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) cktimer: 26 bits, inteiro sem sinal.
   (3) NS: 3 bits, logic vector.
   (4) EW: 3 bits, logic vector.
   (5) 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.

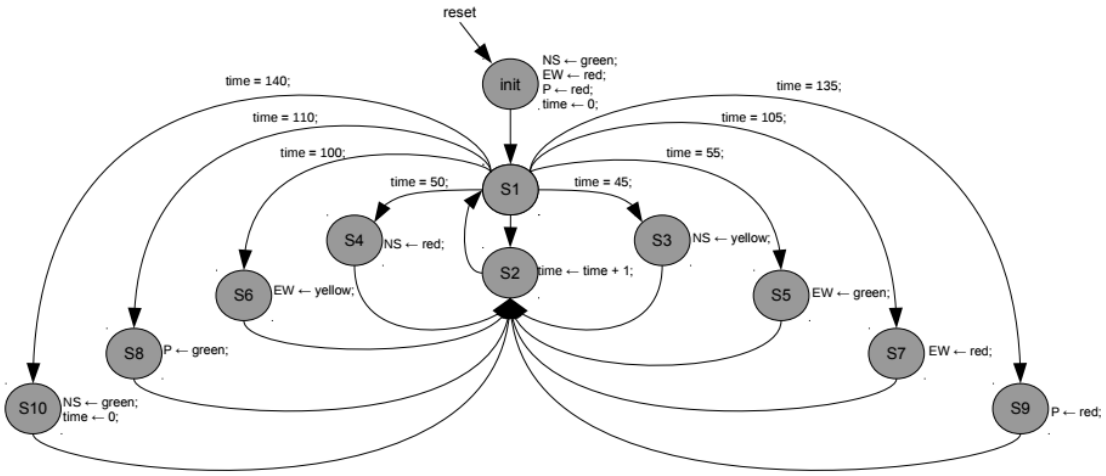

Figura 2.2: FSMD do sistema digital - A representação do comportamento da FSMD foi retirada da representação do sistema digital em algoritmo, já considerando registradores, operações e fluxos de controle.

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.

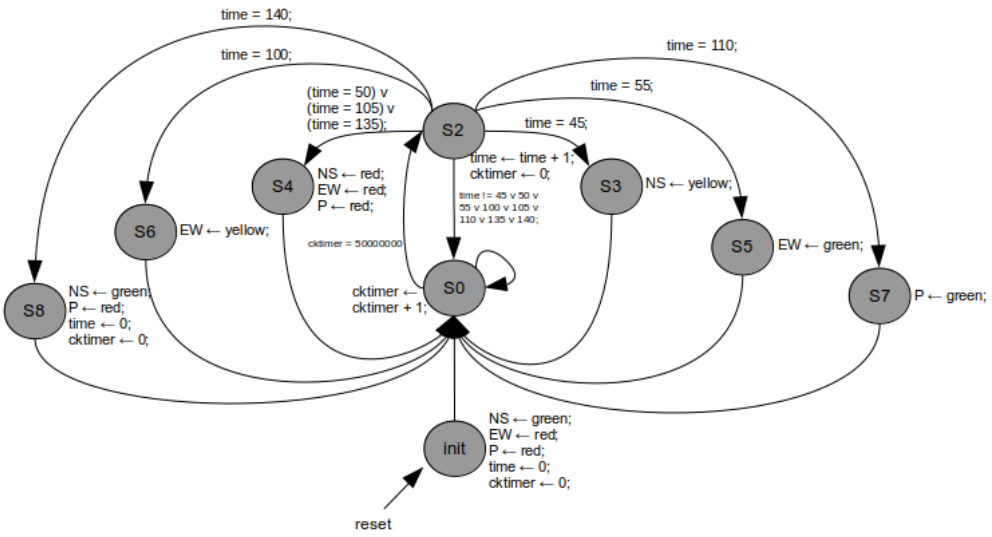

Figura 2.3: FSMD aprimorada do sistema digital proposto.

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.

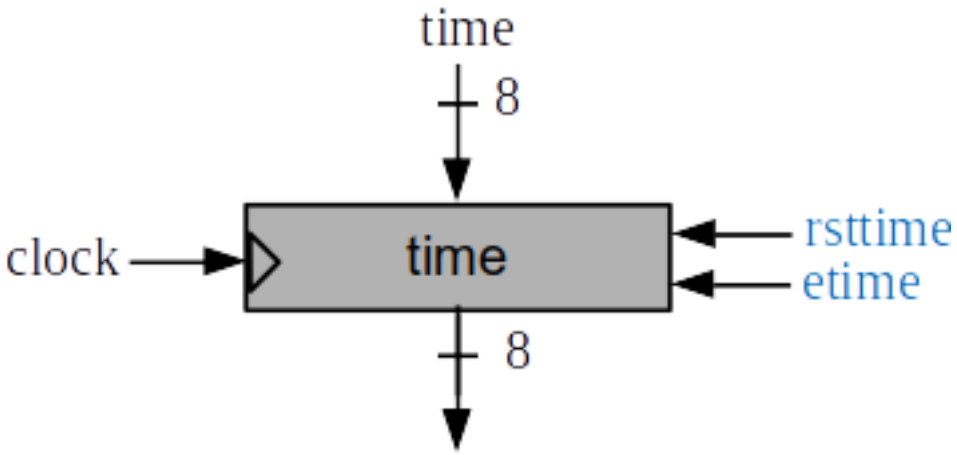

Figura 2.4: Circuito para atribuição do registrador time.

(2) **cktimer.** Será utilizado para contar até um número específico, dependente da frequência da placa utilizada, para que o registrador time valha 1 segundo. Figura 2.5.

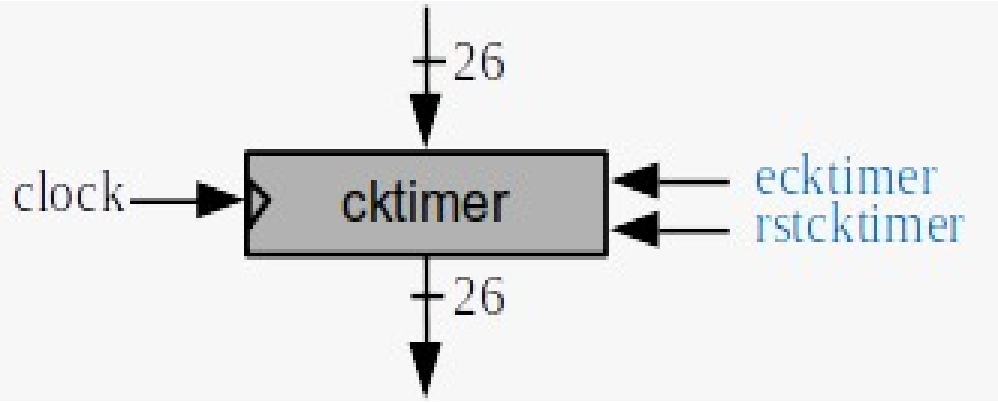

Figura 2.5: Circuito para atribuição do registrador cktimer.

(3) **NS** = "**100" (**linha 9); **NS = "010"** (linha 14); **NS = "001"** (linha 17). Figura 2.6.

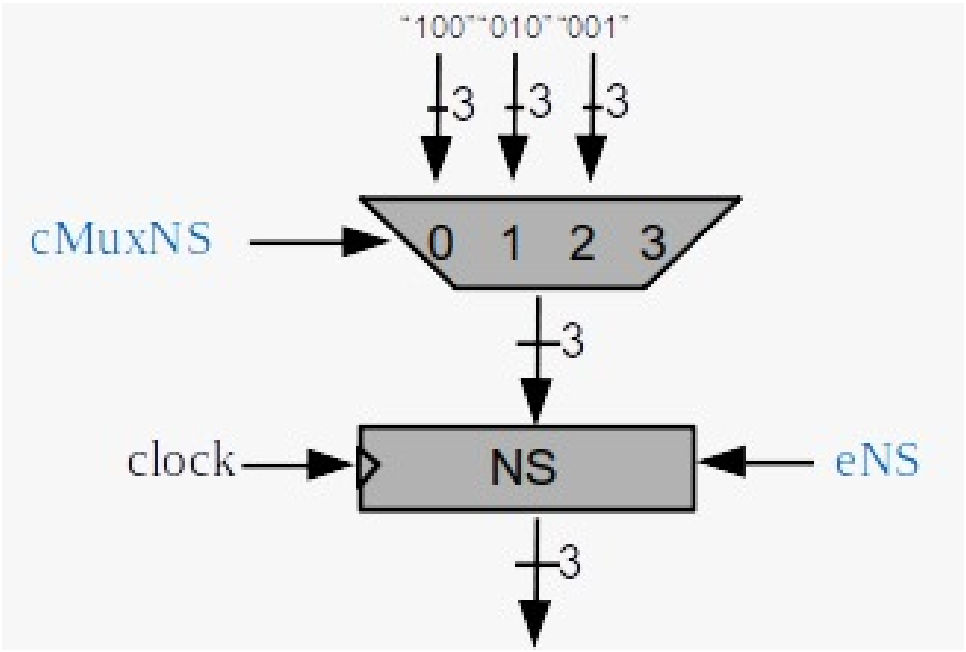

Figura 2.6: Circuito para atribuição do registrador NS.

(4) **W** = "**001"** (linha 10 e 18); **EW = "001"** (linha 18); **EW = "100"** (linha 22); **EW = "010"**; (linha 25) Figura 2.7.

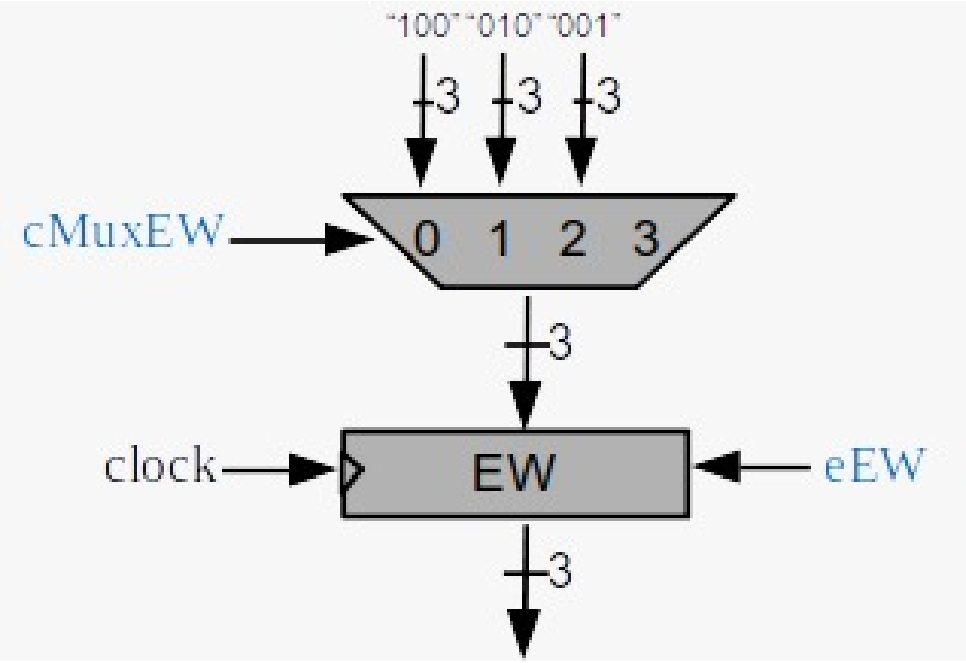

Figura 2.7: Circuito para atribuição do registrador EW.

(5) **P** = **“01”** (linha 11); **P = “01”** (linha 19); **P = “10”** (linha 28). Figura 2.8.

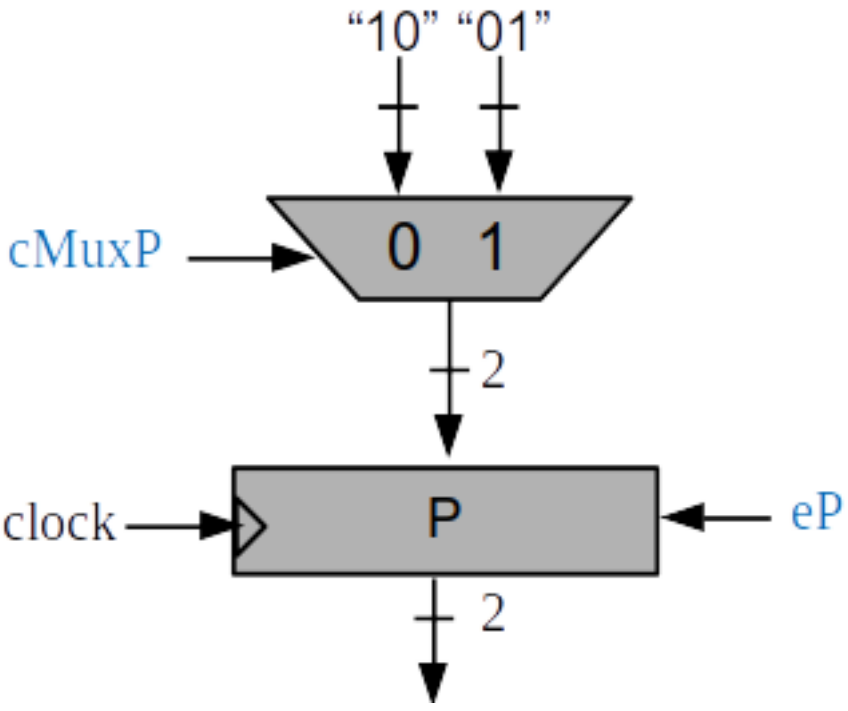

Figura 2.8: Circuito para atribuição do registrador P.

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 2.9.

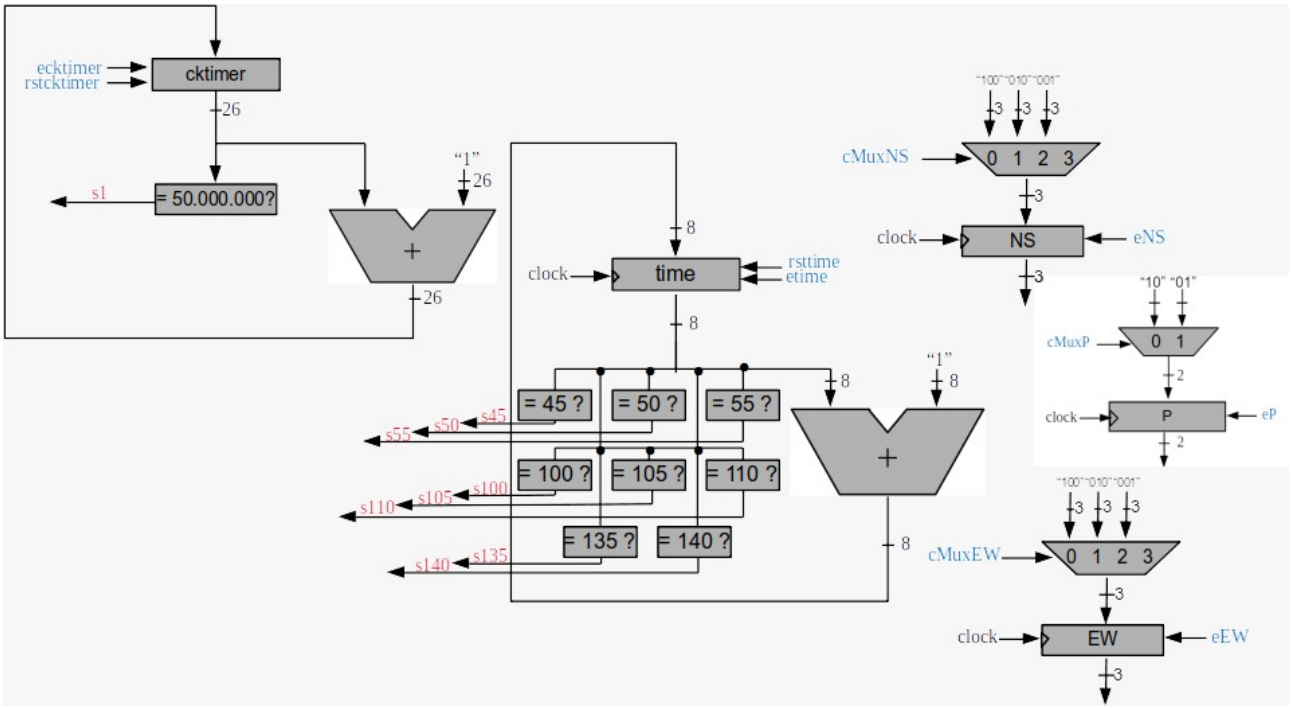

Figura 2.9: Bloco operativo do sistema digital - Todas as operações sobre os dados, assim como a geração de estados para o controle foram considerados.

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

São sinais de status:

(1) *s1*: Indica ao bloco operativo que se passou um tempo determinado (1 segundo).
(2) *s45*: Indica que o registrador *time* é igual a 45.
(3) *s50*: Indica que o registrador *time* é igual a 50.
(4) *s55*: Indica que o registrador *time* é igual a 55.
(5) *s100*: Indica que o registrador *time* é igual a 100.

(6) *s105*: Indica que o registrador *time* é igual a 105.
(7) *s110*: Indica que o registrador *time* é igual a 110.
(8) *s135*: Indica que o registrador *time* é igual a 135.
(9) *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 2.10.

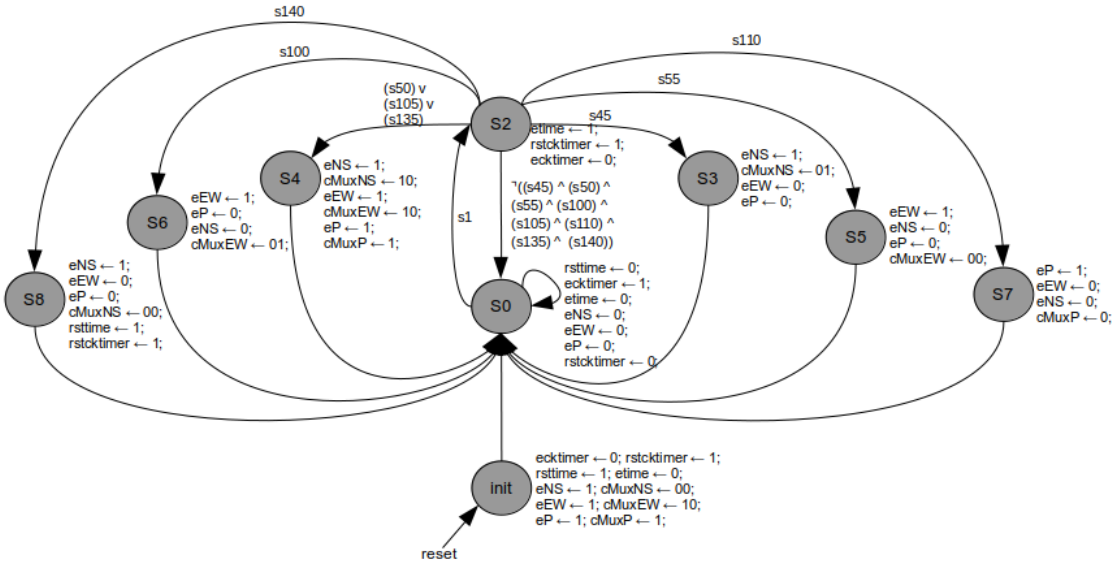

Figura 2.10: Bloco de controle, projetado com uma FSM - Essa FSM representa o controle do sistema, que recebe sinais de controle externos e sinais de status do bloco operativo, e gera saídas de controle e sinais de comando para o bloco operativo.

2.5 Diagrama bloco de controle e bloco operativo

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 2.11.

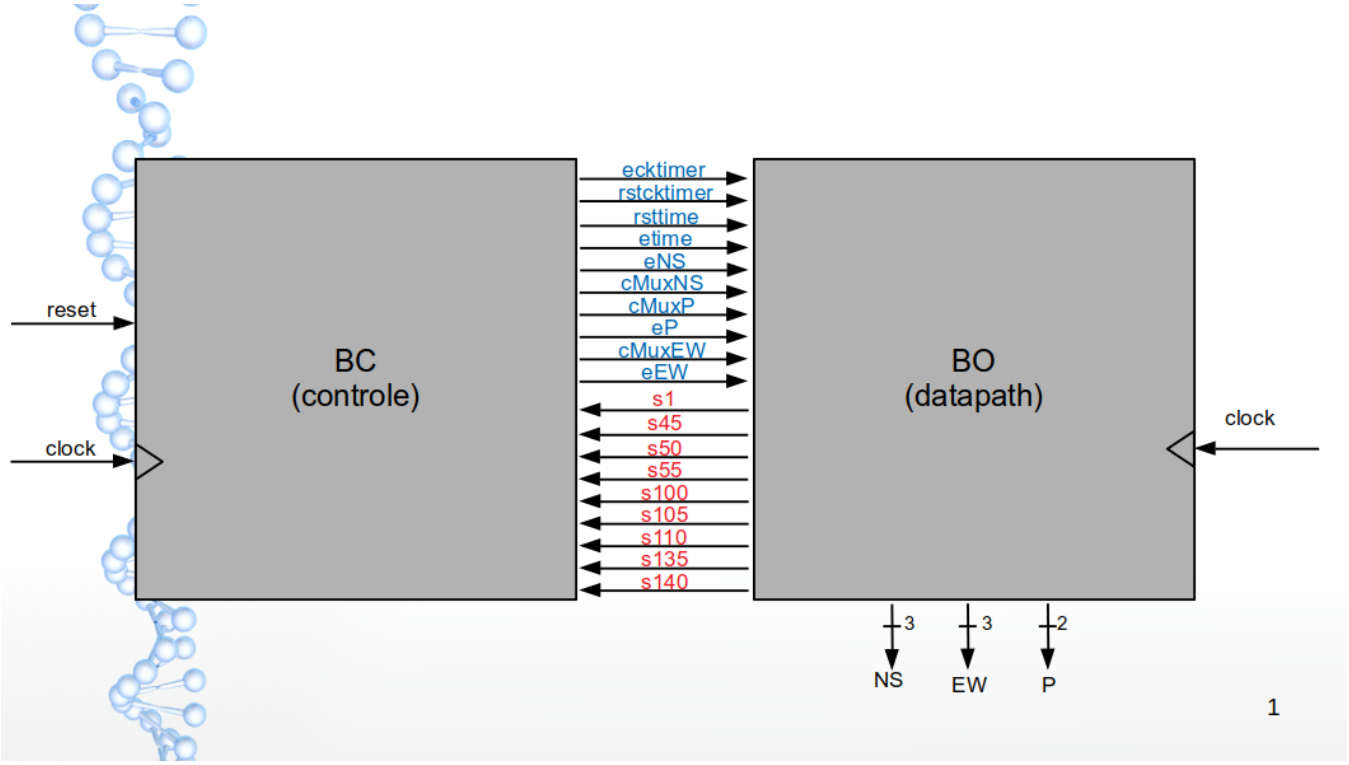

Figure 2.11: Diagrama bloco de controle, bloco operativo – O sistema digital completo proposto pode ser resumido, com base nas seções anteriores, em um diagrama BO/BC.

Para uma visão mais detalhada da ligação dos sinais de entrada e saída do bloco operativo, efetuou-se um mapeamento de seus sinais, visto abaixo, para facilitar a visualização destes.

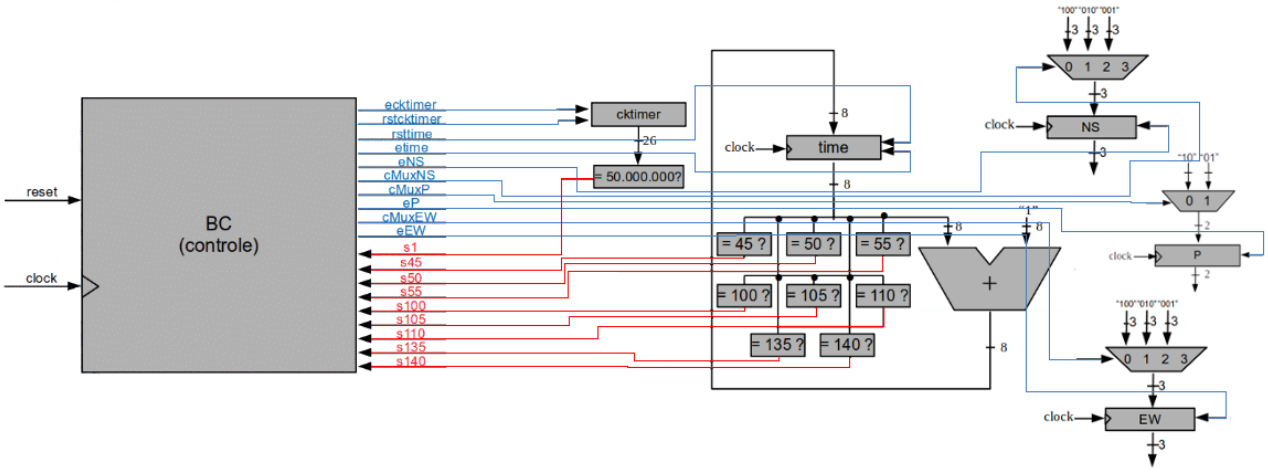

Figure 2.12: Mapeamento dos sinais de entradas e saídas do bloco de controle

3 Desenvolvimento

A partir da descrição do sistema digital apresentado na seção 2, que inclui o psudocódigo, projeto do bloco operativo e projeto do bloco de controle, podemos realizar o desenvolvimento de cada componente exposto, apresentando o VHDL, temporização, área utilizada pelo componente na placa e esquemático RTL.

3.1 Bloco de Controle

O componente “BC” representa nosso Bloco de controle e tem como entrada: o “clock”, servido para temporização e o “reset”, e os sinais s1, s45, s50, s55, s100, s105, s110, s135 e s140 utilizados para tomar decisões e oferecer as saídas ecktimer, rstcktimer, rsttime, etime, eNS, eP, eEW, cMuxNS, cMuxEW e cMuxP. Pode-se dizer que este componente é o cérebro de nosso projeto pois ele tomará todas as decisões dependendo das entradas e enviará sinais ao bloco operativo fazer as operações. Pode-se observar na figura 2.12 a ligação entre os dois componentes (BO e BC) e na figura 2.10 e 3.26 a lógica de transição de estados.

3.1.1 Código VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity BC is
port(
-- control inputs
clock, reset: in std_logic;
s1, s45, s50, s55, s100, s105, s110, s135, s140: in std_logic;

-- control outputs
ecktimer, rstcktimer, rsttime, etime, eNS, eP, eEW: out std_logic;

cMuxNS, cMuxEW: out std_logic_vector(1 downto 0);
cMuxP: out std_logic
);
end entity;

architecture archBC of BC  is
type InternalState is (init, S0, S2, S3, S4, S5, S6, S7, S8);
signal nextState, currentState: InternalState;
begin
NSL: process(s1, s45, s50, s55, s100, s105, s110, s135, s140) is

-- next-state logic (combinatorial)
begin
nextState <= currentState;
case currentState is
when init =>
nextState <= S0;
when S0 =>
if(s1='1') then
nextState <= S2;
else
nextState <= S0;
end if;
when S2 =>
  if not(s45='1' and s50='1' and s55='1' and s100='1' and s105='1' and s110='1' and s135='1' and s140='1') then
 nextState <= S0;
  end if;
  if s45='1' and not(s50='1' or s55='1' or s100='1' or s105='1' or s110='1' or s135='1' or s140='1') then
 nextState <= S3;
  end if;
  if ((s50='1') or (s105='1') or (s135='1')) and not(s45='1' or s55='1' or s100='1' or s110='1' or s140='1') then
 nextState <= S4;
  end if;
  if s55='1' and not(s45='1' or s50='1' or s100='1' or s105='1' or s110='1' or s135='1' or s140='1') then
 nextState <= S5;
  end if;
  if s100='1' and not(s45='1' or s50='1' or s55='1' or s105='1' or s110='1' or s135='1' or s140='1') then
 nextState <= S6;
  end if;
  if s110='1' and not(s45='1' or s50='1' or s55='1' or s100='1' or s105='1' or s135='1' or s140='1') then
 nextState <= S7;
  end if;
  if s140='1' and not(s45='1' or s50='1' or s55='1' or s100='1' or s105='1' or s110='1' or s135='1') then
 nextState <= S8;
  end if;
when S3 =>
nextState <= S0;
when S4 =>
nextState <= S0;
when S5 =>
nextState <= S0;
when S6 =>
nextState <= S0;
when S7 =>
nextState <= S0;
when S8 =>
nextState <= S0;
end case;
end process;
```

```
-- memory element (sequential)

process(clock, reset) is
begin
if reset='1' then
currentState <= init; -- reset state
elsif rising_edge(clock) then
currentState <= nextState;
end if;
end process;

-- output-logic

ecktimer <= '1' when (currentState = S0) else '0';

rstcktimer <= '1' when (currentState = init
or currentState = S2
or currentState = S8) else '0';

rsttime <= '1' when (currentState = init
or currentState = S8) else '0';

etime <= '1' when (currentState = S2) else '0';

eNS <= '1' when (currentState = init
or currentState = S3
or currentState = S4
or currentState = S8) else '0';

cMuxNS <= "00" when (currentState = init
or currentState = S0
or currentState = S2
or currentState = S8) else
"01" when (currentState = S3) else
"10" when (currentState = S4
or currentState = S5
or currentState = S6
or currentState = S7);

cMuxP <= '0' when (currentState = S0
or currentState = S2
or currentState = S7) else '1';

eP <= '1' when (currentState = init
or currentState = S4
or currentState = S7) else '0';

cMuxEW <= "00" when (currentState = S0
or currentState = S2
or currentState = S5) else
"01" when (currentState = S6) else
"10" when (currentState = init
or currentState = S3
or currentState = S4
or currentState = S7
or currentState = S8);

eEW <= '1' when (currentState = init
or currentState = S4
or currentState = S5
or currentState = S6) else '0';

end architecture;
```

3.1.2 Temporização

Resumo do fluxo do componente: Total logic elements 36 / 33,216 (<1%), total combinational functions 36 / 33,216 (<1%), dedicated logic registers 9 / 33.216 (1%), total registers, total pins 23 / 475 (5%).

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	cMuxEW[*]	clock	7.204	7.204	Rise	clock
1	cMuxEW[0]	clock	7.223	7.223	Rise	clock
2	cMuxEW[1]	clock	7.204	7.204	Rise	clock
2	cMuxP	clock	7.185	7.185	Rise	clock
3	eNS	clock	7.138	7.138	Rise	clock
4	eEW	clock	7.122	7.122	Rise	clock
5	cMuxNS[*]	clock	7.056	7.056	Rise	clock
1	cMuxNS[0]	clock	7.088	7.088	Rise	clock
2	cMuxNS[1]	clock	7.056	7.056	Rise	clock
6	rstcktimer	clock	6.943	6.943	Rise	clock
7	eP	clock	6.923	6.923	Rise	clock
8	rsttime	clock	6.916	6.916	Rise	clock
9	etime	clock	6.405	6.405	Rise	clock
10	ecktimer	clock	6.402	6.402	Rise	clock

Figure 3.1: Temporização do bloco de controle.

3.1.3 Área utilizada

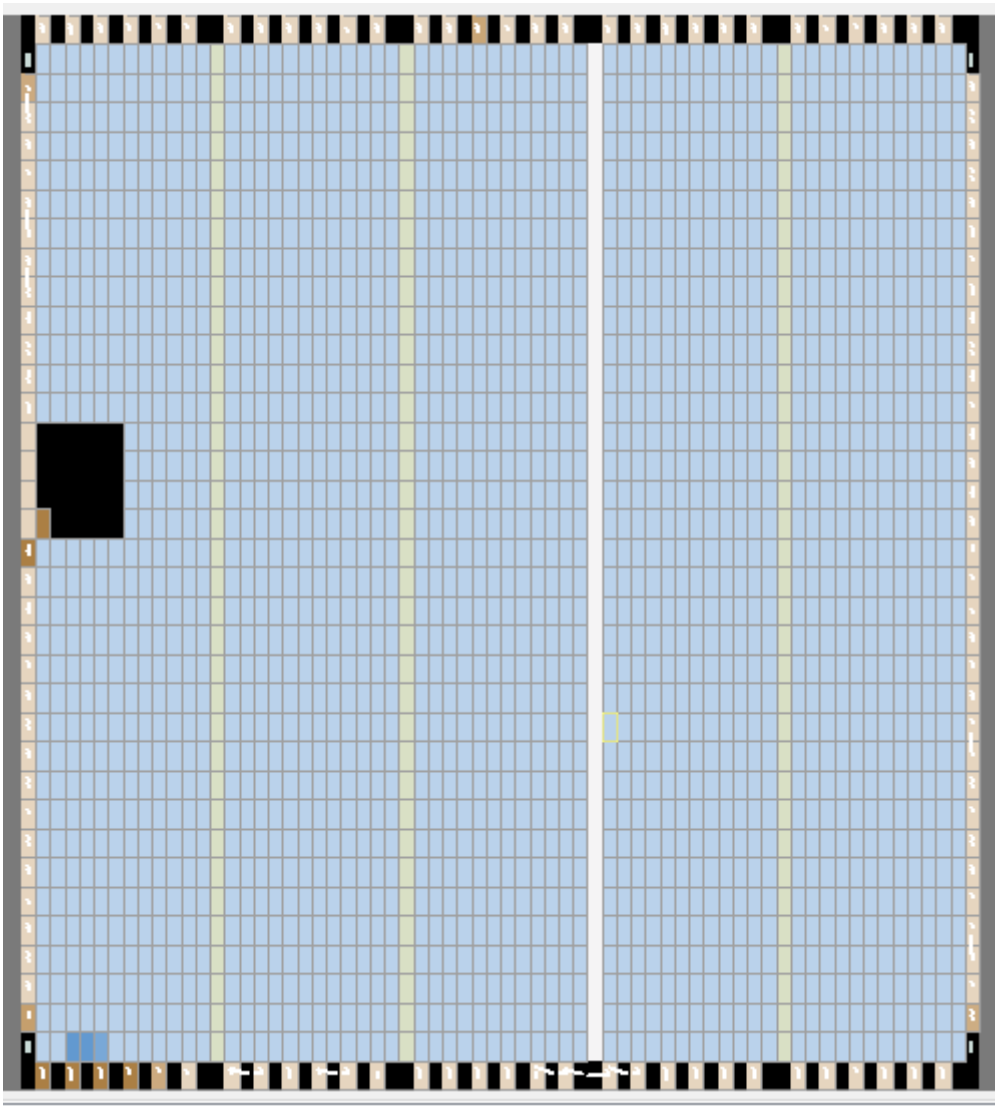

Figure 3.2: Área utilizada pelo bloco de controle.

3.1.4 Esquemático RTL

A figura 3.3 representa o esquemático (RTL) do bloco de controle do projeto. Pode-se notar as entradas do bloco e as suas saídas correspondentes ligando-se a outros componentes gerados automaticamente.

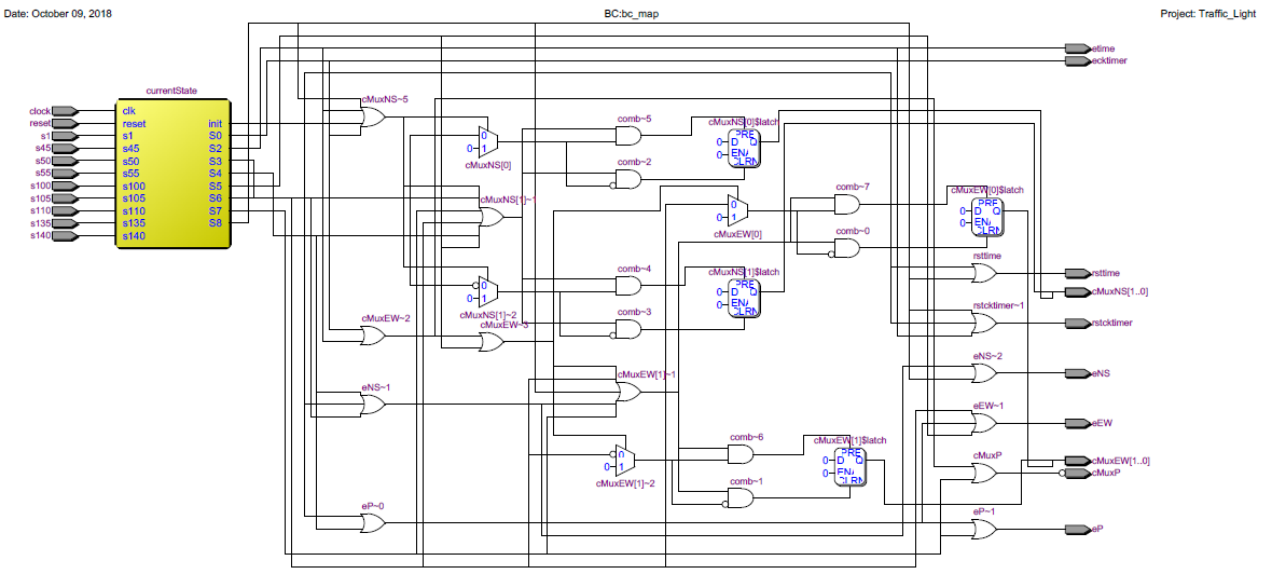

Figure 3.3: Esquemático RTL do bloco de controle.

3.1.5 Componentes Bloco Operativo

Apresentação dos componentes do bloco operativo, contendo seu código VHDL, temporização, área utilizada e esquemático RTL.

3.1.5.1 Adder n bits

O componente "adder_n_bits" admite duas entradas e uma saída, ambas n bits, ou seja, dependendo do propósito que o projeto, pode-se alterar a quantidade de bits que o somador irá ter como entrada e saída. Tem como objetivo principal realizar a soma de dois inteiros com sinal. Este componente é utilizado em dois âmbitos distintos, para a soma da saida do registrador "cktimer" com "1" e também para a soma da saída do registrador "time" com 1.

3.1.5.1.1 Código VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder_n_bits is
	generic(N: positive := 8);
	port(
		inpt0, inpt1: in std_logic_vector(N-1 downto 0);
		outpt: out std_logic_vector(N-1 downto 0)
	);
end entity;

architecture archAdder of adder_n_bits is
	begin
		outpt <= std_logic_vector(signed(inpt0) + signed(inpt1));
end architecture;
```

3.1.5.1.2 Temporização

Resumo do fluxo do componente: Total logic elements 8 / 33,216 (<1%), total combinational functions 8 / 33,216 (<1%), dedicated logic registers 0 / 33.216 (0%), total pins 24 / 475 (5%).

	Input Port	Output Port	RR	RF	FR	FF
1	inpt0[2]	outpt[3]	12.065	12.065	12.065	12.065
2	inpt1[1]	outpt[3]	11.702	11.702	11.702	11.702
3	inpt0[2]	outpt[7]	11.521	11.521	11.521	11.521
4	inpt0[2]	outpt[6]	11.423	11.423	11.423	11.423
5	inpt0[1]	outpt[3]	11.412	11.412	11.412	11.412
6	inpt1[2]	outpt[3]	11.349	11.349	11.349	11.349
7	inpt0[2]	outpt[4]	11.296	11.296	11.296	11.296
8	inpt1[1]	outpt[7]	11.158	11.158	11.158	11.158
9	inpt1[1]	outpt[6]	11.060	11.060	11.060	11.060
10	inpt1[4]	outpt[7]	10.983	10.983	10.983	10.983

Figure 3.4: Temporização do adder.

3.1.5.1.3 Área Utilizada

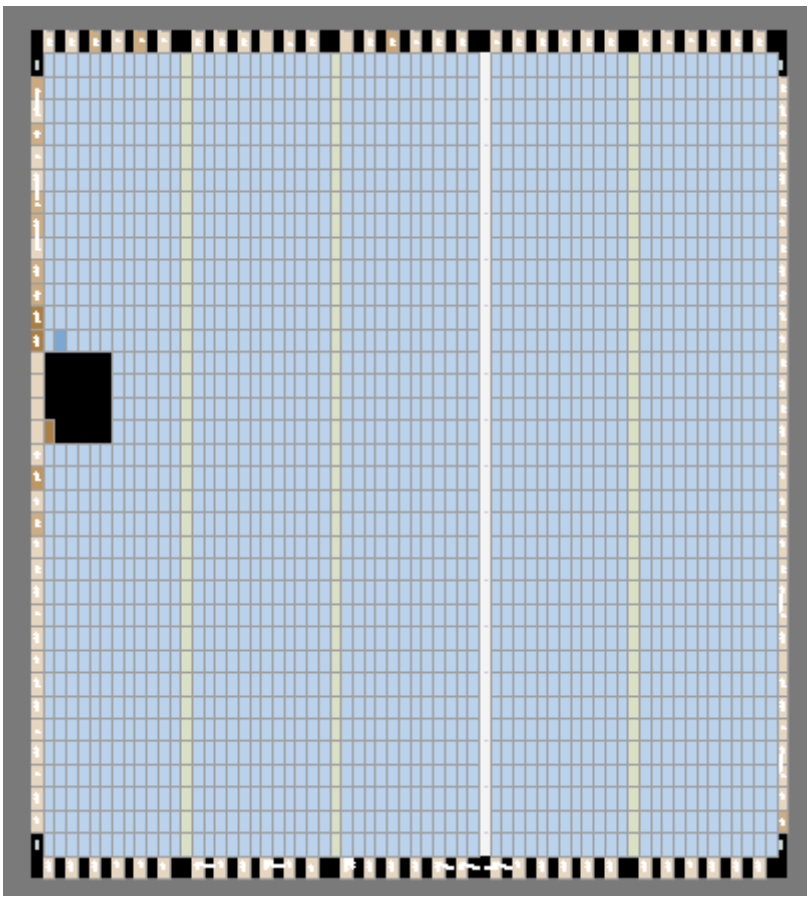

Figure 3.5: Área utilizada pelo adder.

3.1.5.1.4 Esquemático RTL

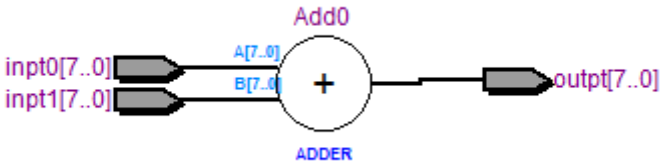

Figure 3.6: Esquemático do adder.

3.1.5.2 Compare if Equal

O componente "compareIfEqual_n_bits" tem dois sinais de entrada e um de saída, ambos com n bits. Seu objetivo é comparar dois valores e retornar o valor "1" se eles forem exatamente iguais, caso não forem, retornará "0". É utilizado massivamente no projeto, tendo em vista que precisamos comparar se um valor é ou não igual a outro. Empregou-se este componente para comparar se a entrada fosse igual a:

- 50000000 = este valor equivale a frequência utilizada na placa da altera e nos auxiliará futuramente para a implementação das próximas etapas. Com finalidade de abstração e simplificação dos testes, utilizou-se o valor de 500. Tem como saída o sinal "s1" que será utilizado no bloco de controle.
- 45, 50, 55, 100, 105, 110, 135, 140 = compararão o valor advindo da saída do registrador time e caso forem verdadeiros, dispararão os sinais s45, s50, s55, s100, s105, s110, s135 e s140 respectivamente, que serão utilizados pelo bloco de controle.

3.1.5.2.1 Código VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity compareIfEqual_n_bits is
	generic(N: integer := 8);
	port(
		inpt0, inpt1: in std_logic_vector(N-1 downto 0);
		outpt: out std_logic
	);
end entity;

architecture archCompareIfEqual of compareIfEqual_n_bits is
	begin
		outpt <= '1' when inpt0 = inpt1 else '0';
end architecture;
```

3.1.5.2.2 Temporização

Resumo do fluxo do componente: Total logic elements 8 / 33,216 (<1%), total combinational functions 8 / 33,216 (<1%), dedicated logic registers 0 / 33.216 (0%), total pins 17 / 475 (4%).

	Input Port	Output Port	RR	RF	FR	FF
1	inpt0[2]	outpt[3]	12.065	12.065	12.065	12.065
2	inpt1[1]	outpt[3]	11.702	11.702	11.702	11.702
3	inpt0[2]	outpt[7]	11.521	11.521	11.521	11.521
4	inpt0[2]	outpt[6]	11.423	11.423	11.423	11.423
5	inpt0[1]	outpt[3]	11.412	11.412	11.412	11.412
6	inpt1[2]	outpt[3]	11.349	11.349	11.349	11.349
7	inpt0[2]	outpt[4]	11.296	11.296	11.296	11.296
8	inpt1[1]	outpt[7]	11.158	11.158	11.158	11.158
9	inpt1[1]	outpt[6]	11.060	11.060	11.060	11.060
10	inpt1[4]	outpt[7]	10.983	10.983	10.983	10.983

Figure 3.7: Temporização do comparador.

3.1.5.2.3 Área Utilizada

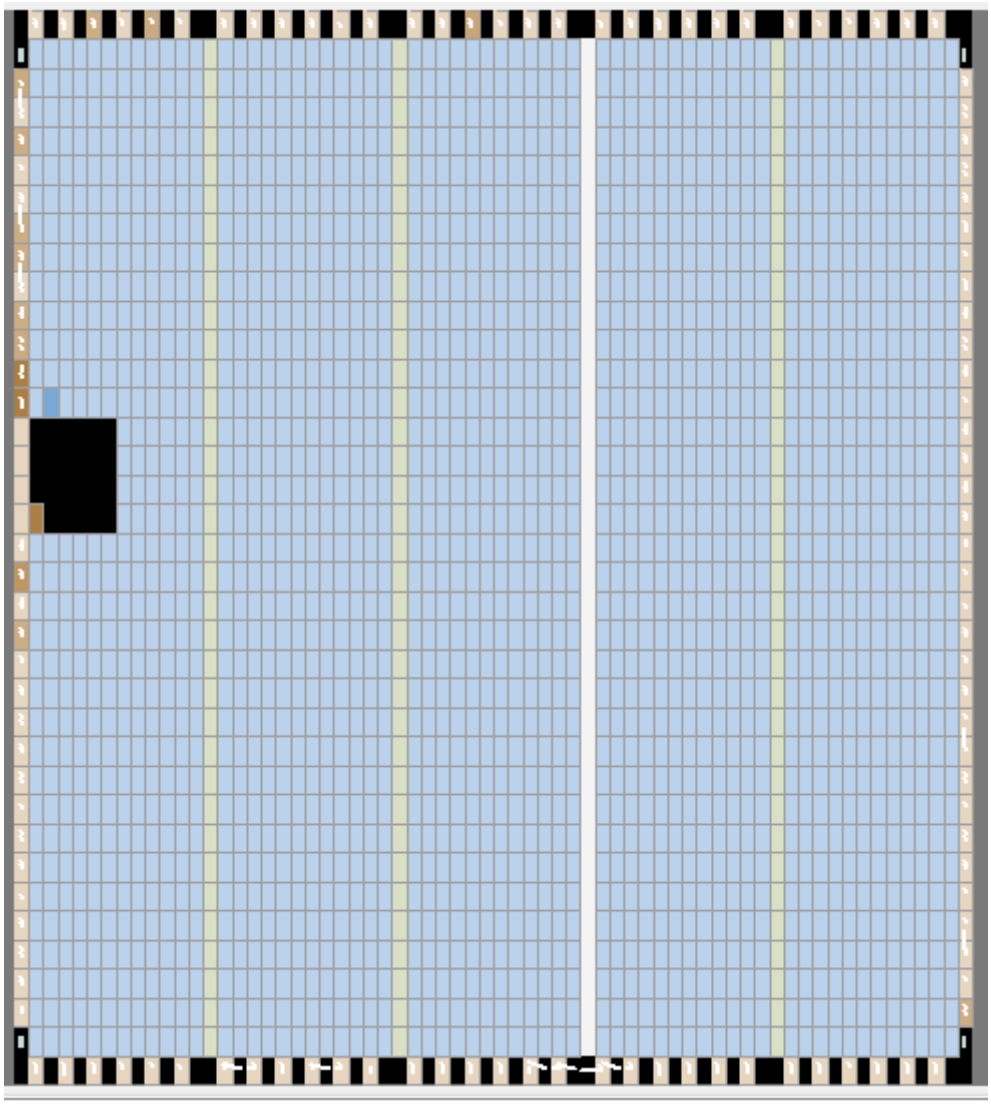

Figure 3.8: Área utilizada pelo comparador.

3.1.5.2.4 Esquemático RTL

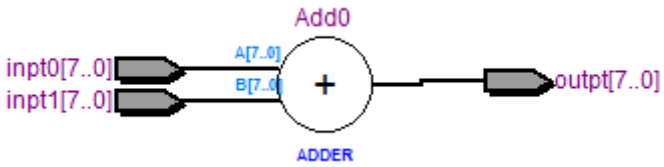

Figure 3.9: Esquemático RTL do comparador.

3.1.5.3 Register

O componente “register_n_bits” tem como entrada de controle: o “clock”, servido para temporização, o “reset”, para zerar o valor do registrador e também o “enable”, dizendo quando o componente deve armazenar o valor, todos com 1 bit. Também tem uma entrada e uma saída, ambas com n bits. Seu objetivo é armazenar valores, ou seja, quando o sinal de “enable” estiver valendo “1”, guarda o valor de entrada o oferece como saída até que outro valor seja tido como entrada, caso estiver valendo “0” fará com que a

entrada não seja mudada, conservando o valor até que seja utilizado. Este componente é aproveitado 5 vezes no bloco operativo, sabendo que o “clock” é compartilhado entre todos os registradores e o sinal “reset” é assíncrono:

- **time**: Tendo os sinais de “rsttime” e “etime” como sinais de “reset” e “enable”, entrada de 8 bits advindas do resultado do somador e saída também de 8 bits. Como já mencionado, ele armazena o resultado da soma de seu valor com “1”.
- **cktimer:** Conta com os sinais de “rstcktimer” e “ecktimer” como sinais de “reset” e “enable”, entrada de 26 bits advindas do resultado do somador e saída também de 26 bits. Como já mencionado, ele armazena o resultado da soma de seu valor com “1”.
- **NS:** Tem como sinal de entrada o valor advindo do mux4x1_n_bits, de 3 bits e tem como “enable” o sinal chamado “eNS”. Como saída, tem um valor com a mesma quantidade de bits de entrada. Tem como finalidade armazenar o valor do semáforo sentido Norte-Sul.
- **P**: Tem como sinal de entrada o valor advindo do mux2x1_n_bits, de 2 bits e tem como “enable” o sinal chamado “eP”. Como saída, tem um valor com a mesma quantidade de bits de entrada. Tem como finalidade armazenar o valor do semáforo dos pedestres.
- **EW**: Tem como sinal de entrada o valor advindo do mux4x1_n_bits, de 3 bits e tem como “enable” o sinal chamado “eEW”. Como saída, tem um valor com a mesma quantidade de bits de entrada. Tem como finalidade armazenar o valor do semáforo sentido Leste-Oeste.

3.1.5.3.1 Código VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity register_n_bits is
	generic (N: positive := 8);
	port(
		-- control inputs
		clock, reset, enable: in std_logic;
		-- data inputs
		inpt: in std_logic_vector(N-1 downto 0);
		-- data outputs
		outpt: out std_logic_vector(N-1 downto 0)
	);
end entity;

architecture archRegister of register_n_bits is
	subtype InternalState is std_logic_vector(N-1 downto 0);
	signal nextState, currentState: InternalState;
	begin

	-- next state logic (combinatorial)
		nextState <= inpt;

	-- memory element (sequential)
	ME: process (clock, reset) is
	begin
		if reset='1' then
				currentState <= (others=>'0'); -- reset state
		elsif rising_edge(clock) then
				if(enable = '1') then
						currentState <= nextState;
				end if;
		end if;
	end process;

	-- output logic (combinatorial)
	outpt <= currentState;

end architecture;
```

3.1.5.3.2 Temporização

Resumo do fluxo do componente: Total logic elements 8 / 33,216 (<1%), total combinational functions 0 / 33,216 (0%), dedicated logic registers 8 / 33.216(<1%), total registers 8, total pins 19 / 475 (4%).

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	outpt[*]	clock	6.226	6.226	Rise	clock
1	outpt[5]	clock	8.673	8.673	Rise	clock
2	outpt[2]	clock	7.845	7.845	Rise	clock
3	outpt[6]	clock	6.632	6.632	Rise	clock
4	outpt[3]	clock	6.402	6.402	Rise	clock
5	outpt[7]	clock	6.388	6.388	Rise	clock
6	outpt[4]	clock	6.377	6.377	Rise	clock
7	outpt[1]	clock	6.341	6.341	Rise	clock
8	outpt[0]	clock	6.226	6.226	Rise	clock

Figure 3.10: Temporização do register.

3.1.5.3.3 Área Utilizada

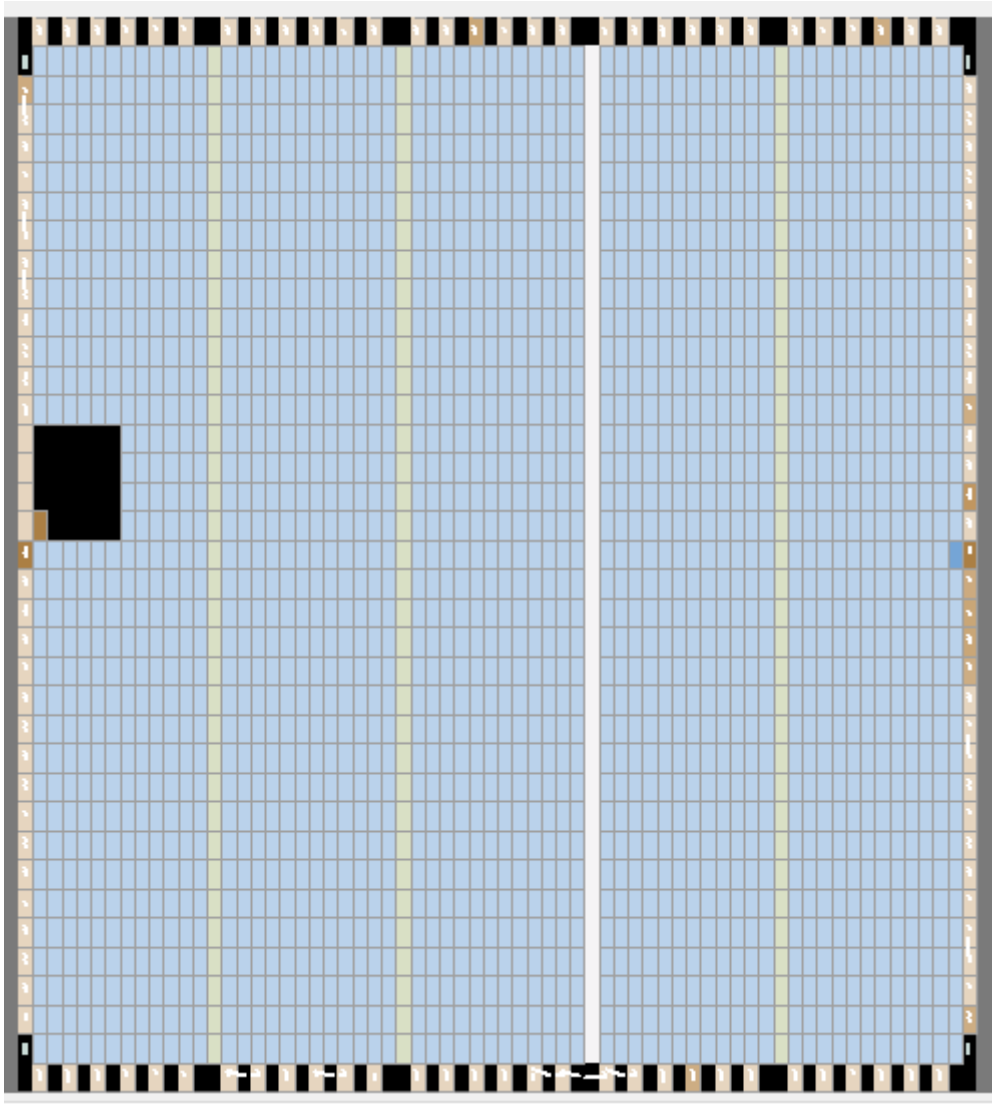

Figure 3.11: Área utilizada pelo register.

3.1.5.3.4 Esquemático RTL

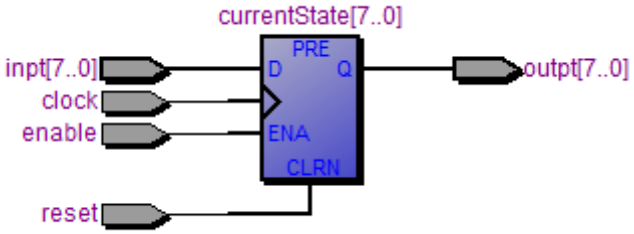

Figure 3.12: Esquemático RTL do register.

3.1.5.4 Mux4x1

O componente "mux4x1_n_bits" terá como sinal de entrada quatro sinais de n bits e uma saída também de n bits. A finalidade do componente é, selecionar a entrada dependendo do valor do sinal de seleção, neste caso de 2 bits. Caso o seletor valer:

- "00", selecionará a primeira entrada fornecida e a determinará como saída ou
- "01", selecionará a segunda entrada fornecida e a determinará como saída ou
- "10", selecionará a terceira entrada fornecida e a determinará como saída, senão a quarta entrada fornecida será a saída.

É utilizado em dois lugares diferentes no bloco operativo, para a atribuição do valor do registrador "NS" e do registrador "EW". Caso o sinal cMuxNS(seletor) valha:

- "00", selecionará "100" como saída ou
- "01", selecionará "010" como saída ou
- "10", selecionará "001" como saída, outro qualquer valor não será utilizado, portanto será feito uma espécie de aterramento da quarta entrada.

3.1.5.4.1 Código VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux4x1_n_bits is
	generic(
		n: positive := 3
	);
	port(
		inpt0, inpt1, inpt2, inpt3: in std_logic_vector(n-1 downto 0);
		sel: in std_logic_vector(1 downto 0);
		outpt: out std_logic_vector(n-1 downto 0)
	);
end mux4x1_n_bits;

architecture archMux of mux4x1_n_bits is
	begin
		outpt <= inpt0 when sel= "00" else
				inpt1 when sel= "01" else
				inpt2 when sel= "10" else inpt3;

end architecture;
```

3.1.5.4.2 Temporização

Resumo do fluxo do componente: Total logic elements 8 / 33,216 (<1%), total combinational functions 6 / 33,216 (<1%), dedicated logic registers 0 / 33.216 (0%), total pins 17 / 475 (4%).

	Input Port	Output Port	RR	RF	FR	FF
1	sel[0]	outpt[2]	11.470	11.470	11.470	11.470
2	inpt0[2]	outpt[2]	11.341	11.341	11.341	11.341
3	inpt1[2]	outpt[2]	11.136	11.136	11.136	11.136
4	inpt3[2]	outpt[2]	10.857			10.857
5	inpt2[2]	outpt[2]	10.653			10.653
6	inpt3[0]	outpt[0]	10.209			10.209
7	sel[0]	outpt[0]	10.161	10.161	10.161	10.161
8	sel[0]	outpt[1]	10.015	10.015	10.015	10.015
9	inpt0[0]	outpt[0]	9.997	9.997	9.997	9.997
10	inpt1[0]	outpt[0]	9.868	9.868	9.868	9.868

Figure 3.13: Temporização do mux4x1.

3.1.5.4.3 Área Utilizada

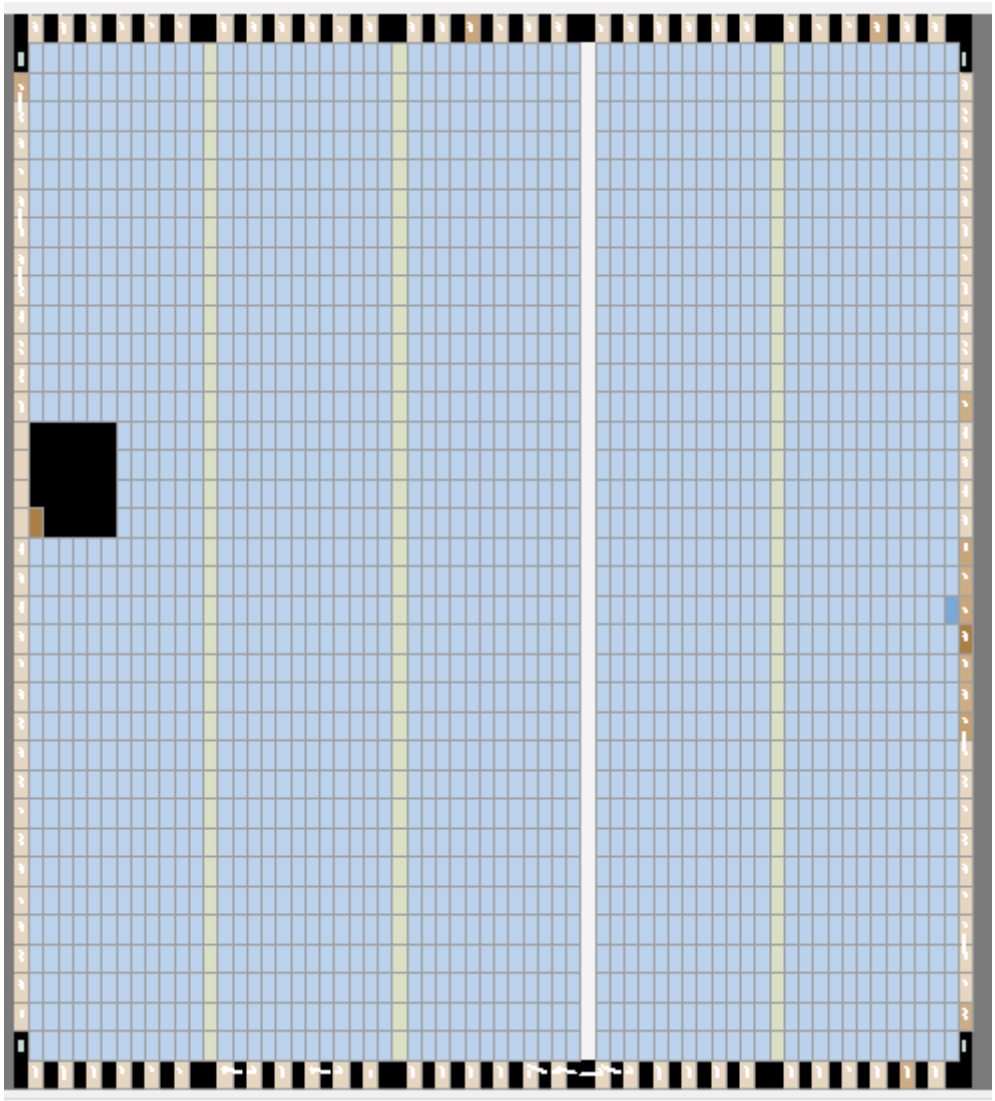

Figure 3.14: Área utilizada pelo mux4x1.

3.1.5.4.4 Esquemático RTL

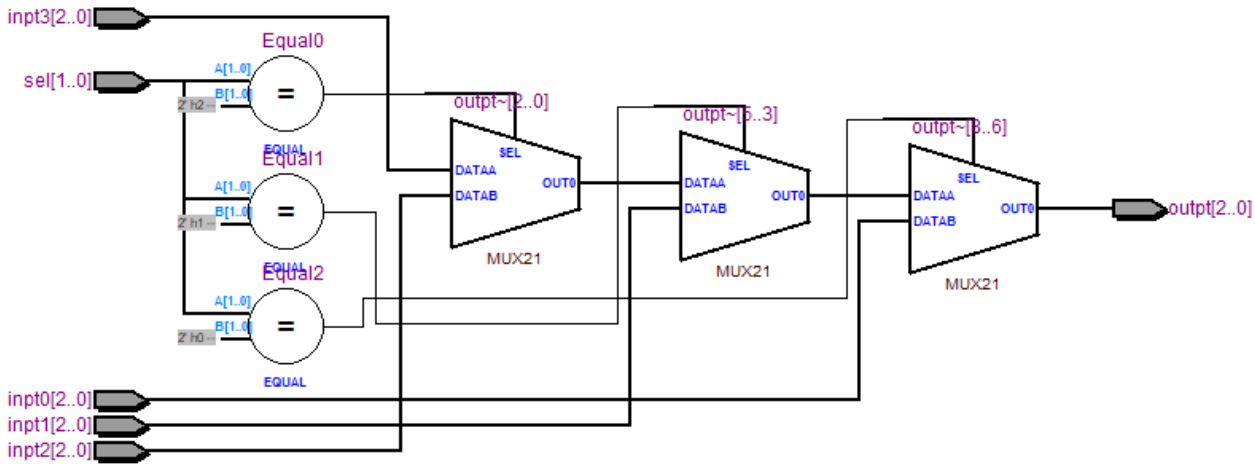

Figure 3.15: Esquemático RTL do mux4x1.

3.1.6 Mux 2x1

O componente "mux2x1_n_bits" terá como sinal de entrada dois sinais de n bits e uma saída também de n bits. A finalidade do componente é, selecionar a entrada dependendo do valor do sinal de seleção, neste caso de 1 bit. Caso o seletor valer "0", selecionará a primeira entrada fornecida como saída, senão, selecionará a segunda. É utilizado em somente um lugar no bloco operativo, para a atribuição do valor do registrador "P". Caso o sinal cMuxP(seletor) valha:

- "0", selecionará "10" como saída ou
- "1", selecionará "01" como saída.

3.1.6.1.1 Código VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux2x1_n_bits is
	generic(N : positive := 2);
	port(
		inpt0, inpt1: in std_logic_vector(N-1 downto 0);
		sel: in std_logic;
		outpt: out std_logic_vector(N-1 downto 0)
	);
end mux2x1_n_bits;

architecture archMux of mux2x1_n_bits is
	begin
		outpt <= inpt0 when sel= '0' else inpt1;

end architecture;
```

3.1.6.1.2 Temporização

Resumo do fluxo do componente: Total logic elements 2 / 33,216 (<1%), total combinational functions 2 / 33,216 (<1%), dedicated logic registers 0 / 33.216 (0%), total pins 7 / 475 (1%).

	Input Port	Output Port	RR	RF	FR	FF
1	sel	outpt[1]	9.488	9.488	9.488	9.488
2	sel	outpt[0]	9.473	9.473	9.473	9.473
3	inpt0[1]	outpt[1]	9.442			9.442
4	inpt1[1]	outpt[1]	9.426			9.426
5	inpt1[0]	outpt[0]	5.394			5.394
6	inpt0[0]	outpt[0]	5.121			5.121

Figure 3.16: Temporização do mux2x1.

3.1.6.1.3 Área Utilizada

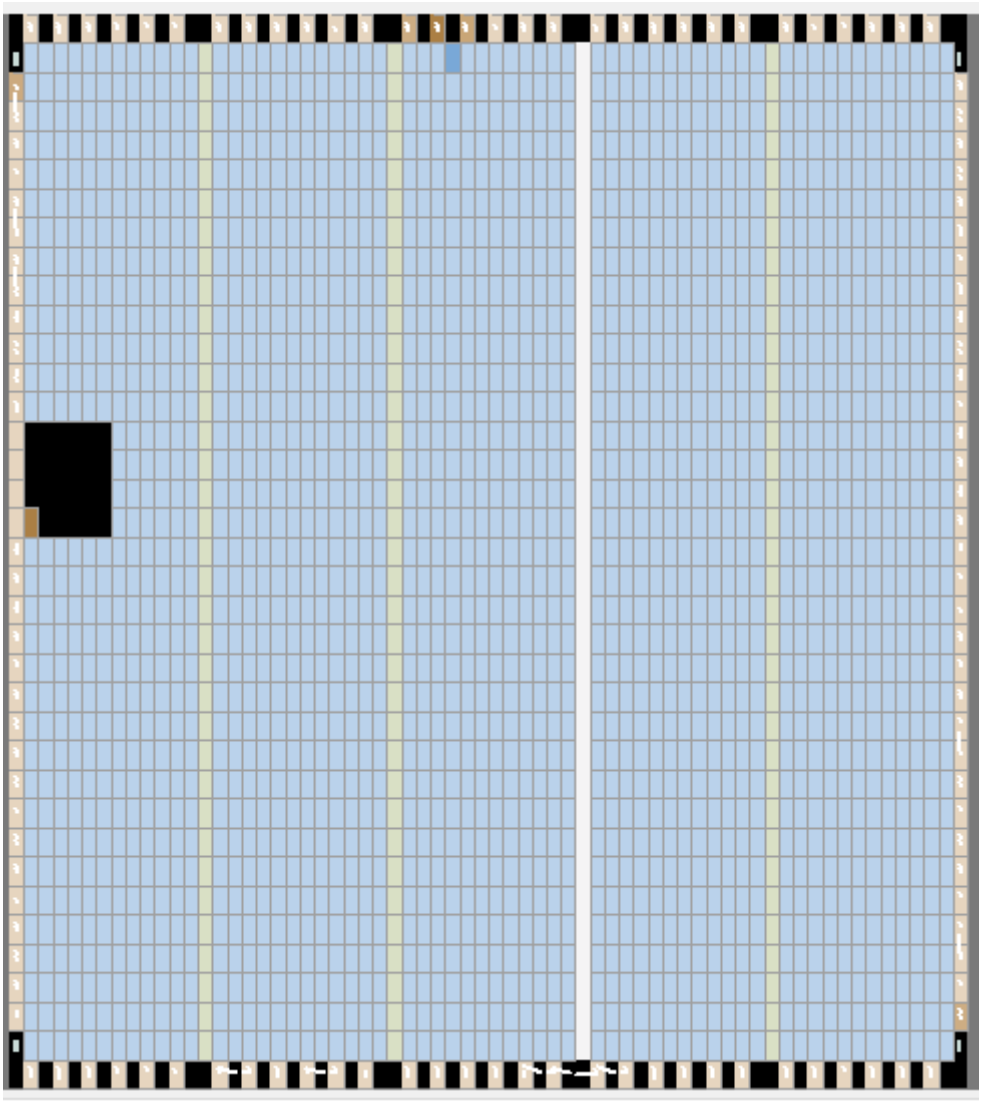

Figure 3.17: Área utilizada pelo mux2x1.

3.1.6.1.4 Esquemático RTL

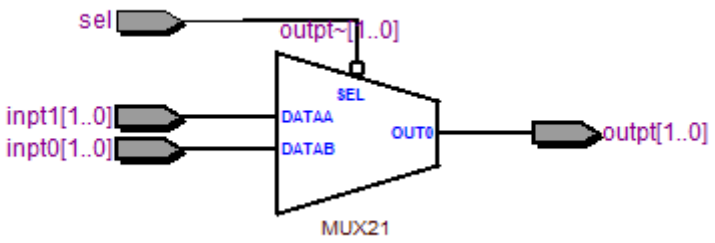

Figure 3.18: Esquemático RTL do mux2x1.

3.2 Bloco Operativo

O componente “BO” representa nosso Bloco Operativo e tem como entrada: o “clock”, servido para temporização e o “reset”, e os sinais ecktimer, rstcktimer, rsttime, etime, eNS, eP, eEW, cMuxNS, cMuxEW e cMuxP advindos do bloco de controle e necessários para oferecer as saídas s1, s45, s50, s55, s100, s105, s110, s135, s140, NS, EW e P corretamente. Pode-se dizer que este componente é os músculos de nosso projeto pois ele realizará todas as operações aritméticas e executará o que deve ser executado. A partir dessas operações ele enviará os resultados ao bloco de controle. Ele integrará todos os componentes já mencionados (“register_n_bits”, “adder_n_bits”, “compareIfEqual_n_bits”, “mux4x1_n_bits” e “mux2x1_n_bits”) através de um mapeamento de portas visto na figura 2.12.

3.2.1 Código VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.math_real.all;

entity BO is
	port(
		-- operative inputs
		clock, reset: in std_logic;
		ecktimer, rstcktimer, rsttime, etime, eNS, eP, eEW: in std_logic;
		cMuxNS, cMuxEW: in std_logic_vector(1 downto 0);
		cMuxP : in std_logic;

		-- operative outputs
		s1, s45, s50, s55, s100, s105, s110, s135, s140: out std_logic;

		-- data outputs
		NS, EW: out std_logic_vector(2 downto 0);
		P: out std_logic_vector(1 downto 0)
	);
end entity;

architecture archBO of BO is

component adder_n_bits
	generic(N: positive := 8);
	port(
		inpt0, inpt1: in std_logic_vector(N-1 downto 0);
		outpt: out std_logic_vector(N-1 downto 0)
	);
end component;

component register_n_bits
	generic (N: positive := 8);
	port(
		clock, reset, enable: in std_logic;
		inpt: in std_logic_vector(N-1 downto 0);
		outpt: out std_logic_vector(N-1 downto 0)
	);
end component;

component compareIfEqual_n_bits
	generic(N: integer := 8);
	port(
		inpt0, inpt1: in std_logic_vector(N-1 downto 0);
		outpt: out std_logic
	);
end component;

```

```
component mux2x1_n_bits is
	generic(N: positive := 2);
	port(
		inpt0, inpt1: in std_logic_vector(n-1 downto 0);
		sel: in std_logic;
		outpt: out std_logic_vector(n-1 downto 0)
	);
end component;

component mux4x1_n_bits is
	generic(n: positive := 3);
	port(
		inpt0, inpt1, inpt2, inpt3: in std_logic_vector(n-1 downto 0);
		sel: in std_logic_vector(1 downto 0);
		outpt: out std_logic_vector(n-1 downto 0)
	);
end component;


--signal declaration

signal saicktimer, saisomacktimer: std_logic_vector (25 DOWNTO 0);
signal saitime, saisomatime: std_logic_vector (7 DOWNTO 0);
signal saimuxNS, saimuxEW: std_logic_vector (2 DOWNTO 0);
signal saimuxP: std_logic_vector (1 DOWNTO 0);

begin

	Rcktimer : register_n_bits GENERIC MAP (26) PORT MAP(clock, rstcktimer, ecktimer, saisomacktimer, saicktimer);
	Acktimer : adder_n_bits GENERIC MAP (26) PORT MAP(saicktimer, "00000000000000000000000001", saisomacktimer);
	Cs1 : compareIfEqual_n_bits GENERIC MAP (26) PORT MAP(saicktimer, "00000000000000000111110100", s1);--500
	--Cs1 : compareIfEqual_n_bits GENERIC MAP (26) PORT MAP(saicktimer, "10111110101111000010000000", s1);--50000000000

	Rtime : register_n_bits PORT MAP(clock, rsttime, etime, saisomatime, saitime);
	Atime : adder_n_bits PORT MAP(saitime, "00000001", saisomatime);
	Cs45 : compareIfEqual_n_bits PORT MAP(saitime, "00101101", s45);
	Cs50 : compareIfEqual_n_bits PORT MAP(saitime, "00110010", s50);
	Cs55 : compareIfEqual_n_bits PORT MAP(saitime, "00110111", s55);
	Cs100 : compareIfEqual_n_bits PORT MAP(saitime, "01100100", s100);
	Cs105 : compareIfEqual_n_bits PORT MAP(saitime, "01101001", s105);
	Cs110 : compareIfEqual_n_bits PORT MAP(saitime, "01101110", s110);
	Cs135 : compareIfEqual_n_bits PORT MAP(saitime, "10000111", s135);
	Cs140 : compareIfEqual_n_bits PORT MAP(saitime, "10001100", s140);

	MNS: mux4x1_n_bits GENERIC MAP (3) PORT MAP("100", "010", "001", "000", cMuxNS, saiMuxNS);
	RNS : register_n_bits GENERIC MAP (3) PORT MAP(clock, reset, eNS, saimuxNS, NS);

	MP: mux2x1_n_bits GENERIC MAP (2) PORT MAP("10", "01", cMuxP, saiMuxP);
	RP : register_n_bits GENERIC MAP (2) PORT MAP(clock, reset, eP, saiMuxP, P);

	MEW: mux4x1_n_bits GENERIC MAP (3) PORT MAP("100", "010", "001", "000",  cMuxEW, saiMuxEW);
	REW : register_n_bits GENERIC MAP (3) PORT MAP(clock, reset, eEW, saiMuxEW, EW);

end architecture;
```

3.2.2 Temporização

Resumo do fluxo do componente: Total logic elements 67 / 33,216 (<1%), total combinational functions 66 / 33,216 (<1%), dedicated logic registers 42 / 33.216 (<1%), total registers 42, total pins 31 / 475 (7%).

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	s1	clock	7.382	7.382	Rise	clock
2	s45	clock	7.349	7.349	Rise	clock
3	s110	clock	7.268	7.268	Rise	clock
4	s100	clock	7.225	7.225	Rise	clock
5	s135	clock	7.148	7.148	Rise	clock
6	s105	clock	7.135	7.135	Rise	clock
7	s140	clock	7.130	7.130	Rise	clock
8	s50	clock	7.110	7.110	Rise	clock
9	s55	clock	7.108	7.108	Rise	clock
10	P[*]	clock	6.333	6.333	Rise	clock
1	P[1]	clock	6.345	6.345	Rise	clock
2	P[0]	clock	6.333	6.333	Rise	clock
11	NS[*]	clock	6.042	6.042	Rise	clock
1	NS[1]	clock	6.054	6.054	Rise	clock
2	NS[0]	clock	6.052	6.052	Rise	clock
3	NS[2]	clock	6.042	6.042	Rise	clock
12	EW[*]	clock	6.037	6.037	Rise	clock
1	EW[2]	clock	6.077	6.077	Rise	clock
2	EW[0]	clock	6.067	6.067	Rise	clock
3	EW[1]	clock	6.037	6.037	Rise	clock

Figure 3.19: Temporização do bloco operativo.

3.2.3 Área Utilizada

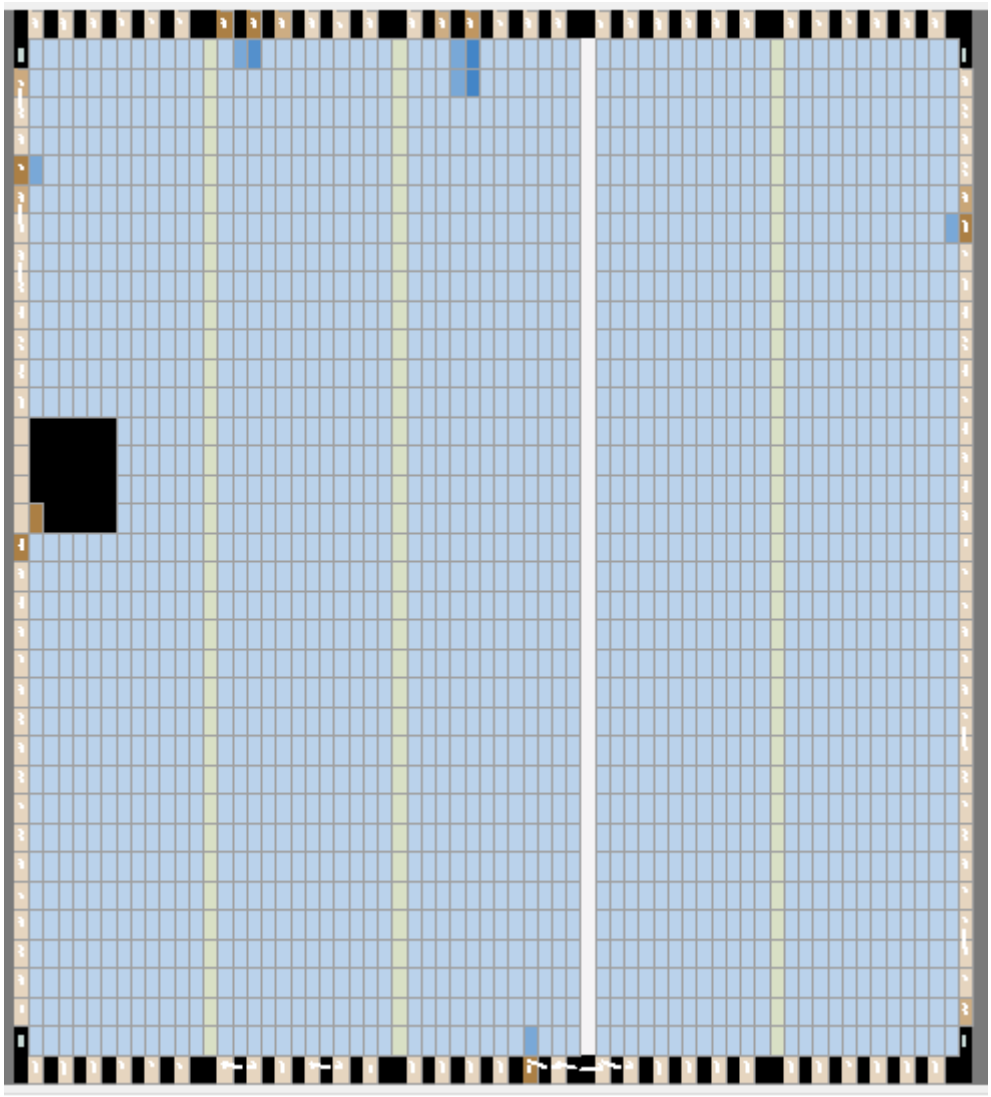

Figure 3.20: Área utilizada pelo bloco operativo.

3.2.4 Esquemático RTL

A figura 3.21 representa o esquemático (RTL) do bloco operativo do projeto. Nele, observa-se as entradas do bloco e as suas ligações com os demais componentes do projeto: muxes, registradores, somadores e comparadores.

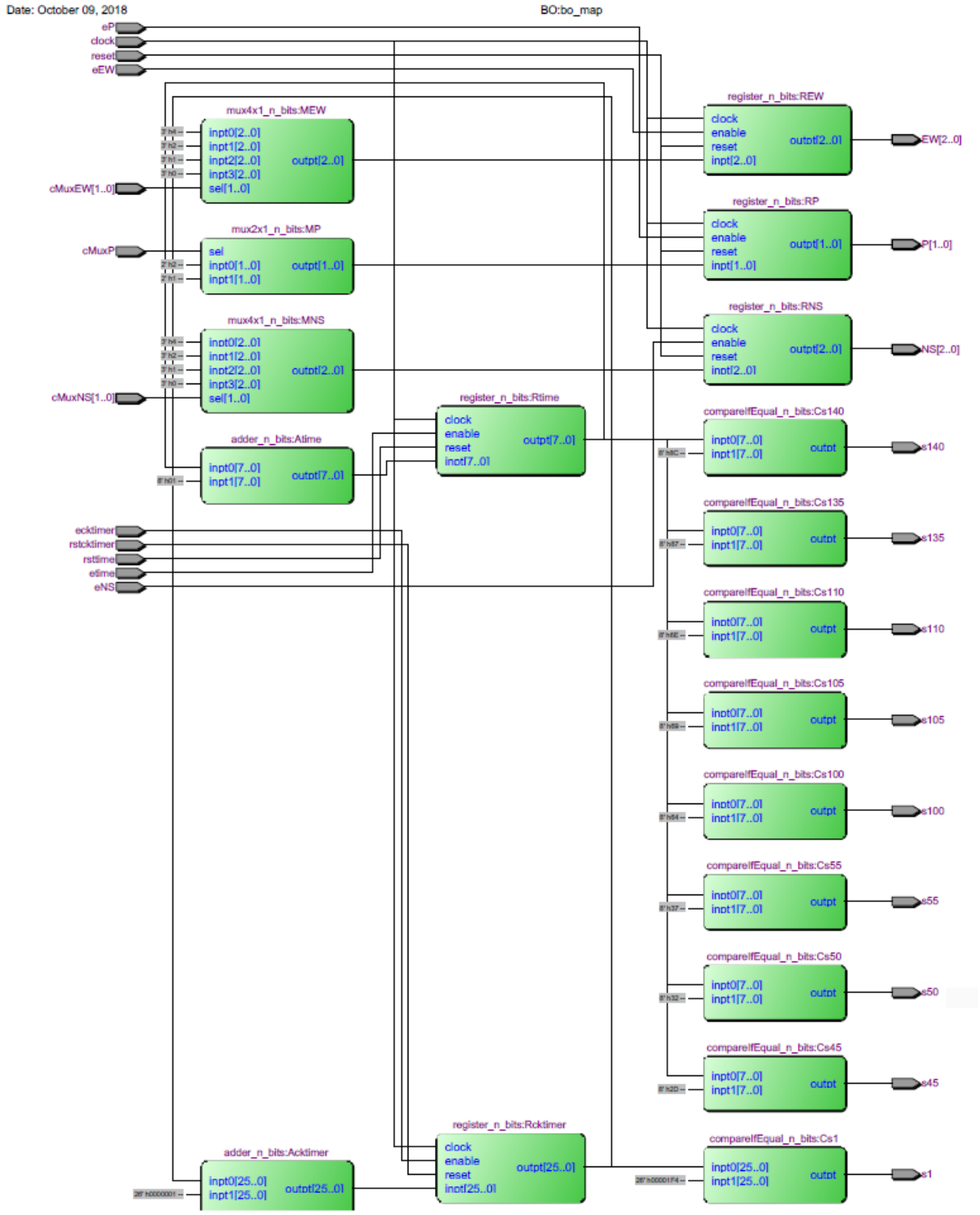

Figure 3.21: Esquemático RTL do bloco operativo.

3.3 Traffic Light

O componente “traffic_light_top” é a nossa entidade “top level”. Nela, será feito as devidas integrações do bloco operativo, bloco de controle. Este componente conta com o “clock”, servido para temporização e o “reset” como entradas, e os sinais “NS” e “EW”, de três bits cada, e “P”, de dois bits. Estes três últimos serão a codificação específica de cada semáforo, sendo verde, amarelo e vermelho quando obedecido às regras.

3.3.1 Código VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity traffic_light_top is
port(
	reset, clock: in std_logic;
	NS, EW: out std_logic_vector(2 downto 0);
	P: out std_logic_vector(1 downto 0)
);
end entity;
--
architecture archTop of traffic_light_top is

	component BC is
	port(
		-- control inputs
		clock, reset: in std_logic;
		s1, s45, s50, s55, s100, s105, s110, s135, s140: in std_logic;

		-- control outputs
		ecktimer, rstcktimer, rsttime, etime, eNS, eP, eEW: out std_logic;

		cMuxNS, cMuxEW: out std_logic_vector(1 downto 0);
		cMuxP: out std_logic
	);
	end component;

	component BO is
	port(
		-- operative inputs
		clock, reset: in std_logic;
		ecktimer, rstcktimer, rsttime, etime, eNS, eP, eEW: in std_logic;
		cMuxNS, cMuxEW: in std_logic_vector(1 downto 0);
		cMuxP : in std_logic;

		-- operative outputs
		s1, s45, s50, s55, s100, s105, s110, s135, s140: out std_logic;

		-- data outputs
		NS, EW: out std_logic_vector(2 downto 0);
		P: out std_logic_vector(1 downto 0)
	);
	end component;


	-- signal declaration
	signal sig_s1, sig_s45, sig_s50, sig_s55, sig_s100, sig_s105, sig_s110, sig_s135, sig_s140: std_logic;
	signal sig_ecktimer, sig_rstcktimer, sig_rsttime, sig_etime, sig_eNS, sig_eP, sig_eEW: std_logic;
	signal sig_cMuxNS, sig_cMuxEW: std_logic_vector(1 downto 0);
	signal sigcMuxP: std_logic;

	begin

	bc_map : BC PORT MAP(clock, reset,
				sig_s1, sig_s45, sig_s50, sig_s55, sig_s100, sig_s105, sig_s110, sig_s135, sig_s140,
				sig_ecktimer, sig_rstcktimer, sig_rsttime, sig_etime, sig_eNS, sig_eP, sig_eEW,
				sig_cMuxNS, sig_cMuxEW, sigcMuxP);

	bo_map : BO PORT MAP(clock, reset,
				sig_ecktimer, sig_rstcktimer, sig_rsttime, sig_etime, sig_eNS, sig_eP, sig_eEW,
				sig_cMuxNS, sig_cMuxEW, sigcMuxP,
				sig_s1, sig_s45, sig_s50, sig_s55, sig_s100, sig_s105, sig_s110, sig_s135, sig_s140,
				NS, EW,
				P);

end architecture;
```

3.3.2 Temporização

Na figura 3.22 explicita-se o tempo de carga, sendo a saída NS com maior tempo de subida e também de descida (6,597), seguido pelo tempo da carga P (6,383) e por último o tempo de carga de EW (6,361). Estes tempos tem uma pequena diferença entre si, porém devem ser estritamente respeitadas em simulações futuras.

Clock to Output Times				
Data Port	**Clock Port**	**Rise**	**Fall**	**Clock Edge**
NS[*]	clock	6,597	6,597	Rise
EW[*]	clock	6,361	6,361	Rise
P[*]	clock	6,383	6,383	Rise

Figure 3.22: Temporização do Traffic Light.

3.3.3 Área Utilizada

A figura 3.23 mostra os componentes utilizados da placa da altera para síntese do hardware do projeto do semáforo. Pode-se observar na simulação que menos de 1% de todo o hardware foi utilizado, totalizando 79 / 33.216 elementos lógicos, 79 / 33,216 funções combinacionais, 51 / 33,216 registradores lógicos dedicados, 51 registradores e 10/475 pinos (2%).

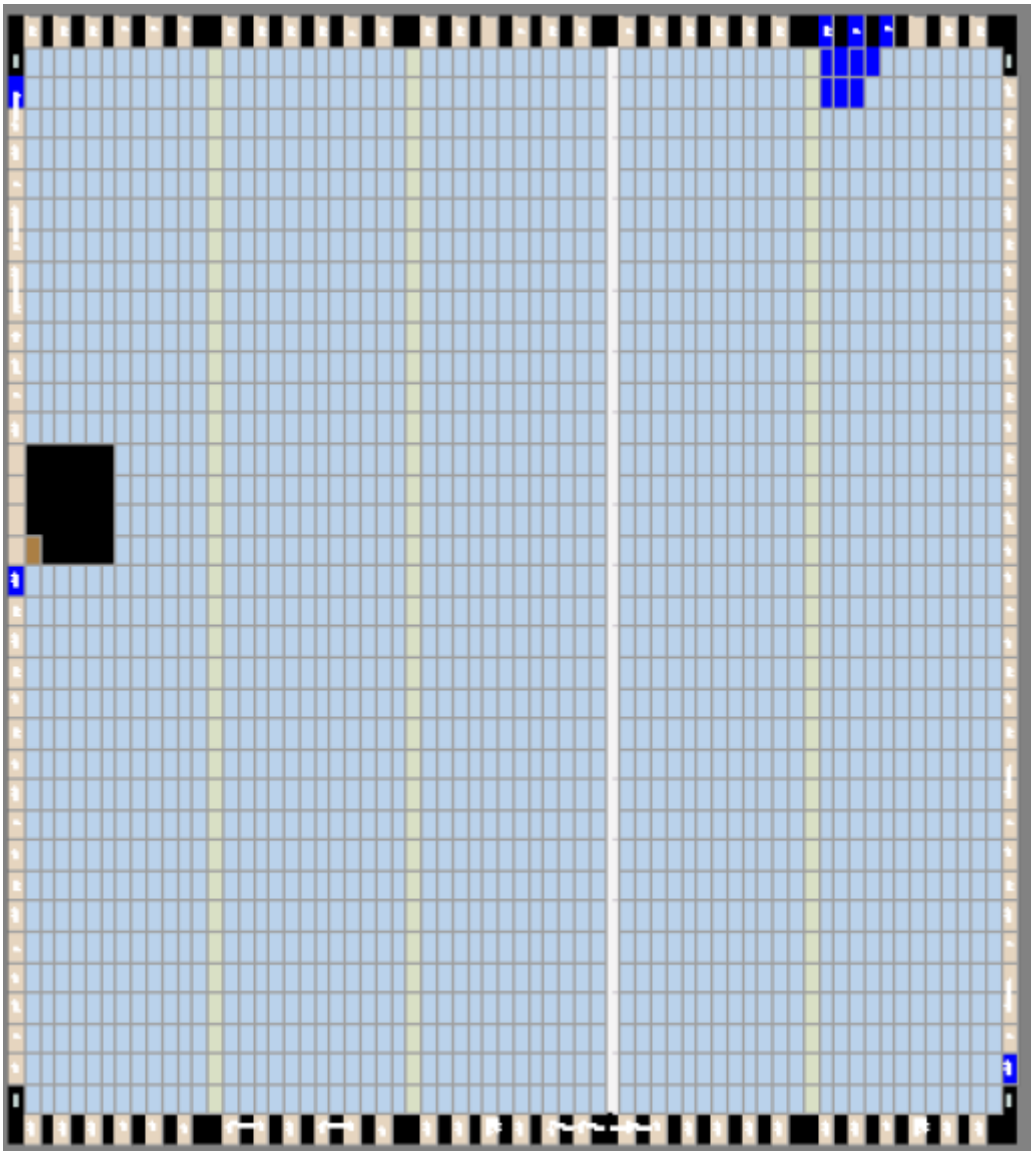

Figure 3.23: Componentes utilizados da placa da altera para síntese do hardware do projeto do sistema digital.

3.3.4 Esquemático RTL

Esquemático (RTL) da entidade top level do projeto. Nele pode-se observar as ligações dos componentes do Bloco Operativo e do Bloco de Controle como na figura 3.24.

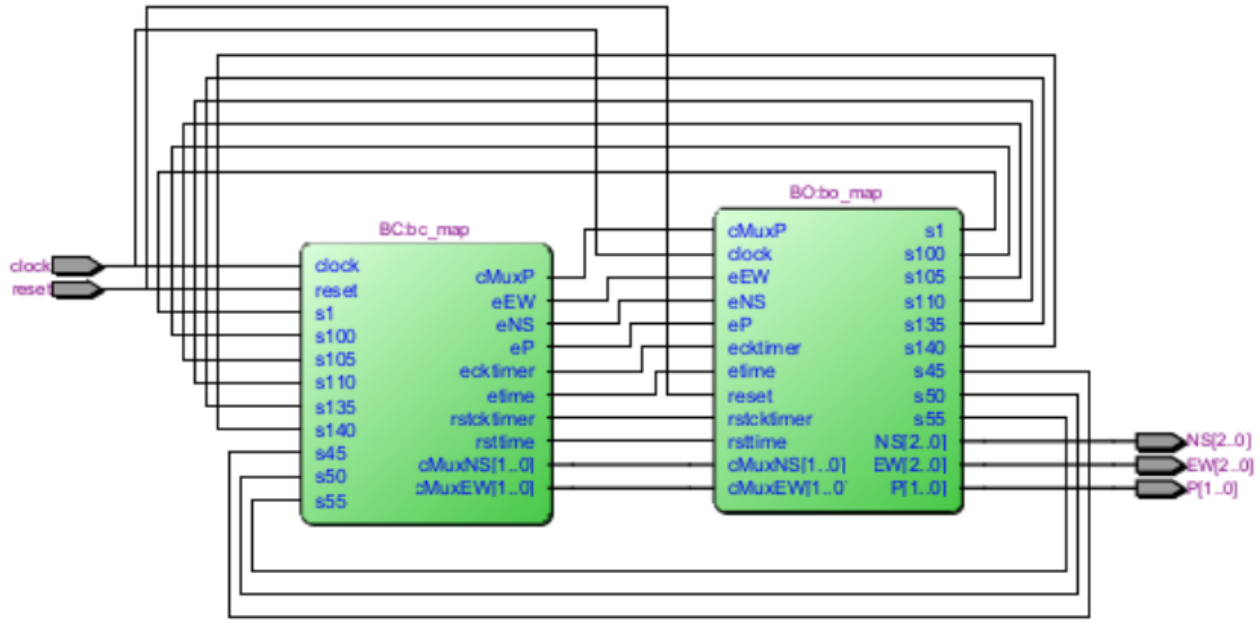

Figure 3.24: Esquemático RTL da entidade top level do projeto.

3.3.5 Tabela de Transição de Estados e FSM

A figura 3.25 explicita a máquina de transição de estados geradas automaticamente. Nela, podemos conferir se a lógica de próximo estado estão corretas e condizentes com a FSM projetada.

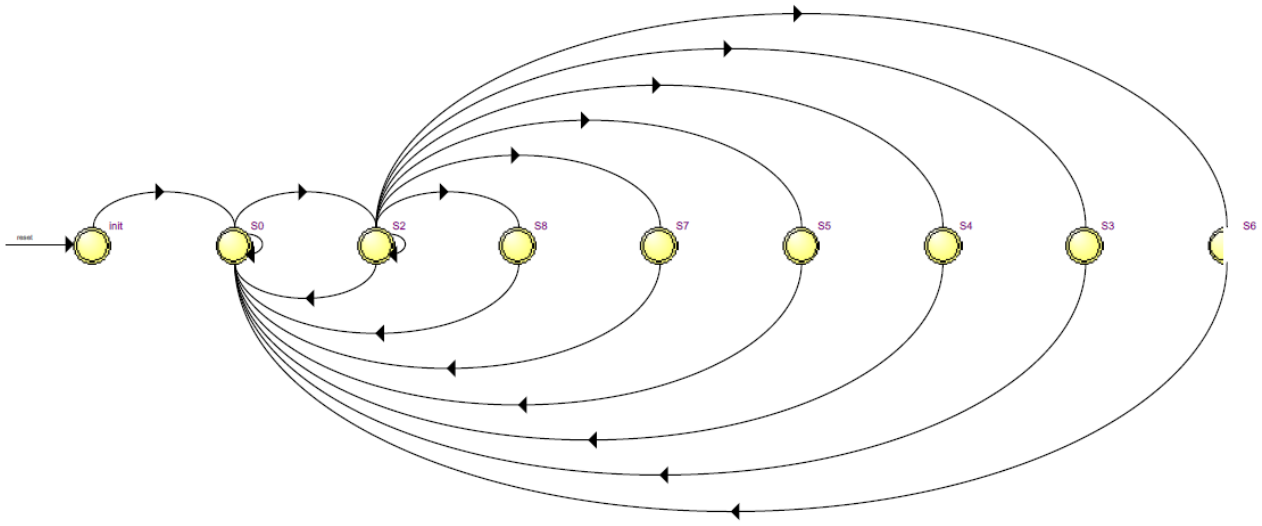

Figure 3.25: Máquina de transição de estados geradas automaticamente.

Na mesma janela também obtêm-se a descrição do motivo de transição, dada na figura 3.26. Exemplo: estando no estado S0, quando o sinal s1 valer ‘1’, troca-se de estado, partindo para o S2 e assim, obedecendo às regras determinadas, continua-se para as outras lógicas.

Source State	Destination State	Condition
init	S0	
S0	S0	(!s1)
S0	S2	(s1)
S2	S0	(!s45).(!s50).(!s55).(!s100).(!s105).(!s110).(!s135).(!s140) + (!s45).(!s50).(!s55).(!s100).(!s105).(!s110).(s135).(s140) + (!s45).(!s50).(!s55).(!s100).(!s105).(s110).(!s135).(s140) + (!s45).(!s50).(!s55).(!s100).(!s105).(s110).(s135) + (!s45).(!s50).(!s55).(!s100).(s105).(!s110).(s140) + (!s45).(!s50).(!s55).(!s100).(s105).(s110) + (!s45).(!s50).(!s55).(s100).(!s105).(!s110).(!s135).(s140) + (!s45).(!s50).(!s55).(s100).(!s105).(!s110).(s135) + (!s45).(!s50).(!s55).(s100).(!s105).(s110) + (!s45).(!s50).(!s55).(s100).(s105) + (!s45).(!s50).(s55).(!s100).(!s105).(!s110).(!s135).(s140) + (!s45).(!s50).(s55).(!s100).(!s105).(!s110).(s135) + (!s45).(!s50).(s55).(!s100).(!s105).(s110) + (!s45).(!s50).(s55).(!s100).(s105) + (!s45).(!s50).(s55).(s100) + (!s45).(s50).(!s55).(!s100). (!s110).(s140) + (!s45).(s50).(!s55).(!s100).(s110) + (!s45).(s50).(!s55).(s100) + (!s45).(s50).(s55) + (s45).(!s50).(!s55).(!s100).(!s105).(!s110).(!s135).(s140) + (s45).(!s50).(!s55).(!s100). (!s105).(!s110).(s135) + (s45).(!s50).(!s55).(!s100).(!s105).(s110) + (s45).(!s50).(!s55).(!s100).(s105) + (s45).(!s50).(!s55).(s100) + (s45).(!s50).(s55) + (s45).(s50).(!s55) + (s45).(s50).(s55).(!s100) + (s45).(s50).(s55).(s100).(!s105) + (s45).(s50).(s55).(s100).(s105).(!s110) + (s45).(s50).(s55).(s100).(s105).(s110).(!s135) + (s45).(s50).(s55).(s100).(s105).(s110).(s135).(!s140)
S2	S2	(s45).(s50).(s55).(s100).(s105).(s110).(s135).(s140)
S2	S3	(s45).(!s50).(!s55).(!s100).(!s105).(!s110).(!s135).(!s140)
S2	S4	(!s45).(!s50).(!s55).(!s100).(!s105).(!s110).(s135).(!s140) + (!s45).(!s50).(!s55).(!s100).(s105).(!s110).(!s140) + (!s45).(s50).(!s55).(!s100).(!s110).(!s140)
S2	S5	(!s45).(!s50).(s55).(!s100).(!s105).(!s110).(!s135).(!s140)
S2	S6	(!s45).(!s50).(!s55).(s100).(!s105).(!s110).(!s135).(!s140)
S2	S7	(!s45).(!s50).(!s55).(!s100).(!s105).(s110).(!s135).(!s140)
S2	S8	(!s45).(!s50).(!s55).(!s100).(!s105).(!s110).(!s135).(s140)
S3	S0	
S4	S0	
S5	S0	
S6	S0	
S7	S0	
S8	S0	

Figure 3.26: Descrição do motivo de transição de estados.

4 Validação

Tendo em mente os componentes descritos acima, podemos validar suas funcionalidades utilizando a ferramenta ModelSim. Modelsim permite simular o funcionamento dos componentes programados. Para tal, informamos diferentes inputs e conferimos se o output gerado é igual ao output real que é esperado do sistema digital. O range de cada simulação pode ser observado no canto esquerdo inferior das imagens de simulação.

4.1 Validação do Bloco Operativo

Nota-se ao observar a figura 4.1 temos que ao mudar os sinais vindos do bloco de controle (inputs), ele alteram os outputs NS, EW e P, que presentam, respectivamente as luzes das ruas norte-sul, leste-oeste e de pedestres. O contorno amarelo à esquerda representa a luz verde na pista norte-sul, o contorno laranja da pista leste-oeste representa a luz vermelha na pista leste-oeste, e o contorno vermelha representa a luz vermelha da faixa de pedestres. Nota-se também que o ciclo recomeça a partir do instante 200ns, simbolizados por contornos na parte esquerda da imagem.

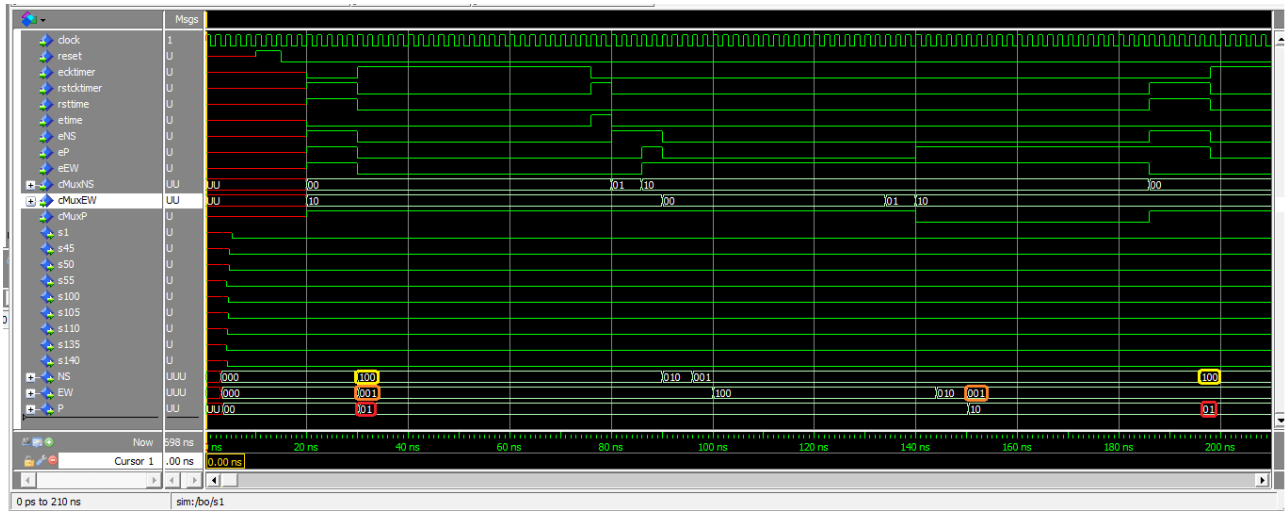

Figure 4.1 Simulação de ondas do bloco operativo no ModelSim

Estímulos usados para gerar as ondas do bloco operativo no simulador:

```
force clock 1 0, 0 {1 ns} -r 2 ns
force reset 1 10 ns
force reset 0 15 ns

# Init
force ecktimer 0 20 ns
force rstcktimer 1 20 ns
force rsttime 1 20 ns
force etime 0 20 ns
force eNS 1 20 ns
force cMuxNS 00 20 ns
force cMuxP 1 20 ns
force eP 1 20 ns
force cMuxEW 10 20 ns
force eEW 1 20 ns

# S0
force ecktimer 1 30 ns
force rstcktimer 0 30 ns
force rsttime 0 30 ns
force etime 0 30 ns
force eNS 0 30 ns
force eP 0 30 ns
force eEW 0 30 ns

# S2
force ecktimer 0 76 ns
force rstcktimer 1 76 ns
force rsttime 0 76 ns
force etime 1 76 ns
force eNS 0 76 ns
force eP 0 76 ns
```

```
force eEW 0 76 ns

# S3
force ecktimer 0 80 ns
force rstcktimer 0 80 ns
force rsttime 0 80 ns
force etime 0 80 ns
force eNS 1 80 ns
force cMuxNS 01 80 ns
force eP 0 80 ns
force eEW 0 80 ns

# S4
force ecktimer 0 86 ns
force rstcktimer 0 86 ns
force rsttime 0 86 ns
force etime 0 86 ns
force eNS 1 86 ns
force cMuxNS 10 86 ns
force cMuxP 1 86 ns
force eP 1 86 ns
force cMuxEW 10 86 ns
force eEW 1 86  ns

# S5
force ecktimer 0 90 ns
force rstcktimer 0 90 ns
force rsttime 0 90 ns
force etime 0 90 ns
force eNS 0 90 ns
force eP 0 90 ns
force cMuxEW 00 90 ns
force eEW 1 90 ns

# S6
force ecktimer 0 136 ns
force rstcktimer 0 136 ns
force rsttime 0 136 ns
force etime 0 136 ns
force eNS 0 136 ns
force eP 0 136 ns
force cMuxEW 01 136 ns
force eEW 1 136 ns

# S7
force ecktimer 0 140 ns
force rstcktimer 0 140 ns
force rsttime 0 140 ns
force etime 0 140 ns
force eNS 0 140 ns
force cMuxP 0 140 ns
```

```
force eP 1 140 ns
force cMuxEW 10 140 ns
force eEW 1 140 ns

# S8
force ecktimer 0 186 ns
force rstcktimer 1 186 ns
force rsttime 1 186 ns
force etime 0 186 ns
force eNS 1 186 ns
force cMuxNS 00 186 ns
force eP 1 186 ns
force cMuxP 1 186 ns
force eEW 0 186 ns

# S0
force ecktimer 1 198 ns
force rstcktimer 0 198 ns
force rsttime 0 198 ns
force etime 0 198 ns
force eNS 0 198 ns
force eP 0 198 ns
force eEW 0 198 ns
```

4.1.1 Adder n Bits

A figura 4.2 monstra a validação do componente adder n bits. Nota-se, por exemplo, que 00000001 + 00000001 = 00000010 em binário, ou seja, 1 + 1 = 2 em decimal.

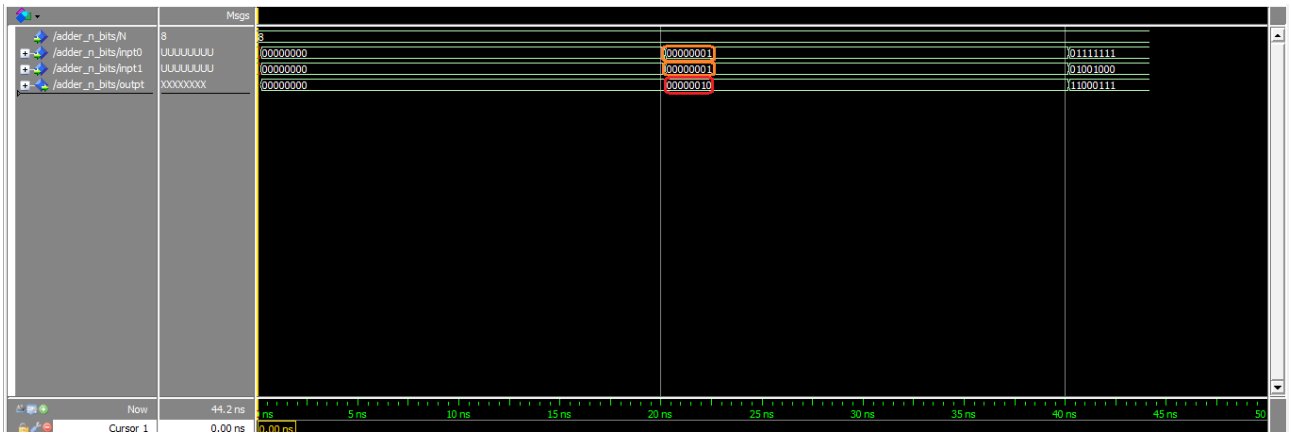

Figure 4.2: Simulação de ondas do adder n bits no ModelSim:

Estímulos usados para gerar as ondas do adder n bits no simulador:

```
force inpt0 00000000 0 ns
force inpt1 00000000 0 ns
force inpt0 00000001 20 ns
force inpt1 00000001 20 ns
```

```
force inpt0 01111111 40 ns
force inpt1 01001000 40 ns
```

4.1.2 Compare if Equal

A figura 4.3 mostra a validação do componente compare if equal, observa-se que ao dar como entrada 1 os valores 01111111 e entrada 2 os valores 01001011, recebemos o output 0, isto é, tais inputs são diferentes entre si. Porém, ao entrar com o valor 01010111 e 01010111, percebemos que o output gerado é 1, ou seja, ou valores são iguais entre si.

Figure 4.3: Simulação de ondas do compare if equal no ModelSim:

Estímulos usados para gerar as ondas do compare if equal no simulador:

```
force inpt0 00000000 0 ns
force inpt1 00000000 0 ns
force inpt0 00000001 20 ns
force inpt1 00000001 20 ns
force inpt0 01111111 40 ns
force inpt1 01001000 40 ns
force inpt0 01010111 60 ns
force inpt1 01010111 60 ns
```

4.1.3 Register

A figura 4.4 mostra o funcionamento do componente register. Percebe-se que ao entrar com um valor 00001111 e o enable estiver desabilitado, nada ocorrerá e o output ficará o mesmo. Porém, se o enable estiver ativo, significando que o registrador admite um novo valor, seu valor será armazenado, oferecendo-o como saída. Observa-se também que ao habilitar o input reset, o registrador apaga seu output e currentState, para e substitui-os por 00000000.

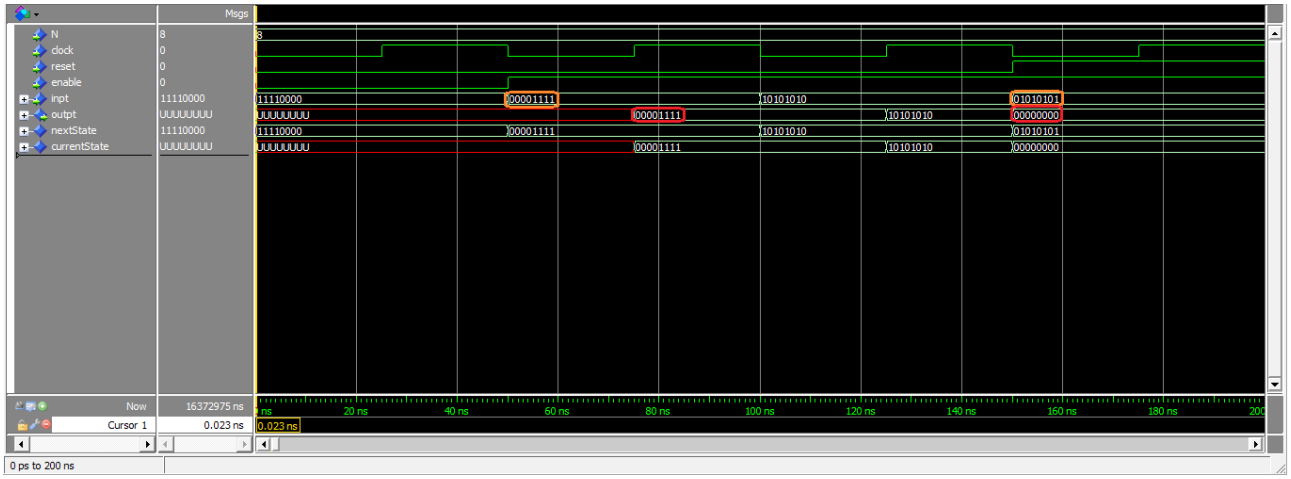

Figure 4.4: Simulação de ondas do register no ModelSim.

Estímulos usados para gerar as ondas do register no simulador:

```
force clock 0 0 ns, 1 25 ns -r 50 ns
force reset 0 0 ns
force enable 0 0 ns
force inpt 11110000 0 ns
force reset 0 50 ns
force enable 1 50 ns
force inpt 00001111 50 ns
force inpt 10101010 100 ns
force reset 1 150  ns
force enable 1 150 ns
force inpt 01010101 150 ns
```

4.1.4 Mux4x1

A figura 4.5 demonstra o funcionamento do Mux4x1. Nota-se, pelo contorno amarelo, que ao entrar como input na variável inpt0 o valor 111 e em seguida selecionar o valor, no select, 00 (contorno laranja), o valor 111 é retornado (contorno vermelho). Neste mux o valor, no select, 00 retorna o inpt0, 01 retorna o inpt1, 10 retorna o inpt2 e 11 retorna o inpt3.

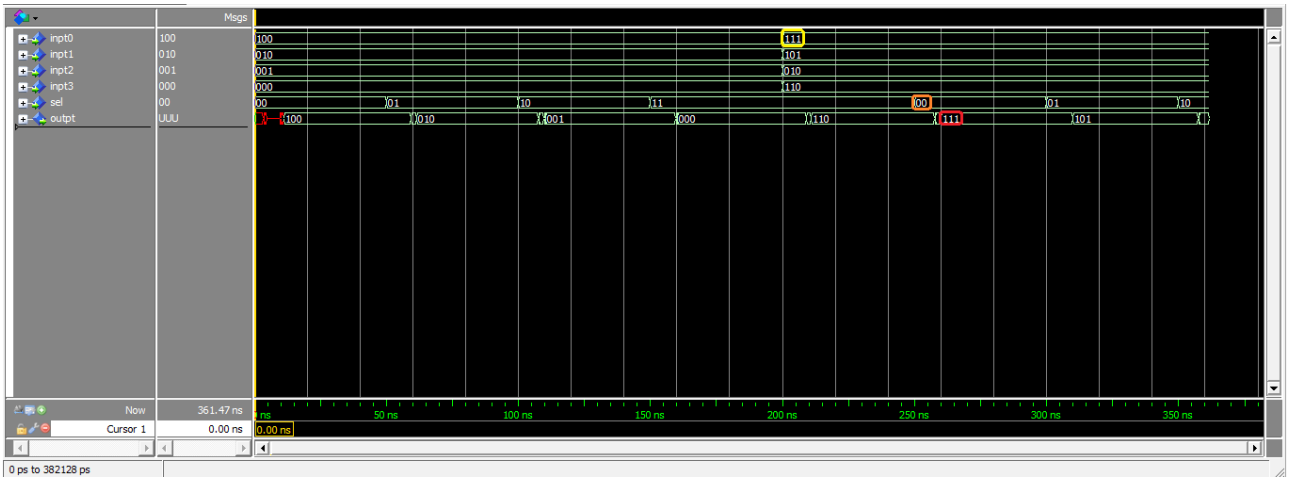

Figure 4.5: Simulação de ondas do Mux4x1 no ModelSim.

Estímulos usados para gerar as ondas do Mux4x1 no simulador:

```
force inpt0 100 0 ns
force inpt1 010 0 ns
force inpt2 001 0 ns
force inpt3 000 0 ns
force sel 00 0 ns
force sel 01 20 ns
force sel 10 40 ns
force inpt0 111 60 ns
force inpt1 101 60 ns
force inpt2 010 60 ns
force inpt3 110 60 ns
force sel 00 80 ns
force sel 01 100 ns
force sel 10 120 ns
```

4.1.5 Mux 2x1

A figura 4.6 demonstra o funcionamento do Mux2x1. Nota-se, pelo contorno amarelo, que ao entrar como input na variável inpt0 o valor 10, e, em seguida selecionar o valor, no select, 0 (flecha laranja), o valor 10 é retornado (contorno vermelho). Neste mux o valor, no select, 0 retorna o inpt0 e o valor 1 retorna o inpt1.

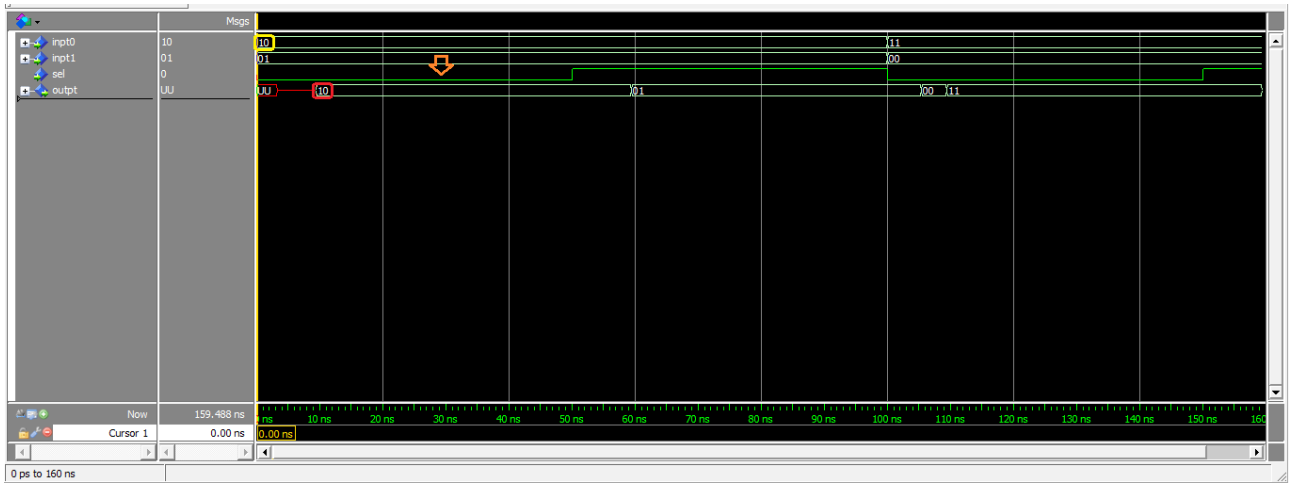

Figure 4.6: Simulação de ondas do Mux2x1 no ModelSim.

Estímulos usados para gerar as ondas do Mux2x1 no simulador:

```
force inpt0 10 0 ns
force inpt1 01 0 ns
force sel 0 0 ns
force sel 1 50 ns
force inpt0 11 100 ns
force inpt1 00 100 ns
force sel 0 100 ns
force sel 1 150 ns
```

4.2 Validação do Bloco de Controle

A figura 4.7 simula os sinais gerados pelo bloco de controle e a alteração dos outputs gerados em decorrência desta simulação. Como ele é responsável pelo gerenciamento de decisões, a partir dos sinais advindos do bloco operativo e seguindo a lógica de próximo estado prevista na figura 3.26.

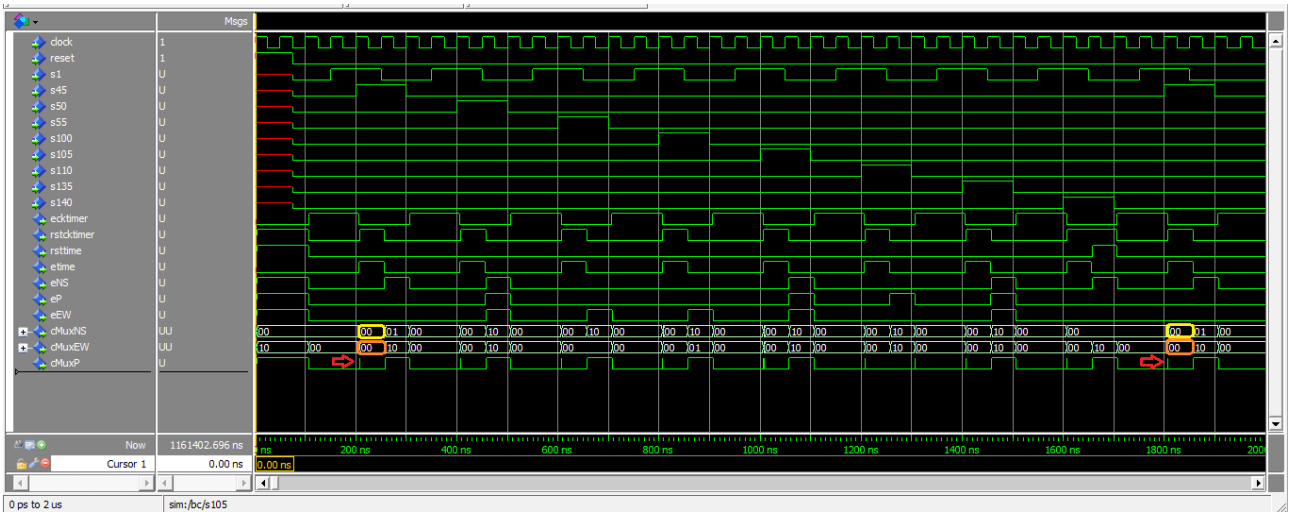

Figure 4.7: Simulação de ondas do Mux2x1 no ModelSim.

Estímulos usados para gerar as ondas do Mux2x1 no simulador:

```
force clock 1 0, 0 {25 ns} -r 50 ns
force reset 1 0 ns
force reset 0 75 ns

force s1 0 75 ns
force s45 0 75 ns
force s50 0 75 ns
force s55 0 75 ns
force s100 0 75 ns
force s105 0 75 ns
force s110 0 75 ns
force s135 0 75 ns
force s140 0 75 ns

force s1 1 150 ns
force s45 1 200 ns
force s1 0 250 ns
force s45 0 300 ns

force s1 1 350 ns
force s50 1 400 ns
force s1 0 450 ns
force s50 0 500 ns

force s1 1 550 ns
force s55 1 600 ns
```

```
force s1 0 650 ns
force s55 0 700 ns

force s1 1 750 ns
force s100 1 800 ns
force s1 0 850 ns
force s100 0 900 ns

force s1 1 950 ns
force s105 1 1000 ns
force s1 0 1050 ns
force s105 0 1100 ns

force s1 1 1150 ns
force s110 1 1200 ns
force s1 0 1250 ns
force s110 0 1300 ns

force s1 1 1350 ns
force s135 1 1400 ns
force s1 0 1450 ns
force s135 0 1500 ns

force s1 1 1550 ns
force s140 1 1600 ns
force s1 0 1650 ns
force s140 0 1700 ns

force s1 1 1750 ns
force s45 1 1800 ns
force s1 0 1850 ns
force s45 0 1900 ns
```

4.3 Validação do Traffic Light

A imagem 4.8 mostra o funcionamento do componente top level do sistema digital sem a simulação de reset. Foi usado 500 clocks para simulação, ao invés de 50000000, a fim de simplificar a simulação. Em relação a parte esquerda da imagem nota-se, a partir do contorno amarelo, que o semáforo norte-sul recebe o valor 100 que, como descrito em seções anteriores, representa a cor verde. O semáforo leste-oeste recebe o valor 001 (vermelho), representado pelo contorno laranja, e o semáforo de pedestres recebe o valor 01 (vermelho), representado pelo contorno vermelho. Após aproximadamente 1200000 ns o semáforo norte-sul passa de verde (100) para amarelo (010) e em seguida para vermelho (001). Após 5 segundos (na prática) ou aproximadamente 1000000 ns (na abstração simulada) o semáforo leste-oeste passa de 001 (vermelho) para 100 (verde) e em seguida para amarelo (010) e vermelho (001, contorno laranja da direita). Após 5 segundos, o semáforo de pedestres torna-se verde (10) e em seguida vermelho (01, contorno vermelho da direita). Os contornos da direita são utilizados para demonstrar o ciclo do semáforo, no qual os sinais começam a se repetir, ou seja, o sistema digital está

funcionando corretamente. Em seguida será apresentado uma simulação usando o input reset.

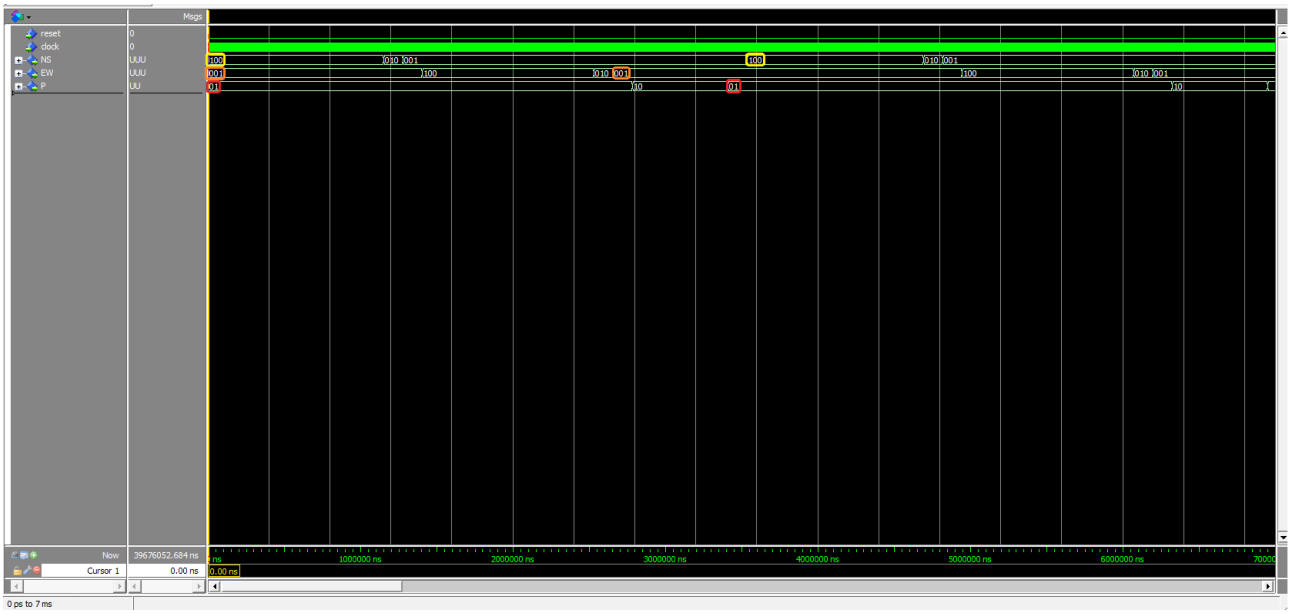

Figure 4.8: Simulação de ondas do traffic light no ModelSim, sem reset.

Estímulos usados para gerar as ondas do Mux2x1 no simulador, sem reset:

```
force clock 0 0 ns, 1 25 ns -r 50 ns
force reset 0 0 ns
```

A figura 4.9 mostra o funcionamento idêntico a figura anterior, porém, utiliza-se o input do reset para simular o reset do sistema. Observa-se, a partir da flecha azul, que o reset foi ativado e que, quando ativado, os inputs NS, EW e P alteram seus valores para os valores iniciais, e o semáforo continua funcionando corretamente.

Figure 4.9: Simulação de ondas do traffic light no ModelSim, com reset.

Estímulos usados para gerar as ondas do Mux2x1 no simulador, com reset:

```
force clock 0 0 ns, 1 25 ns -r 50 ns
force reset 0 0 ns
force reset 1 3000000 ns
force reset 0 3000001 ns
```