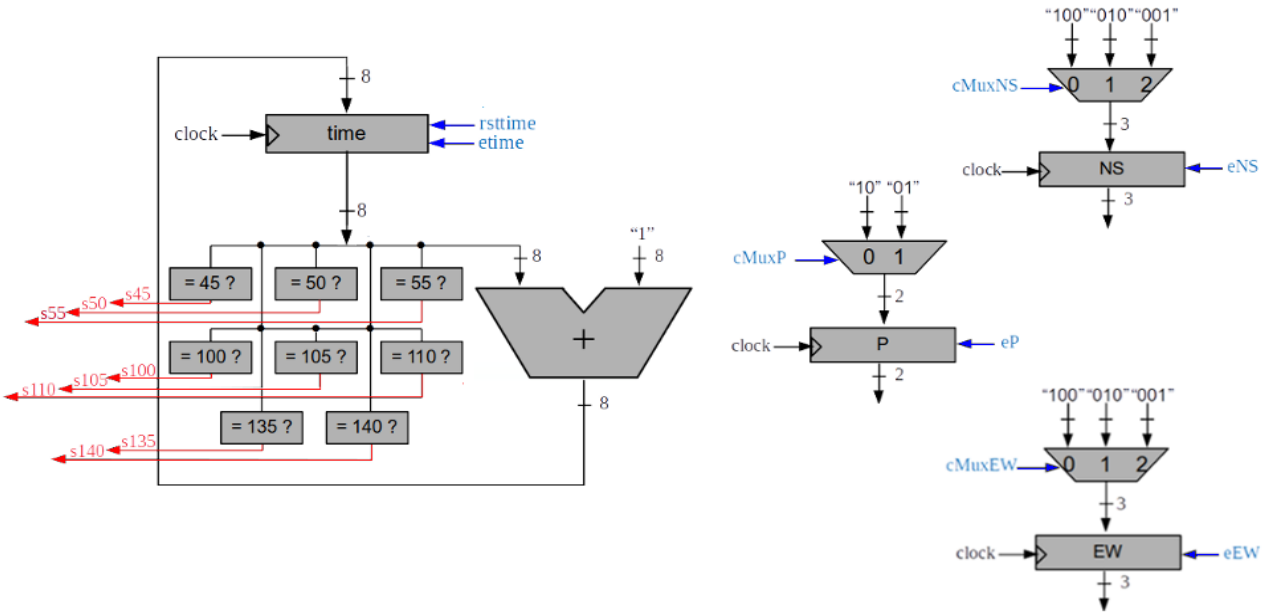

Figura 2.8: Bloco Operativo do sistema digital - Todas as operações sobre os dados, assim como a geração de estados para o controle foram considerados.

Ao observar o projeto do bloco operativo, podemos listar os sinais de controle (vindos do bloco de controle que será analisado adiante) e os sinais de status (que serão encaminhados para o bloco de controle). São sinais de controle:

(1) *rsttime*: Faz com que o registrador *time* tenha o valor 0.
(2) *etime*: Controla a carga do registrador *time* (0: Mantém, 1: Carrega).
(3) *eNS*: Controla a carga do registrador *NS* (0: Mantém, 1: Carrega).
(4) *eEW*: Controla a carga do registrador *EW* (0: Mantém, 1: Carrega).
(5) *eP*: Controla a carga do registrador *P* (0: Mantém, 1: Carrega).
(6) *cMuxNS*: Controla o dado a ser carregado em *NS* (0: “100”, 1: “010”, 2: “001”).
(7) *cMuxEW*: Controla o dado a ser carregado em *EW* (0: “100”, 1: “010”: 2: “001”).
(8) *cMuxP*: Controla o dado a ser carregado em *P* (0: “10”, 1: “01”).

São sinais de status:

(1) *s45*: Indica que o registrador *time* é igual a 45.
(2) *s50*: Indica que o registrador *time* é igual a 50.
(3) *s55*: Indica que o registrador *time* é igual a 55.
(4) *s100*: Indica que o registrador *time* é igual a 100.
(5) *s105*: Indica que o registrador *time* é igual a 105.
(6) *s110*: Indica que o registrador *time* é igual a 110.
(7) *s135*: Indica que o registrador *time* é igual a 135.
(8) *s140*: Indica que o registrador *time* é igual a 140.

2.4 Projeto do Bloco de Controle

Com base na definição dos passos anteriores (bloco operativo, algoritmo e FSMD) que capturam o comportamento do sistema, podemos projetar a FSM que considera o projeto do bloco operativo. Os nomes fornecidos aos sinais de controle e de estado e descreve o funcionamento do bloco de controle. Essa FSM é apresentada na figura 2.9.

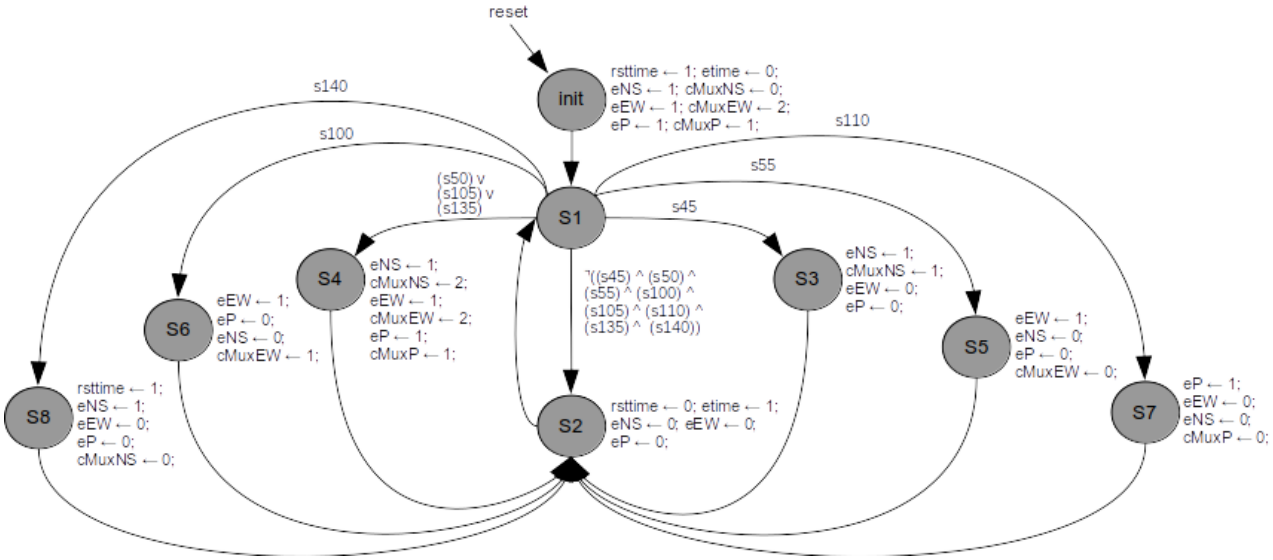

Figura 2.9: Bloco de controle, projetado com uma FSM - Essa FSM representa o controle do sistema, que recebe sinais de controle externos e sinais de status do bloco operativo, e gera saídas de controle e sinais de comando para o bloco operativo.

2.5 Projeto da Unidade Aritmética

Após analisar os projetos do bloco operativo e do bloco de controle, podemos realizar o projeto do sistema digital completo (semáforo). O sistema digital completo será constituído por um único bloco de controle e um único bloco operativo. Sendo assim, o projeto completo resulta em apenas agregar ambos os projetos anteriores. A estrutura do semáforo é apresentada na figura 2.10.

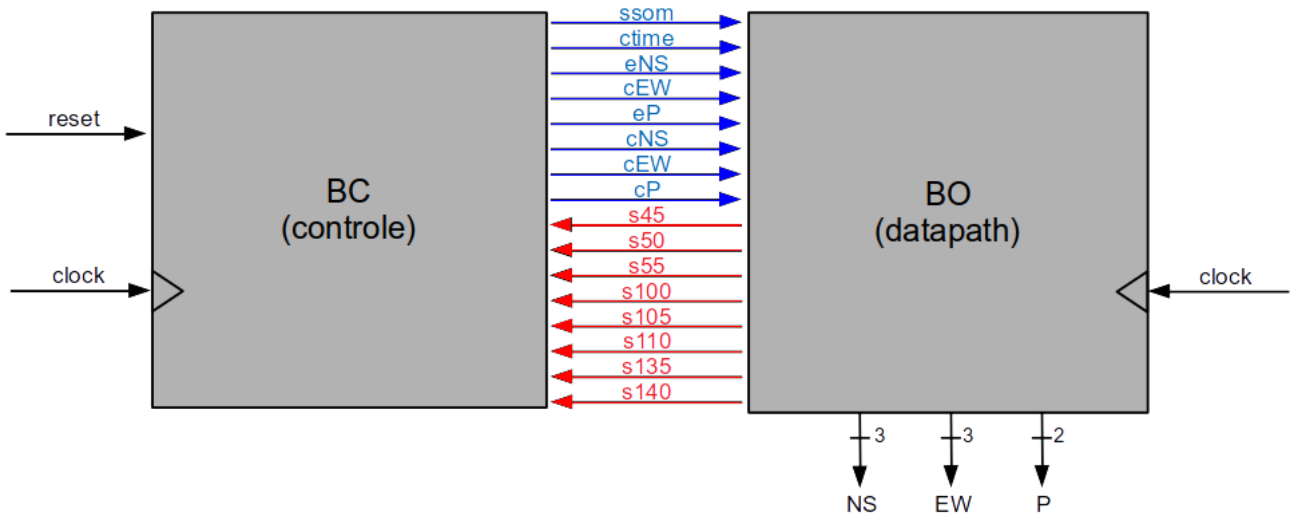

Figure 2.10: Diagrama bloco de controle, bloco operativo – O sistema digital completo proposto pode ser resumido, com base nas seções anteriores, em um diagrama BO/BC.