



UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA

REGISTER-TRANSFER LEVEL DESIGN
SISTEMA DE CONTROLE DE LAVADORA DE ROUPAS

Breno Augusto Silva Muniz

Gabriel de Oliveira Groppo

Matheus Marcondes de Oliveira

Belo Horizonte – MG

2024

SUMÁRIO

1. INTRODUÇÃO	3
2. MODELAGEM DO SISTEMA	6
2.1. PERIFÉRICOS	6
2.2. DESIGN DO PROCESSADOR	9
2.2.1. FSM DE ALTO NÍVEL	11
2.2.2. DESCRIÇÃO DA FSM DE ALTO NÍVEL	13
2.2.3. DATAPATH	15
2.3. DESIGN DO BLOCO DE CONTROLE	44
2.3.1. CONEXÃO DATAPATH – CONTROLADORA	44
2.3.2. FSM DE BAIXO NÍVEL DA CONTROLADORA	45
2.3.3. DESCRIÇÃO DA FSM DE BAIXO NÍVEL	47
2.3.4. ARQUITETURA DO BLOCO DE CONTROLE	48
2.4. DESIGN E INTEGRAÇÃO DO SISTEMA	49
2.4.1. CÓDIGOS VHDL	49
2.4.2. RTL VIEWER	50
2.4.3. STATE MACHINE VIEWER	53
2.4.4. MEMÓRIA FLASH	56
3. SIMULAÇÃO DO SISTEMA	57
3.1. SIMULAÇÃO VIRTUAL	57
3.2. SIMULAÇÃO NO KIT FPGA	66
4. CONCLUSÃO	67
5. REFERÊNCIAS	68
6. APÊNDICE A – REPOSITÓRIO	69

1. INTRODUÇÃO

O presente relatório descreve a modelagem de uma Máquina de Estados Finitos (FSM - Finite State Machine) para o controle de uma lavadora de roupas. Este tipo de design foi concebido de forma a garantir que o sistema opere de maneira organizada, eficiente e previsível, executando as etapas de lavagem conforme programado.

A FSM utiliza uma série de estados, transições e condições para representar o comportamento da lavadora durante o processo de lavagem. De forma resumida, são eles:

- 0 **Start:** Estado inicial onde a máquina aguarda a seleção do programa de lavagem e a inserção das entradas necessárias (P, N, ME, EE). Avança para o estado seguinte quando o usuário acionar o botão start.
- 1 **Programas (P: rápida, normal, intenso, enxágue + centrifugação):** quatro estados distintos que representam diferentes programas de lavagem, com diferentes tempos para cada etapa, configuração do motor para baixa ou alta rotação, número de molhos e enxágues necessários, número de etapas para cada programa. Todos esses dados são lidos de uma memória flash.
 - **P rápido:** tempo de lavagem, de molho, de enxágue e de centrifugação curtos. Motor agita em baixa rotação durante a lavagem e enxágue. Molho e enxágue única vez cada. São seis etapas.
 - **P normal:** tempo de lavagem, de molho, de enxágue e de centrifugação moderados. Motor em baixa rotação. Molho único e dois enxágues. São oito etapas.
 - **P intensa:** tempo de lavagem estendido. Tempo de molho, enxágue e de centrifugação moderados. Motor em alta rotação. São dois molhos e dois enxágues. Ao todo são 10 etapas.
 - **P enxágue + centrifugação:** pula a etapa de lavagem, logo não tem molho. Tempo de enxágue e centrifugação moderados. Motor em baixa rotação durante etapa de agitação do enxágue. São dois enxágues por padrão.

Obs: o número de etapas pode alterar em cada programa se o usuário acionar o botão para molho extra, ou enxágue extra, ou ambos. Por padrão, durante a etapa de centrifugação o motor é configurado para alta rotação em sentido único.

- 2 Encher:** são dois estados distintos. Isso se deve pelo uso de válvulas de duplo comando. A máquina enche de água até atingir o necessário de acordo com o nível selecionado (N). O pressostato (PS) indica o nível atual de água:
- **Encher (válvula 1):** nesse estado a água é condicionada para o dispenser que contém o sabão e alvejante. Portanto, a válvula é acionada de acordo para guiar a água para o reservatório correto. Faz parte da etapa de lavagem.
 - **Encher (válvula 2):** nesse estado a água é condicionada para o dispenser que contém o amaciante. Portanto, a válvula é acionada de acordo para guiar a água para o reservatório correto. Faz parte da etapa do enxágue.
- 3 Atualiza:** 4 estados distintos auxiliares (um para cada etapa: lavagem, molho, enxágue e centrifugação) responsáveis por resetar o temporizador e fazer a configuração de um tempo distinto respectivo daquela etapa, além de atualizar a etapa em que a máquina se encontra:
- Atualiza LV (estado lavagem): reseta o timer e atualiza a etapa. Seta o tempo a ser decorrido: TL.
 - Atualiza ML (estado molho): reseta o timer e atualiza a etapa. Seta o tempo a ser decorrido: TM.
 - Atualiza EN (estado enxágue): reseta o timer e atualiza a etapa. Seta o tempo a ser decorrido: TE.
 - Atualiza CE (estado centrifugação): reseta o timer e atualiza a etapa. Seta o tempo a ser decorrido: TC.
- 4 Agitação:** são dois estados distintos para a máquina executar corretamente a etapa de agitação. Esse processo faz a agitação do rotor tanto para a lavagem quanto para o enxágue. O temporizador está setado para TL se estiver durante a lavagem ou para TE se estiver durante o enxágue, então irá alternar entre horário e anti-horário até que TL ou TE termine.
- **Agitar horário:** a saída faz com que o motor gire em sentido horário independente da rotação estar configurada para alta ou baixa.
 - **Agitar anti-horário:** a saída faz com que o motor gire em sentido anti-horário independente da rotação estar configurada para alta ou baixa.

- 5 **Molho:** neste estado o motor é desligado e o tempo necessário para o molho é disparado, ou seja, o temporizador está setado para TM. Quando esse tempo é atingido, sai desse estado e prossegue com o processo.
- 6 **Drenagem:** Estado onde a água é drenada da lavadora. Fica nesse nível enquanto o sensor d'água não sinalizar ausência de água.
- 7 **Centrifugação:** Estado onde a roupa é centrifugada para remover o excesso de água. O temporizador está configurado para TC, o motor está em alta rotação para o sentido horário. Quando o tempo para centrifugação terminar, vá para o estado final.
- 8 **Fim:** Estado final que indica o término do ciclo de lavagem.

Cada estado possui transições condicionais baseadas em entradas específicas e temporizadores (TE, TM, TL, TC) que controlam o tempo de enxágue, molho, lavagem e centrifugação, respectivamente. As entradas são descritas por variáveis como P (programa de lavagem), N (nível de água), ME (molho extra) e EE (enxágue extra).

O comportamento detalhado da FSM busca garantir que a lavadora execute as etapas de forma sequencial e lógica, ajustando-se conforme os sensores de água e as condições temporais. Este controle preciso é fundamental para otimizar a eficiência da lavagem e assegurar mínimas interferências no processo.

2. MODELAGEM DO SISTEMA

2.1. PERIFÉRICOS

Foram determinados todos os periféricos com os quais o processador deve interagir para o funcionamento da lavadora. Bem como, entradas, saídas e protocolos de comunicação. A figura 1 a seguir mostra o diagrama de blocos do sistema:

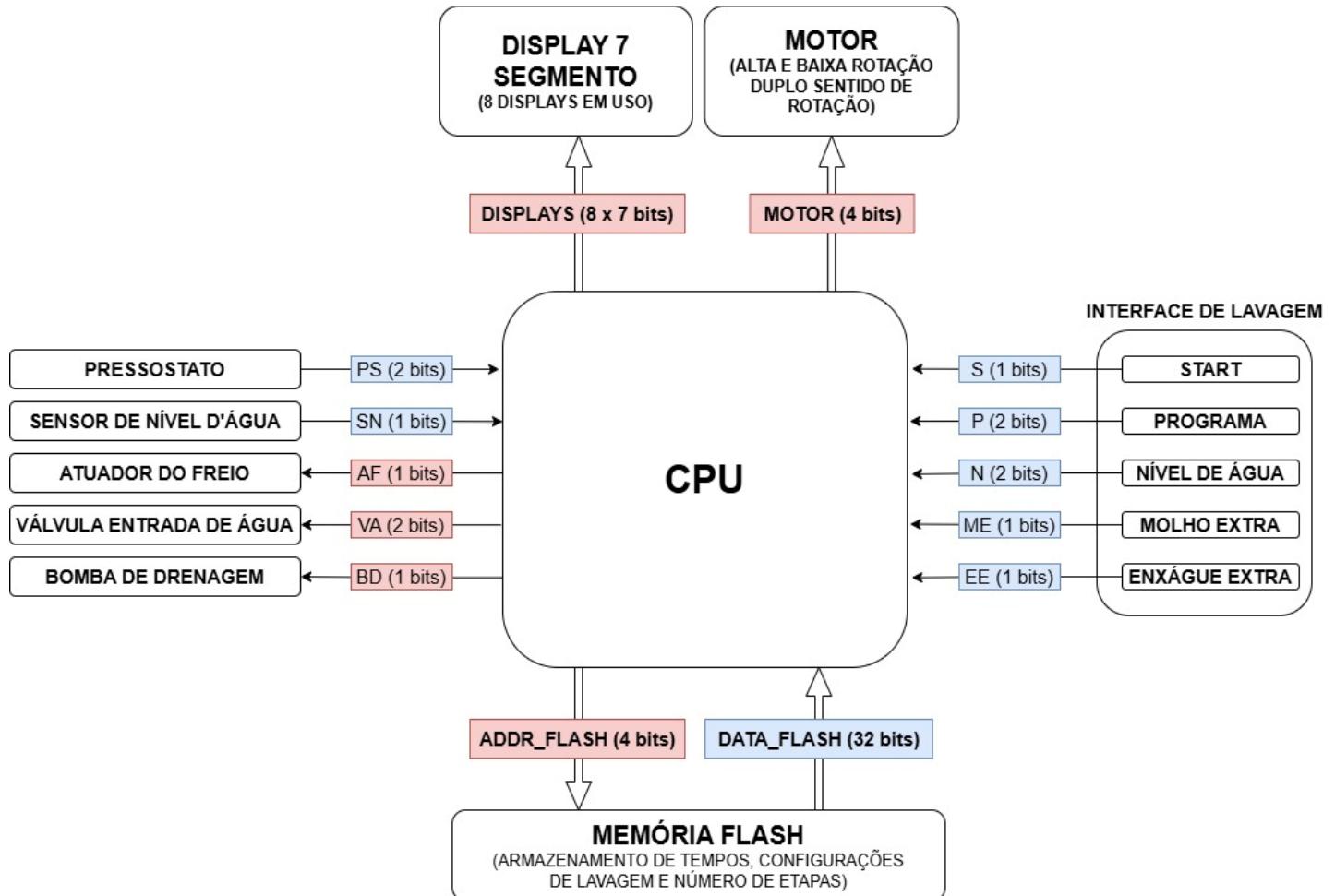


Figura 1 – Modelagem dos blocos do sistema

A seguir serão especificadas as funções, entradas e saídas de cada um dos periféricos, a fim de projetar-se o processador a partir de tais especificações.

SENSOR DE NÍVEL D'ÁGUA

Descrição: Indica se ainda há umidade considerável no interior do tanque.

Entradas: N/A

Saídas: Variável “SN” de 1 bit, ativa em nível lógico 1, que indica presença de água no tanque.

PRESSOSTATO

Descrição: Comunica ao sistema o nível de água no interior do tanque.

Entradas: N/A

Saídas: Variável “PS” de 2 bits, que indicam qual o nível d'água no tanque.

ATUADOR DO FREIO

Descrição: Quando não está acionado, ativa o freio da rotação do tanque e libera rotação do rotor. Quando acionado, libera rotação do tanque.

Entradas: Variável “AF”, ativa em nível lógico 1.

Saídas: N/A

BOMBA DE DRENAGEM

Descrição: Quando acionada, executa a drenagem da água de dentro do tanque.

Entradas: Variável “BD”, ativa em nível lógico 1.

Saídas: N/A

VÁLVULA DE ENTRADA D'ÁGUA

Descrição: Uma válvula com duplo comando, possui duas passagens de água distintas, a depender do estado. Valvula1 passa pela porta de sabão, quando o programa está no processo da Lavagem. Valvula2 passa pela porta de amaciante, quando o programa está no processo do Enxágue.

Entradas: Variável “VALV_AGUA” de 2 bits. Cada bit aciona a válvula correspondente.

Saídas: N/A

INTERFACE DE LAVAGEM

Descrição: Interface com o usuário para que sejam definidos os parâmetros da lavagem.

Entradas: N/A

Saídas:

- Variável “EE” de 1 bit, que indica um ciclo extra de enxágue;
- Variável “ME” de 1 bit, que indica um ciclo extra de molho;
- Variável “P” de 2 bits, que indica o programa a ser executado;
- Variável “N” de 2 bits, que indica o nível de água desejado;
- Variável “S” de 1 bit, que inicia o programa nas configurações definidas.

DISPLAYS

Descrição: Conjunto de 8 displays de 7 segmentos que compõe a interface do usuário.

Entradas: Variáveis D0 a D7 de 7 bits cada, que controlam os 8 displays.

Saídas: N/A

MOTOR

Descrição: Motor responsável pela rotação do tanque ou do rotor, a depender da configuração do elemento ATUADOR DO FREIO. Atua em 4 diferentes regimes de operação:

Entradas: Barramento “MOTOR_SAIDA” de 4 bits, onde cada um em nível lógico 1 indica um diferente regime de operação, conforme descritos abaixo:

- [0] Rotação em baixa potência;
- [1] Rotação em alta potência;
- [2] Rotação horária;
- [3] Rotação anti-horária.

Saídas: N/A

MEMÓRIA FLASH DAS INFORMAÇÕES DE LAVAGEM

Descrição: Está armazenado informações a respeito de cada programa de lavagem. As quais estão o tempo da lavagem, do molho, do enxágue e da centrifugação, o número de molhos e enxáguas que devem ocorrer ao longo do processo, além do número de etapas gasto por cada programa.

Entradas: Barramento de 16 bits chamado de “ADDR_FLASH”. Serve para posicionar a memória no endereço correto na hora da leitura. Apesar do tamanho, estará registrado dados apenas nos 4 primeiros endereços, respectivos de cada lavagem. Este tamanho permite alterações futuras em caso de inserção de novos programas de lavagem.

Saídas: Barramento de 32 bits chamado de “DATA_FLASH”. Esse barramento será dividido internamente no caminho de dados para 7 registradores diferentes. A ordem dessa disposição fica da seguinte forma:

- De [0] a [5] do barramento: 6 bits para o registrador TL, o qual guardará o tempo de agitação para a lavagem referente ao programa escolhido.
- De [6] a [11] do barramento: 6 bits para o registrador TE, o qual guardará o tempo de agitação para o enxágue referente ao programa escolhido.
- De [12] a [17] do barramento: 6 bits para o registrador TM, o qual guardará o tempo para o molho referente ao programa escolhido.
- De [18] a [23] do barramento: 6 bits para o registrador TC, o qual guardará o tempo para a centrifugação referente ao programa escolhido.
- De [24] a [25] do barramento: 2 bits para o registrador NM, o qual guardará o número de molhos necessários para aquele programa.
- De [26] a [27] do barramento: 2 bits para o registrador NE, o qual guardará o número de enxáguas necessários para aquele programa.
- De [28] a [31] do barramento: 4 bits para o registrador NUM_ETAPAS, o qual guardará o número de etapas para executar aquele programa.

2.2. DESIGN DO PROCESSADOR – METODOLOGIA REGISTER-LEVEL TRANSFER

2.2.1. FINITE STATE MACHINE DE ALTO NÍVEL

A tabela 1 abaixo mostra as entradas, saídas e registradores necessários. A descrição das entradas e saídas foi feita na seção 2.1. A figura 2 mostra a FSM de alto nível que o processador deverá implementar. Em sequência, a tabela 3 mostra o que acontece com cada saída em cada estado. Foi utilizada a convenção de que saídas não explicitadas na tabela estão em nível lógico 0. Por fim, é explicado em mais detalhes o funcionamento da FSM na seção 2.2.2.

ENTRADAS	SAÍDAS	REGISTRADORES
S (botão start)	MOTOR	PROG (armazena P)
P (programa de lavagem)	VALV_AQUA	NIVEL (armazena N)
N (seleção do nível de água)	ADDR_FLASH (endereço da configuração requerida para a memória)	MOTOR (armazena 4 saídas da controladora)
ME (molho extra)	BD (ativa a bomba de drenagem)	VALVULA (armazena 2 saídas da controladora)
EE (enxágue extra)	AF (ativa o atuador do freio)	ADDR (armazena o endereço da configuração requerida)
PS (sinal do pressostato)	DO (display 0 – mostra a etapa atual – unidades)	TL (armazena o tempo da lavagem que vem da memória)
SN (sensor de nível d'água)	D1 (display 1 – mostra a etapa atual – dezenas)	TE (armazena o tempo do enxágue que vem da memória)
DATA_FLASH	D2 (display 2 – auxilia na identificação da etapa atual)	TM (armazena o tempo do molho que vem da memória)
-	D3 (display 3 – auxilia na identificação da etapa atual)	TC (armazena o tempo da centrifugação que vem da memória)
-	D4 (display 4 – mostra o número total de etapas do processo – unidades)	NM (armazena o número de molhos que vem da memória)
-	D5 (display 5 – mostra o número total de etapas do processo – dezenas)	NE (armazena o número de enxágues que vem da memória)
-	D6 (display 6 - mostra a contagem regressiva para finalização de uma etapa - unidades)	NUM_ETAPAS (armazena o número de etapas do processo que vem da memória)
-	D7 (display 7 - mostra a contagem regressiva para finalização de uma etapa - dezenas)	ETAPA_ATUAL (vai armazenando a etapa atual conforme o processo avança)

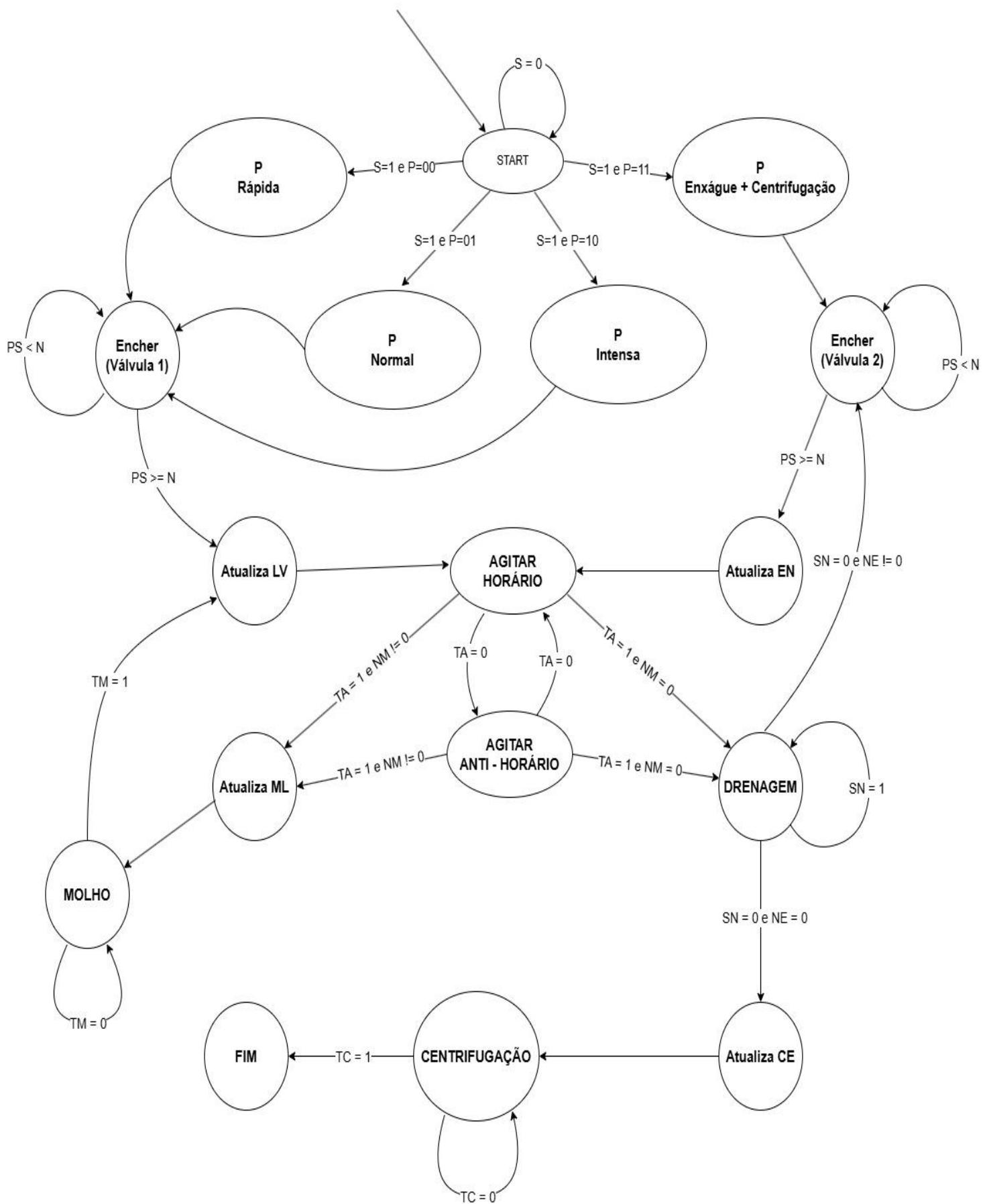


Figura 2 – Design da FSM de alto nível

As ações realizadas em cada um dos estados estão descritas a seguir:

ESTADOS	AÇÃO
<i>START</i>	Recebe informações da interface de lavagem
<i>P Rápida</i>	ADDR_FLASH = 0000 Carrega configurações da memória
<i>P Normal</i>	ADDR_FLASH = 0001 Carrega configurações da memória
<i>P Intensa</i>	ADDR_FLASH = 0010 Carrega configurações da memória
<i>P Enxágue + Centrifugação</i>	ADDR_FLASH = 0011 Carrega configurações da memória
<i>Encher (Válvula 1)</i>	VALV_ÁGUA = 01
<i>Atualiza LV</i>	Atualiza valor da etapa atual Reseta o timer e seta para TL
<i>Encher (Válvula 2)</i>	VALV_ÁGUA = 10
<i>Atualiza EN</i>	Atualiza valor da etapa atual Reseta o timer e seta para TE Atualiza NE (nº de enxágues restante) Dispara o timer (pra TL ou TE) Motor baixa rotação horária MOTOR = 0101
<i>AGITAR HORÁRIO</i>	Dispara o timer (pra TL ou TE) Motor baixa rotação anti-horária MOTOR = 0110
<i>AGITAR ANTI-HORÁRIO</i>	Atualiza valor da etapa atual Reseta o timer e seta pra TM Atualiza NM (nº de molhos restante)
<i>Atualiza ML</i>	Atualiza valor da etapa atual Reseta o timer e seta pra TM Atualiza NM (nº de molhos restante)
<i>MOLHO</i>	Desliga motor Dispara timer para valor de TM
<i>DRENAGEM</i>	Aciona bomba de drenagem – BD = 1
<i>Atualiza CE</i>	Seta motor para alta rotação no sentido horário Reseta o timer e seta para TC Atualiza valor da etapa atual MOTOR = 1001
<i>CENTRIFUGAÇÃO</i>	Aciona atuador do freio – AF = 1 Aciona bomba de drenagem – BD = 1 Dispara timer para valor de TC
<i>FIM</i>	Reseta tudo

2.2.2. DESCRIÇÃO DA FSM DE ALTO NÍVEL

A FSM é iniciada no estado “START”, no qual todas as saídas são resetadas, exceto os registradores responsáveis pela definição do programa de lavagem NIVEL (N) e PROG (P). Numa transição condicionada por S=1, a máquina passa para o estado definido pelo número do programa:

P	Estado seguinte (Programa de lavagem)	ADDR_FLASH
00	“P Rápida”	0000
01	“P Normal”	0001
10	“P Intensa”	0010
11	“P Enxágue + Centrifugação”	0011

Em qualquer que seja o estado assumido de P, a saída ADDR_FLASH que representa o endereço dos dados armazenados na memória referente à configuração de lavagem, assume algum valor condicionado por P. É feito então a carga de todos os dados provenientes da *flash memory*, o que inclui o tempo de lavagem (TL), tempo do enxágue (TE), tempo do molho (TM), tempo da centrifugação (TC), quantidade de enxágues (NE) e quantidade de molhos (NM) que devem ocorrer e o número de etapas total do processo (NUM_ETAPAS).

Em seguida, tomemos P=00 como primeiro exemplo. A partir do estado “P Rápida”, a próxima borda de *clock* o levará ao estado “ENCHER (Valv 1)”. Isso vale para as demais possibilidades de P exceto quando P=11, pois a transição será para o estado “ENCHER (Valv 2)”.

Em “ENCHER (Valv 1)”, a entrada de água dessa válvula, ramal para a entrada de sabão no tanque, é acionada fazendo VALV_ÁGUA = 01. Já em “ENCHER (Valv 2)”, a entrada de água acionada para o ramal para a entrada de amaciante no tanque é feita com VALV_ÁGUA = 10. Assim ela permanece até que o pressostato indique que o tanque está cheio. Isso é feito pela comparação entre o dado retornado pelo pressostato (PS) e o nível de água selecionado no início da lavagem (N). Uma vez atingido, ocorre a transição para o estado “ATUALIZA LV” se estiver em “ENCHER (Valv 1)” ou para o estado “ATUALIZA EN” se estiver em “ENCHER (Valv 2)”.

Em “ATUALIZA LV”, o temporizador é resetado e passa a receber valor de TL, que armazena o tempo de lavagem, além de incrementar o número da etapa atual (ETAPA ATUAL += 1), dando início ao processo.

Em “ATUALIZA EN”, o temporizador é resetado e passa a receber valor de TE, que armazena o tempo de enxágue. Além de incrementar o número da etapa atual (ETAPA ATUAL += 1) e decrementar os enxágues restantes do processo (NE = NE -1).

Seja em “ATUALIZA LV” ou “ATUALIZA EN”, o processo segue para o estado de “AGITAR HORÁRIO” que alterna com o estado “AGITAR ANTI-HORÁRIO” em cada borda de *clock*. Nesse momento, o sistema dispara o temporizador e aciona o motor em baixa rotação no sentido horário (MOTOR = 0101) ou anti-horário (MOTOR = 0110). Enquanto a contagem não chegar em zero, essa intercalação permanece. Quando atinge zero a máquina verifica se há etapas de molhos restantes. Se sim, vá para o estado “ATUALIZA ML”, caso contrário vá para “DRENAGEM”.

No estado “ATUALIZA ML” o temporizador é resetado e passa a receber valor de TM, que armazena o tempo de molho, além de incrementar o número da etapa atual (ETAPA ATUAL += 1) e decrementar os molhos restantes (NM = NM -1). Em seguida, irá para “MOLHO”.

No estado “MOLHO”, o temporizador é disparado e o motor desligado. Quando a contagem terminar, volte para os estados de agitação, mais especificamente, vá para “AGITAR HORÁRIO”.

Quando a FSM vai para o estado “DRENAGEM”, isso significa que todo o processo de lavagem terminou, o que inclui o tempo de agitação e de molho. Aqui a bomba de drenagem é acionada (BD = 1) para retirar toda água do tanque. Permanece nessa condição enquanto o sensor d’água (SN) estiver em nível lógico ‘1’ (contém água). Quando não houver mais água, a máquina verifica se há etapas de enxágues restantes. Se sim, vá para “ENCHER (Valv 2)”, caso contrário vá para “ATUALIZA CE”.

Em “ATUALIZA CE”, o temporizador é resetado e passa a receber valor de TC, que armazena o tempo de centrifugação, além de incrementar o número da etapa atual (ETAPA ATUAL += 1). Em uma transição incondicional vá para “CENTRIFUGAÇÃO”.

No estado de “CENTRIFUGAÇÃO”, a controladora dispara o temporizador e aciona o motor em alta rotação sentido horário (MOTOR = 1001). Enquanto isso, a bomba de drenagem e o atuador do freio estão também ativos (BD = 1 e AF = 1). Após a contagem terminar vá para “FIM”.

No estado “FIM”, todos os registradores com informações guardadas são resetados para então, voltar para o estado “START” na próxima borda de *clock*.

2.2.3. DATAPATH

A figura 3 a seguir mostra o Caminho de Dados implementado. A funcionalidade de cada um dos blocos construtivos é demonstrada adiante.

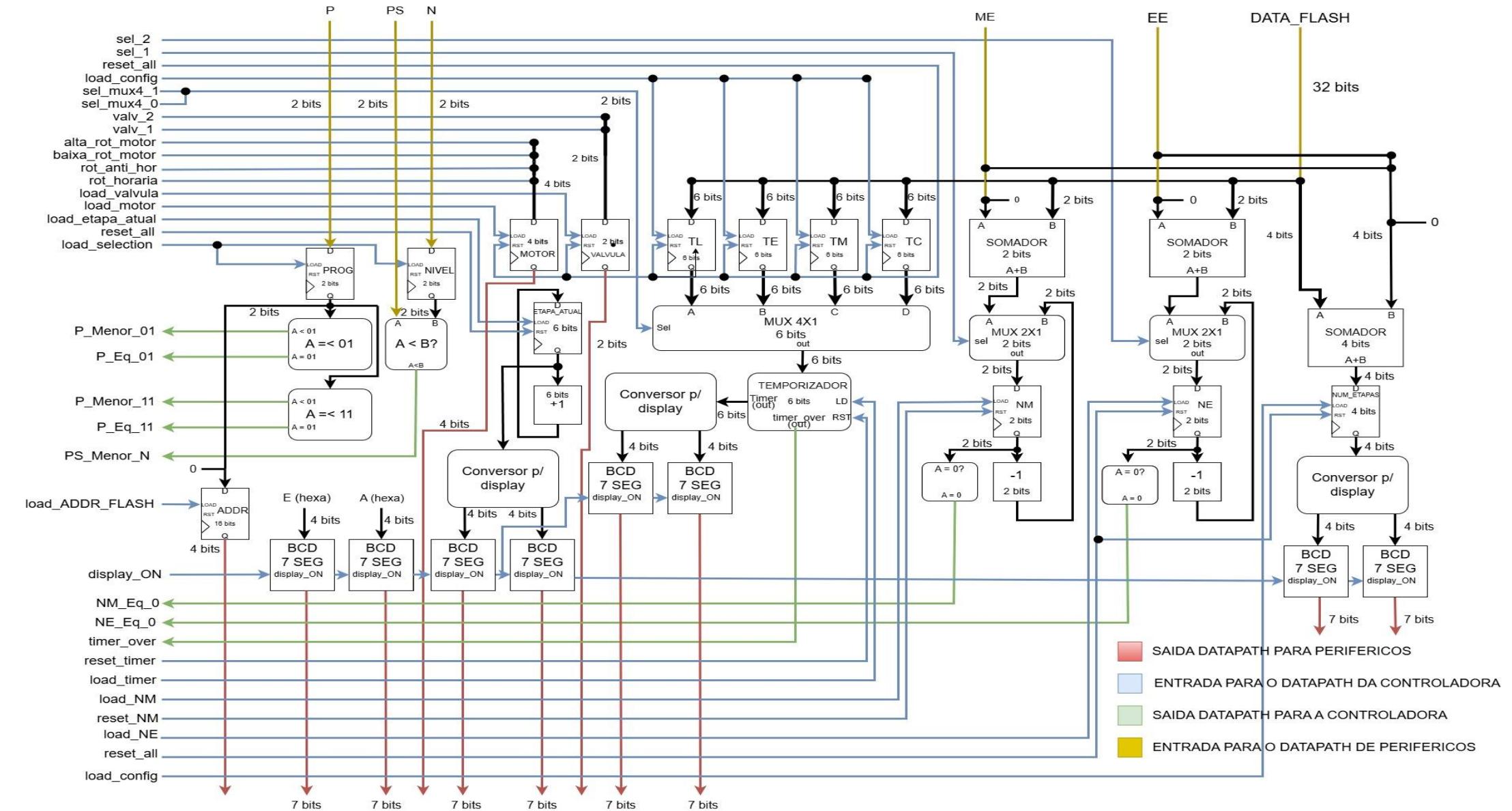


Figura 3 - Datapath

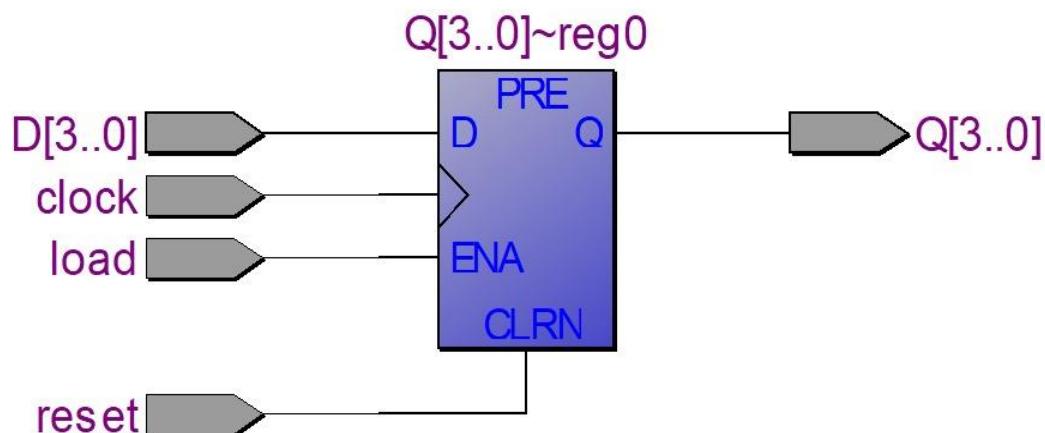
REGISTRADOR GENÉRICO

*Registrador multipropósito de tamanho genérico.
Implementado com base no registrador descrito na literatura de VAHID, 2008.*

CÓDIGO VHDL:

```
1  -- Projeto de registrador tamanho genérico
2  -- com entrada reset e load síncronas
3
4  LIBRARY IEEE;
5  use ieee.std_logic_1164.all;
6
7  Entity Reg_Generic is
8  |
9  generic(
10    DATA_WIDTH : natural := 4
11  );
12  port(
13    -- Inputs
14    clock, reset, load : in std_logic;
15    D                  : in std_logic_vector(DATA_WIDTH - 1 downto 0);
16    -- Outputs
17    Q : out std_logic_vector(DATA_WIDTH - 1 downto 0) := (others => '0')
18  );
19
20 end Reg_Generic;
21
22 Architecture RTL of Reg_Generic is
23
24 begin
25  -- Reseta em nível lógico '1'
26  -- Carga em nível lógico '1'
27  Q <= (others => '0') when reset = '1' else
28  D when rising_edge(clock) and load='1';
29
30 end RTL;
```

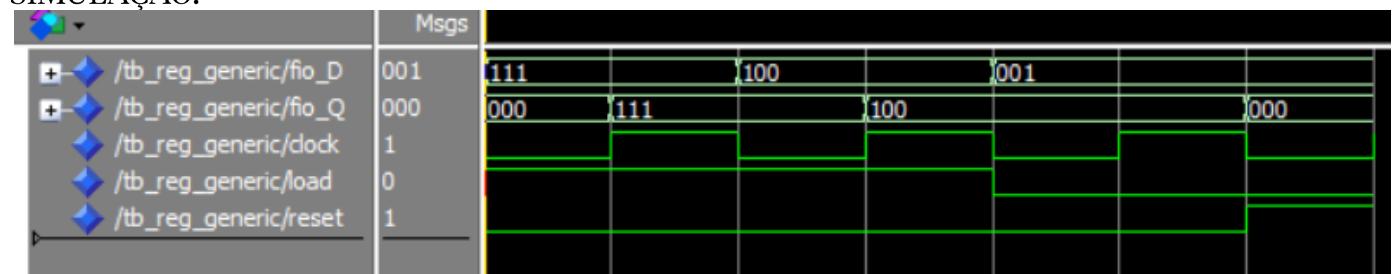
RTL VIEWER:



TESTBENCH:

```
1  LIBRARY IEEE;
2  use ieee.std_logic_1164.all;
3
4  entity TB_Reg_Generic is
5  end TB_Reg_Generic;
6
7  architecture RTL of TB_Reg_Generic is
8
9      constant DATA_WIDTH      : natural := 3;
10     signal fio_D, fio_Q      : std_logic_vector(DATA_WIDTH - 1 downto 0);
11     signal clock, load, reset : std_logic := '0';
12
13    component Reg_Generic is
14        generic(DATA_WIDTH : natural);
15        port(
16            clock, reset, load : in std_logic;
17            D                 : in std_logic_vector(DATA_WIDTH - 1 downto 0);
18            Q                 : out std_logic_vector(DATA_WIDTH - 1 downto 0) := (others => '0')
19        );
20    end component;
21
22 begin
23     Reg_1 : Reg_Generic
24         generic map (DATA_WIDTH => 3)
25         port map (
26             D      => fio_D,
27             Q      => fio_Q,
28             reset => reset,
29             load   => load,
30             clock  => clock
31         );
32
33     clock <= not clock after 5 ns;
34
35     process
36         begin
37             load <= '1'; reset <= '0'; fio_D <= "111";
38             wait for 10 ns;
39             fio_D <= "100";
40             wait for 10 ns;
41             load <= '0'; fio_D <= "001";
42             wait for 10 ns;
43             reset <= '1';
44             wait;
45     end process;
46 end RTL;
```

SIMULAÇÃO:



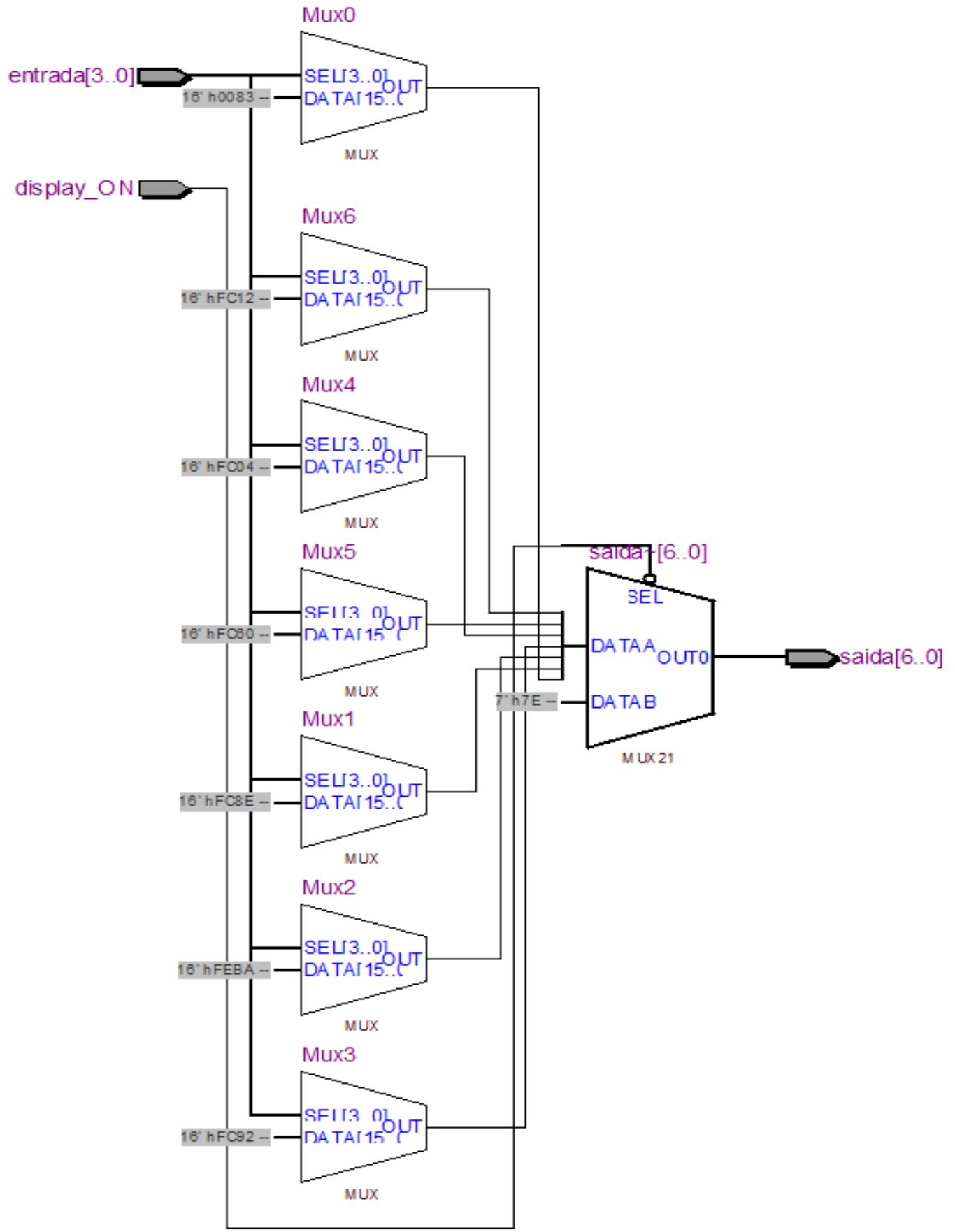
DISPLAY DE 7 SEGMENTOS

Converte um número hexadecimal em uma codificação para exibição em um display de 7 segmentos.
Possui uma entrada semelhante a um botão ON

CÓDIGO VHDL:

```
1  LIBRARY IEEE;
2  use ieee.std_logic_1164.all;
3
4  Entity Bcd_7seg is
5    port (
6      display_ON  : in std_logic;
7      entrada      : in std_logic_vector (3 downto 0);
8      saida        : out std_logic_vector (6 downto 0)
9    );
10 end Bcd_7seg;
11
12 architecture with_select_bcd7seg of Bcd_7seg is
13 begin
14   process(display_ON, entrada)
15   begin
16     if (display_ON = '0') then
17       saida <= "0111111";
18     else
19       case entrada is
20         when "0000" => saida <= "1000000";
21         when "0001" => saida <= "1111001";
22         when "0010" => saida <= "0100100";
23         when "0011" => saida <= "0110000";
24         when "0100" => saida <= "0011001";
25         when "0101" => saida <= "0010010";
26         when "0110" => saida <= "0000010";
27         when "0111" => saida <= "1111000";
28         when "1000" => saida <= "0000000";
29         when "1001" => saida <= "0010000";
30         when "1010" => saida <= "0001000";
31         when "1011" => saida <= "0000011";
32         when "1100" => saida <= "1000110";
33         when "1101" => saida <= "0100001";
34         when "1110" => saida <= "0000110";
35         when "1111" => saida <= "0001110";
36         when others => saida <= "0111111";
37       end case;
38     end if;
39   end process;
40 end with select bcd7seg;
```

RTL VIEWER:



TESTBENCH:

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  entity TB_Bcd_7seg is
5  end TB_Bcd_7seg;
6
7  architecture teste OF TB_Bcd_7seg is
8
9  component Bcd_7seg is
10 end component;
11
12 signal fio_display_ON : std_logic;
13 signal fio_entrada      : std_logic_vector(3 downto 0);
14 signal fio_saida        : std_logic_vector(6 downto 0);
15
16 begin
17     instance_BCD : Bcd_7seg
18         port map(
19             display_ON  => fio_display_ON,
20             entrada    => fio_entrada,
21             saida       => fio_saida
22         );
23     fio_display_ON <= '0', '1' after 20 ns;
24     fio_entrada <= "1111", "0000" after 20 ns,
25                           "0011" after 40 ns, "0101" after 60 ns,
26                           "0111" after 80 ns, "1001" after 100 ns,
27                           "1011" after 120 ns, "1100" after 140 ns;
28
29 end teste;
```

SIMULAÇÃO:

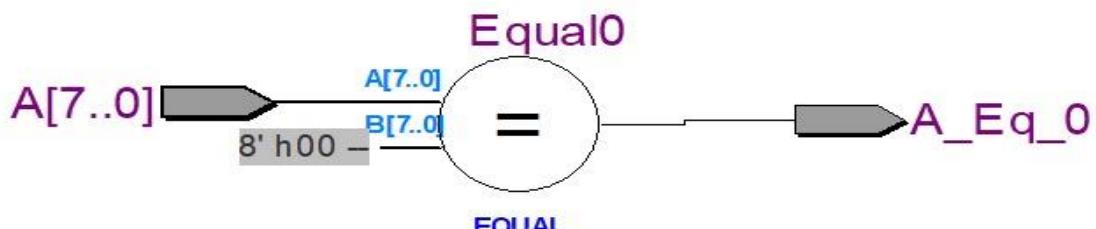
COMPARADOR IGUAL A ZERO

Comparador tamanho genérico que ativa nível lógico '1' quando a entrada tem valor igual a zero.

CÓDIGO VHDL:

```
1  -- Projeto de Comparador igual a zero
2  library ieee;
3  use ieee.std_logic_1164.all;
4
5  Entity Comparador_Eq_0 is
6
7      Generic (
8          DATA_WIDTH : natural := 8
9      );
10     Port (
11         -- Inputs
12         A      : in std_logic_vector(DATA_WIDTH-1 downto 0);
13         -- Outputs
14         A_Eq_0 : out std_logic
15     );
16 End Comparador_Eq_0;
17
18 Architecture RTL of Comparador_Eq_0 is
19
20     signal B : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
21
22 Begin
23
24     process(A)
25     begin
26         if(A = B) then
27             A_eq_0 <= '1';
28         else
29             A_eq_0 <= '0';
30         end if;
31     end process;
32
33 End RTL;
```

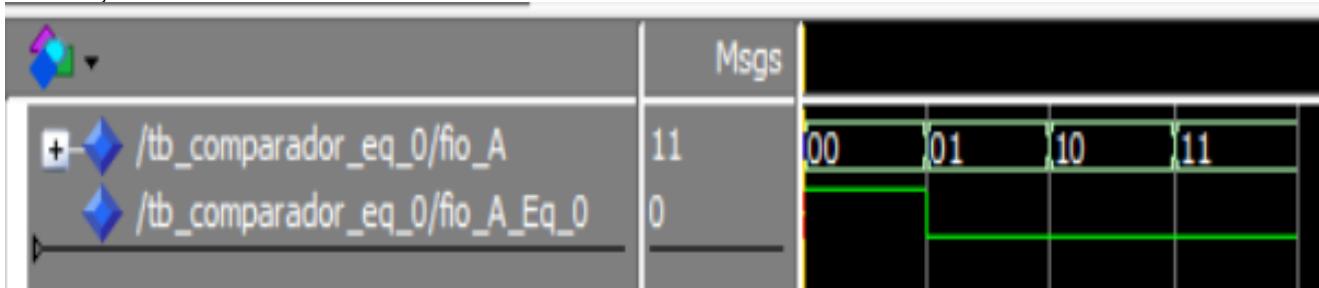
RTL VIEWER:



TESTBENCH:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity TB_Comparador_Eq_0 is
5  end TB_Comparador_Eq_0;
6
7  architecture RTL of TB_Comparador_Eq_0 is
8
9  component Comparador_Eq_0 is
10   Generic (DATA_WIDTH : natural := 8);
11   Port(
12     A      : in std_logic_vector(DATA_WIDTH-1 downto 0);
13     A_Eq_0 : out std_logic
14   );
15  end component;
16
17  signal fio_A : std_logic_vector(1 downto 0);
18  signal fio_A_Eq_0 : std_logic;
19
20 begin
21   Comparador_1 : Comparador_Eq_0
22     Generic map (DATA_WIDTH => 2)
23     Port map(
24       A      => fio_A,
25       A_Eq_0 => fio_A_Eq_0
26     );
27   process
28     begin
29       fio_A <= "00";
30       wait for 10 ns;
31       fio_A <= "01";
32       wait for 10 ns;
33       fio_A <= "10";
34       wait for 10 ns;
35       fio_A <= "11";
36       wait;
37     end process;
38
39 end RTL;
```

SIMULAÇÃO:



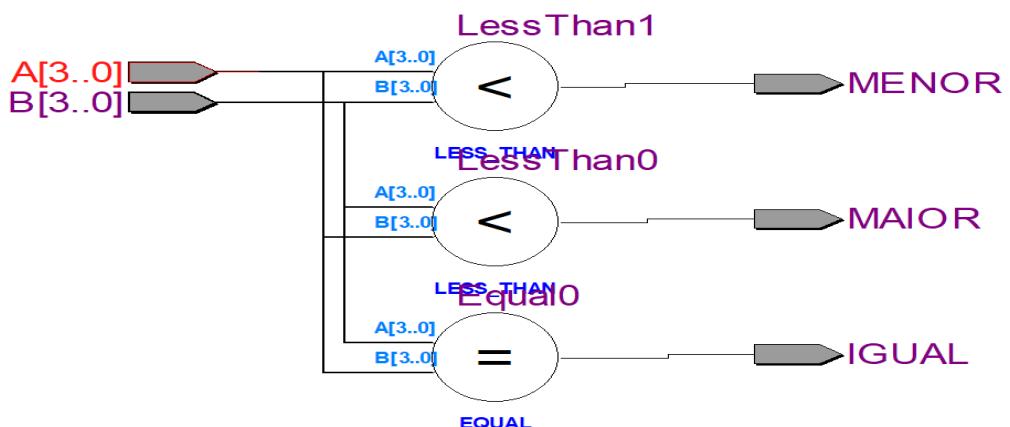
COMPARADOR DE MAGNITUDE

Comparador de magnitude tamanho genérico. Possui 3 saídas lógicas que ativa em nível lógico '1' se: $A > B$, ou $A < B$ ou $A = B$

CÓDIGO VHDL:

```
1  -- Comparador de magnitude tamanho genérico
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.all;
4  use ieee.numeric_std.all;
5
6  entity Comparador is
7    Generic(
8      DATA_WIDTH : natural := 4
9    );
10   Port (
11     -- Inputs
12     A, B : in std_logic_vector(DATA_WIDTH - 1 downto 0);
13
14     --Outputs
15     MENOR, IGUAL, MAIOR : out std_logic
16   );
17 end Comparador;
18
19 architecture RTL of comparador is
20
21 begin
22
23   MAIOR <= '1' when (A>B) else '0';
24   MENOR <= '1' when (A<B) else '0';
25   IGUAL <= '1' when (A=B) else '0';
26
27 end RTL;
```

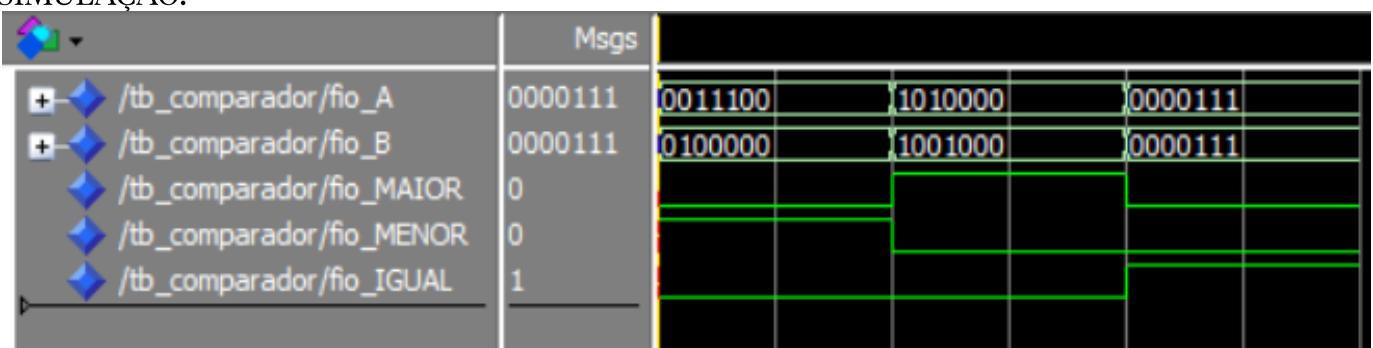
RTL VIEWER:



TESTBENCH:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use ieee.numeric_std.all;
4
5  entity TB_Comparador is
6  end TB_Comparador;
7
8  architecture RTL of TB_Comparador is
9
10   constant DATA_WIDTH : natural := 7;
11   signal fio_A, fio_B : std_logic_vector(DATA_WIDTH-1 downto 0);
12   signal fio_MAIOR, fio_MENOR, fio_IGUAL : std_logic;
13
14  component Comparador
15    Generic(
16      DATA_WIDTH : natural
17    );
18    Port (
19      A, B : in std_logic_vector(DATA_WIDTH - 1 downto 0);
20      MAIOR, MENOR, IGUAL : out std_logic
21    );
22  end component;
23
24 begin
25   Comparador_1 : Comparador
26     generic map (DATA_WIDTH => 7)
27     port map (
28       A      => fio_A,
29       B      => fio_B,
30       MAIOR => fio_MAIOR,
31       MENOR => fio_MENOR,
32       IGUAL => fio_IGUAL
33     );
34   process
35     begin
36       fio_A <= "0011100";
37       fio_B <= "0100000";
38       wait for 10 ns;
39       fio_A <= "1010000";
40       fio_B <= "1001000";
41       wait for 10 ns;
42       fio_A <= "0000111";
43       fio_B <= "0000111";
44       wait;
45   end process;
46 end RTL;
```

SIMULAÇÃO:



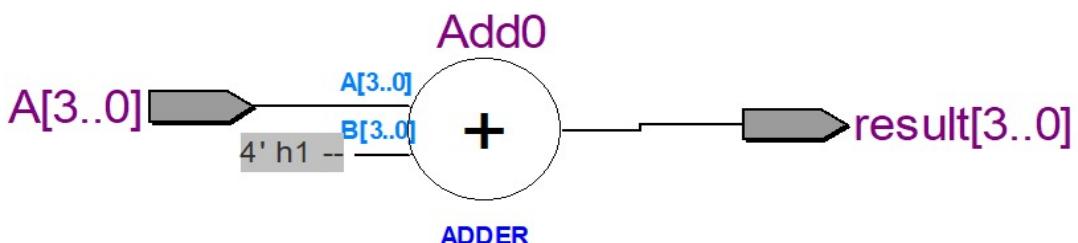
INCREMENTADOR

Incrementador multipropósito utilizado para iterar sobre a contagem de determinados ciclos da máquina. Implementado com base no incrementador descrito na literatura de VAHID, 2008.

CÓDIGO VHDL:

```
1  LIBRARY IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.numeric_std.all;
4
5  Entity Incrementador is
6    generic(
7      DATA_WIDTH : natural := 4
8    );
9    port(
10      A          : in    std_logic_vector(DATA_WIDTH-1 downto 0);
11      result     : out   std_logic_vector(DATA_WIDTH-1 downto 0)
12    );
13  end Incrementador;
14
15 Architecture RTL of Incrementador is
16   signal convert : unsigned(DATA_WIDTH-1 downto 0);
17 begin
18   convert <= unsigned(A);
19   process(convert) is
20   begin
21     result <= std_logic_vector(convert + 1);
22   end process;
23 end RTL;
```

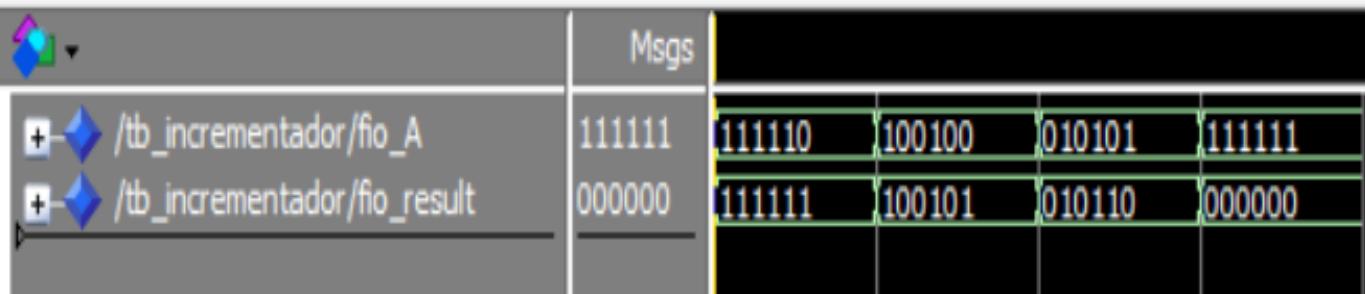
RTL VIEWER:



TESTBENCH:

```
1  LIBRARY IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.numeric_std.all;
4
5  entity TB_Incrementador is
6  end TB_Incrementador;
7
8  architecture RTL of TB_Incrementador is
9
10 component Incrementador is
11 generic(
12     DATA_WIDTH : natural := 4
13 );
14 port(
15     A      : in    std_logic_vector(DATA_WIDTH-1 downto 0);
16     result : out   std_logic_vector(DATA_WIDTH-1 downto 0)
17 );
18 end component;
19
20 signal fio_A, fio_result : std_logic_vector(5 downto 0);
21
22 begin
23     instance_incrementador : incrementador
24     generic map(DATA_WIDTH => 6)
25     port map(
26         A => fio_A,
27         result=>fio_result
28     );
29     fio_A <= "111110", "100100" after 10 ns,
30     "010101" after 20 ns, "111111" after 30 ns;
31
32 end RTL;
```

SIMULAÇÃO:



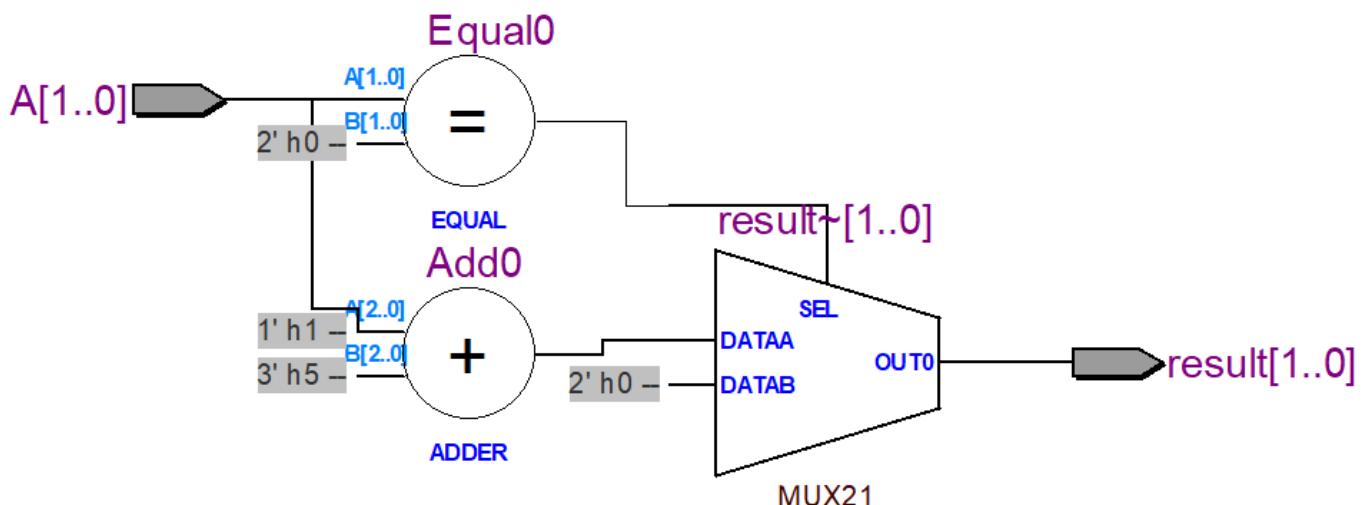
DECREMENTADOR

Decrementador multipropósito de 2 bits utilizado para iterar sobre a contagem de determinados ciclos da máquina. Implementado com base no incrementador descrito na literatura de VAHID, 2008.

CÓDIGO VHDL:

```
1  LIBRARY IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.numeric_std.all;
4  entity decrementador is
5    port(
6      A          : in    std_logic_vector(1 downto 0);
7      result     : out   std_logic_vector(1 downto 0)
8    );
9  end decrementador;
10
11 architecture decrementar of decrementador is
12 begin
13 process(A) is
14 begin
15 if A = "00" then
16   result <= "00";
17 else
18   result <= std_logic_vector(unsigned(A)-"01");
19 end if;
20 end process;
21
22 end decrementar;
```

RTL VIEWER:



TESTBENCH:

```
1  LIBRARY IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.numeric_std.all;
4
5  Entity tb_decrementador is
6  End tb_decrementador;
7
8  Architecture teste of tb_decrementador is
9  Component decrementador is
10 End Component;
11 Port(
12   A      : in    STD_LOGIC_VECTOR(1 DOWNTO 0);
13   result : out   STD_LOGIC_VECTOR(1 DOWNTO 0)
14 );
15 End Component;
16 Signal fio_A, fio_result : STD_LOGIC_VECTOR(1 DOWNTO 0);
17
18 Begin
19   Instance_decrementador: decrementador
20   Port Map(A => fio_A, result=>fio_result);
21
22   fio_A <= "11", "10" after 10 ns, "01" after 20 ns, "00" after 30 ns;
23
24 End teste;
```

SIMULAÇÃO:

		Msgs				
			11	10	01	00
	+ /tb_decrementador/fio_A	00				
	+ /tb_decrementador/fio_result	00	10	01	00	

SOMADOR

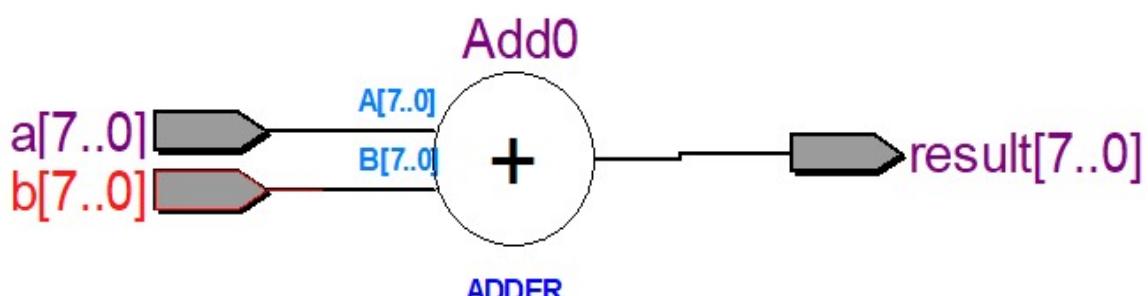
Implementa uma função de soma sem sinal.

Implementado com base no somador multipropósito descrito na literatura de VAHID, 2008.

CÓDIGO VHDL:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity somador is
6
7    Generic (
8      DATA_WIDTH : natural := 8
9    );
10   Port
11   (
12     a, b      : in std_logic_vector ((DATA_WIDTH-1) downto 0);
13     result    : out std_logic_vector ((DATA_WIDTH-1) downto 0)
14   );
15
16 end entity;
17
18 architecture soma of somador is
19 begin
20
21   result <= std_logic_vector(unsigned(a) + unsigned(b));
22
23 end soma;
24
```

RTL VIEWER:



TESTBENCH:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity tb_somador is
5  end tb_somador;
6
7  architecture teste of tb_somador is
8
9  component somador is
10 generic
11 (
12     DATA_WIDTH : natural := 8
13 );
14 port
15 (
16     a : in std_logic_vector ((DATA_WIDTH-1) downto 0);
17     b : in std_logic_vector ((DATA_WIDTH-1) downto 0);
18     result : out std_logic_vector ((DATA_WIDTH-1) downto 0)
19 );
20 end component;
21
22 signal fio_a, fio_b, fio_soma: std_logic_vector(3 downto 0);
23
24 begin
25
26 instancia_somador: somador
27 generic map (DATA_WIDTH => 4)
28 port map (a=>fio_a,b=>fio_b,result=>fio_soma);
29
30 -- As próximas linhas criam os estímulos da simulação,
31 -- A letra 'x' indica que os valores a seguir estão expressos em base hexadecimal
32 fio_a <= x"0", x"A" after 20 ns, x"2" after 40 ns, x"4" after 60 ns;
33 fio_b <= x"0", x"4" after 10 ns, x"3" after 30 ns, x"1" after 50 ns;
34
35 end teste;
```

SIMULAÇÃO:

	Msgs					
+◆ /tb_somador/fio_a	0100	0000	1010	0010		
+◆ /tb_somador/fio_b	0001	0000	0100	0011	0001	
+◆ /tb_somador/fio_soma	0101	0000	0100	1110	1101	0101

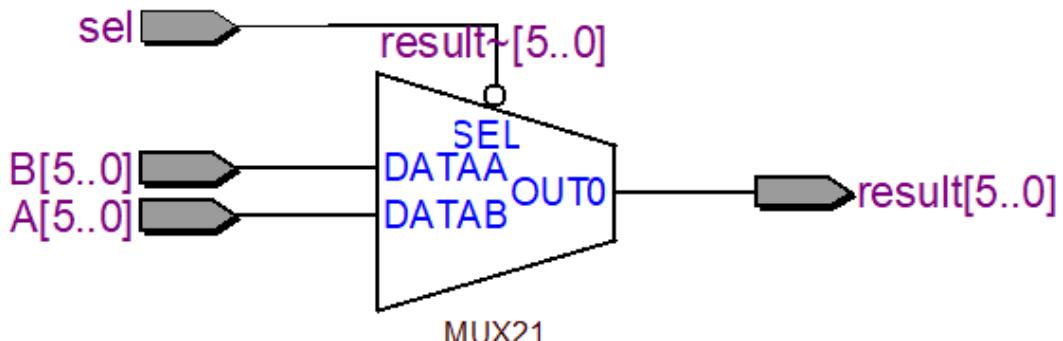
MULTIPLEXADOR 2X1

MUX 2x1 com entradas de tamanho genérico. Implementado com base no multiplexador multipropósito descrito na literatura de VAHID, 2008.

CÓDIGO VHDL:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux_2 is
5    generic (
6      Size : natural := 6
7    );
8    port(
9      A, B      : in std_logic_vector(Size-1 downto 0);
10     sel       : in std_logic;
11     result    : out std_logic_vector(Size-1 downto 0)
12   );
13 end entity;
14
15 architecture behaviour of mux_2 is
16
17 begin
18
19   with sel select
20     result <= A when '0',
21                   B when others;
22
23 end behaviour;
```

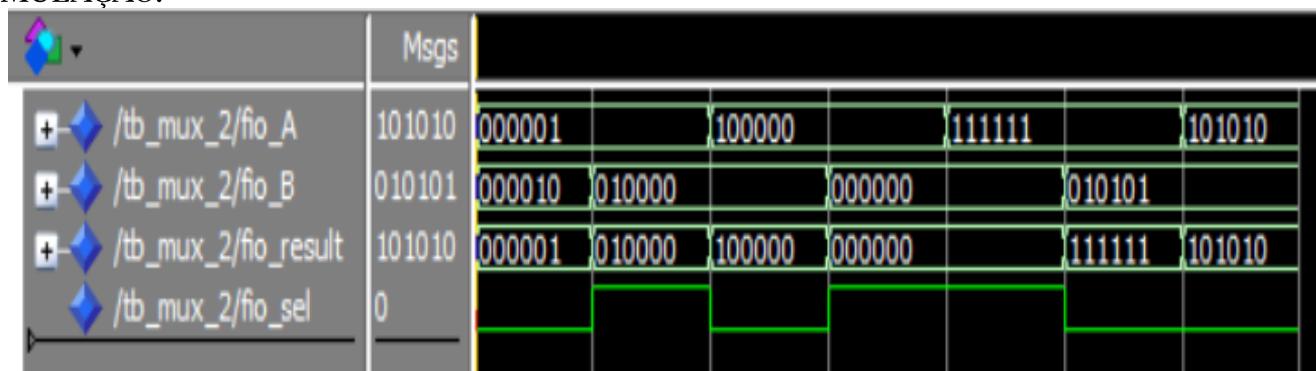
RTL VIEWER:



TESTBENCH:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity tb_mux_2 is
5  end entity;
6
7  architecture teste of tb_mux_2 is
8  component mux_2 is
9  generic (
10      Size : natural := 6
11  );
12  port(
13      A, B          : in std_logic_vector(Size-1 downto 0);
14      sel          : in std_logic;
15      result       : out std_logic_vector(Size-1 downto 0)
16  );
17  end component;
18  constant Size      : natural := 6;
19  signal fio_A, fio_B : std_logic_vector(Size-1 downto 0);
20  signal fio_result : std_logic_vector(Size-1 downto 0);
21  signal fio_sel    : std_logic;
22
23  begin
24      instance_mux: mux_2 generic map(Size => Size)
25      port map(
26          A          => fio_A,
27          B          => fio_B,
28          result     => fio_result,
29          sel        => fio_sel
30      );
31      fio_A <= "000001", "100000" after 20 ns,
32          "111111" after 40 ns, "101010" after 60 ns;
33      fio_B <= "000010", "010000" after 10 ns,
34          "000000" after 30 ns, "010101" after 50 ns;
35      fio_sel <= '0', '1' after 10 ns, '0' after 20 ns,
36          '1' after 30 ns, '1' after 40 ns, '0' after 50 ns;
37  end teste;
```

SIMULAÇÃO:



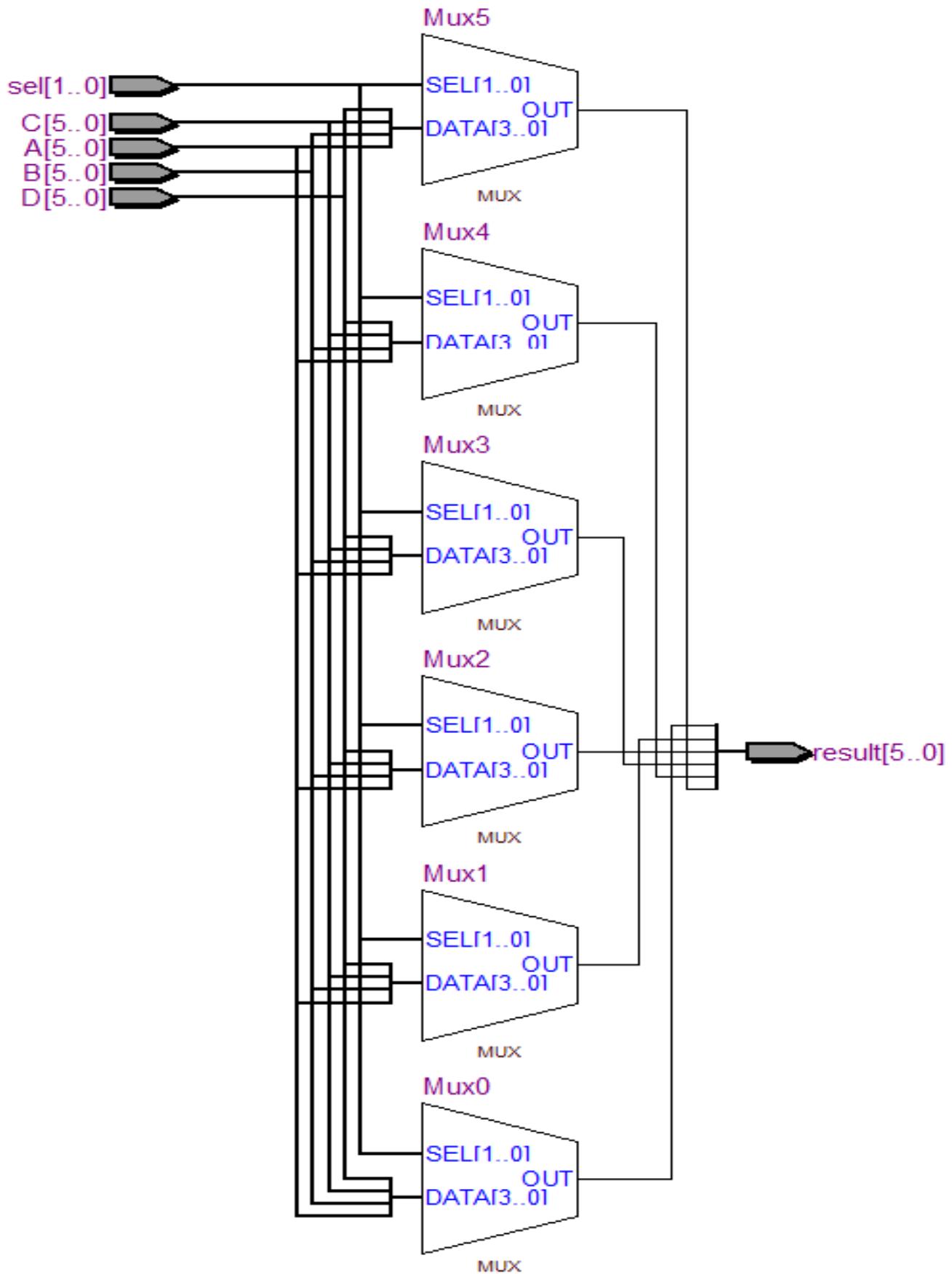
MULTIPLEXADOR 4X1

MUX 4x1 com entradas de tamanho genérico. Implementado com base no multiplexador multipropósito descrito na literatura de VAHID, 2008.

CÓDIGO VHDL:

```
1 -- Projeto de um MUX 4X1 entrada tamanho genérico
2 library ieee;
3 use ieee.std_logic_1164.all;
4
5 entity mux_4 is
6   generic (
7     Size : natural := 6
8   );
9   port(
10    A, B, C, D : in std_logic_vector(Size-1 downto 0);
11    sel        : in std_logic_vector(1 downto 0);
12    result      : out std_logic_vector(Size-1 downto 0)
13  );
14 end entity;
15
16 architecture behaviour of mux_4 is
17
18 begin
19
20  with sel select
21    result <= A when "00",
22                  B when "01",
23                  C when "10",
24                  D when others;
25 end behaviour;
```

RTL VIEWER:



TESTBENCH:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity tb_mux_4 is
5 end entity;
6
7 architecture teste of tb_mux_4 is
8 component mux_4 is
9 generic (
10     Size : natural := 6
11 );
12 port(
13     A, B, C, D : in std_logic_vector(Size-1 downto 0);
14     sel         : in std_logic_vector(1 downto 0);
15     result      : out std_logic_vector(Size-1 downto 0)
16 );
17 end component;
18 constant Size : natural := 6;
19 signal fio_A, fio_B, fio_C, fio_D : std_logic_vector(Size-1 downto 0);
20 signal fio_result              : std_logic_vector(Size-1 downto 0);
21 signal fio_sel                 : std_logic_vector(1 downto 0);
22
23 begin
24     instance_mux: mux_4 generic map(Size => Size)
25     port map(
26         A => fio_A,
27         B => fio_B,
28         C => fio_C,
29         D => fio_D,
30         result => fio_result,
31         sel => fio_sel
32     );
33     fio_A <= "000001", "100000" after 40 ns;
34     fio_B <= "000010", "010000" after 40 ns;
35     fio_c <= "000100", "111111" after 40 ns;
36     fio_D <= "001000", "000000" after 40 ns;
37     fio_sel <= "00", "01" after 10 ns, "10" after 20 ns, "11" after 30 ns,
38     "00" after 40 ns, "10" after 50 ns, "11" after 60 ns;
39
40 end teste;
```

SIMULAÇÃO:

	Msgs					
+◆ /tb_mux_4/fio_A	100000	000001			100000	
+◆ /tb_mux_4/fio_B	010000	000010			010000	
+◆ /tb_mux_4/fio_C	111111	000100			111111	
+◆ /tb_mux_4/fio_D	000000	001000			000000	
+◆ /tb_mux_4/fio_result	000000	000001	000010	000100	100000	111111
+◆ /tb_mux_4/fio_sel	11	00	01	10	11	00

DIVISOR DE CLOCK DE 1Hz

Gera um sinal de frequência 1Hz a partir de um clock de 50MHz do FPGA.

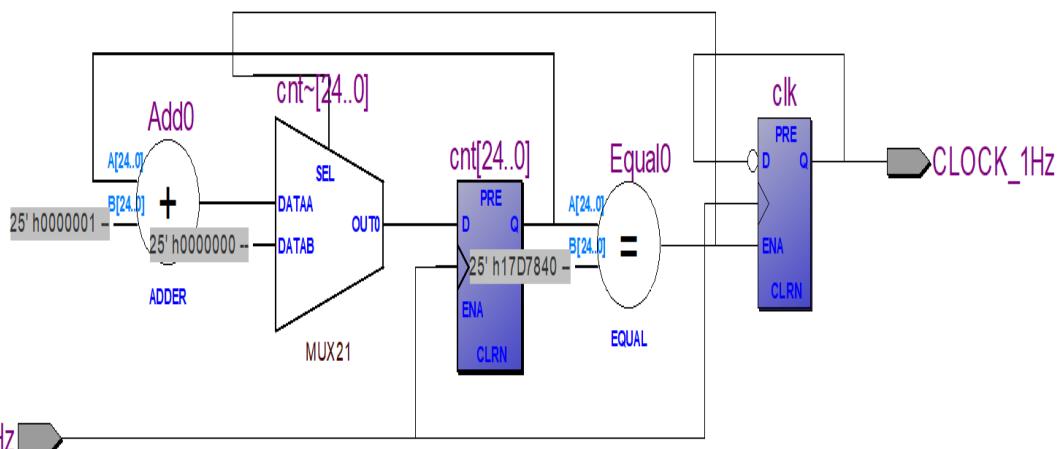
CÓDIGO VHDL:

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity div_clock_1Hz is
6      port
7      (
8          CLOCK_50MHz : in std_logic;
9          CLOCK_1Hz   : out std_logic := '0'
10     );
11
12 end entity;
13
14 architecture behaviour of div_clock_1Hz is
15
16     constant Divisor : integer := 25000000;
17
18 begin
19
20     process (CLOCK_50MHz)
21         variable cnt : integer range 0 to Divisor;
22         variable clk : std_logic := '0';
23     begin
24         if (rising_edge(CLOCK_50MHz)) then
25             if (cnt = Divisor) then
26                 clk := not clk;
27                 cnt := 0;
28             else
29                 cnt := cnt + 1;
30             end if;
31         end if;
32         CLOCK_1Hz <= clk;
33     end process;
34
35 end behaviour;

```

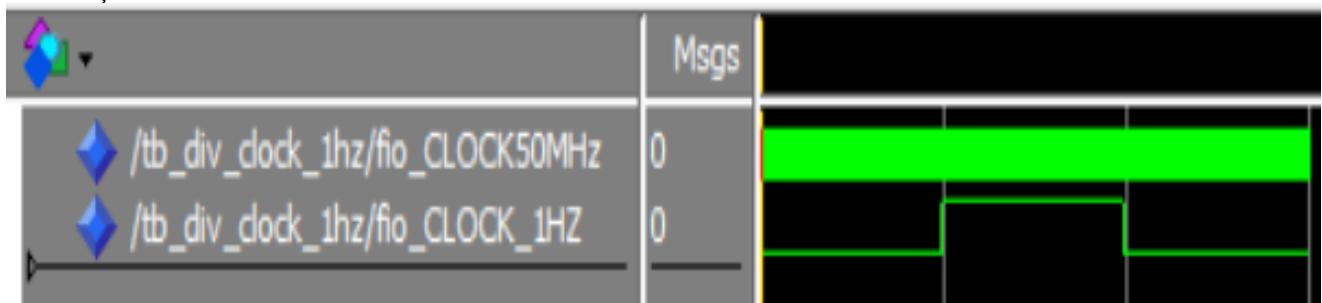
RTL VIEWER:



TESTBENCH:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity tb_div_clock_1Hz is
6  end entity;
7
8  architecture teste of tb_div_clock_1Hz is
9
10 component div_clock_1Hz is
11   port
12   (
13     CLOCK_50MHz : in std_logic;
14     CLOCK_1Hz   : out std_logic
15   );
16 end component;
17
18 signal fio_CLOCK50MHz, fio_CLOCK_1Hz : std_logic := '0';
19
20 begin
21
22   instance_divisor : div_clock_1Hz
23   port map(
24     CLOCK_50MHz => fio_CLOCK50MHz,
25     CLOCK_1Hz  => fio_CLOCK_1Hz
26   );
27   fio_CLOCK50MHz <= not fio_CLOCK50MHz after 10 ns;
28
29 end teste;
```

SIMULAÇÃO:



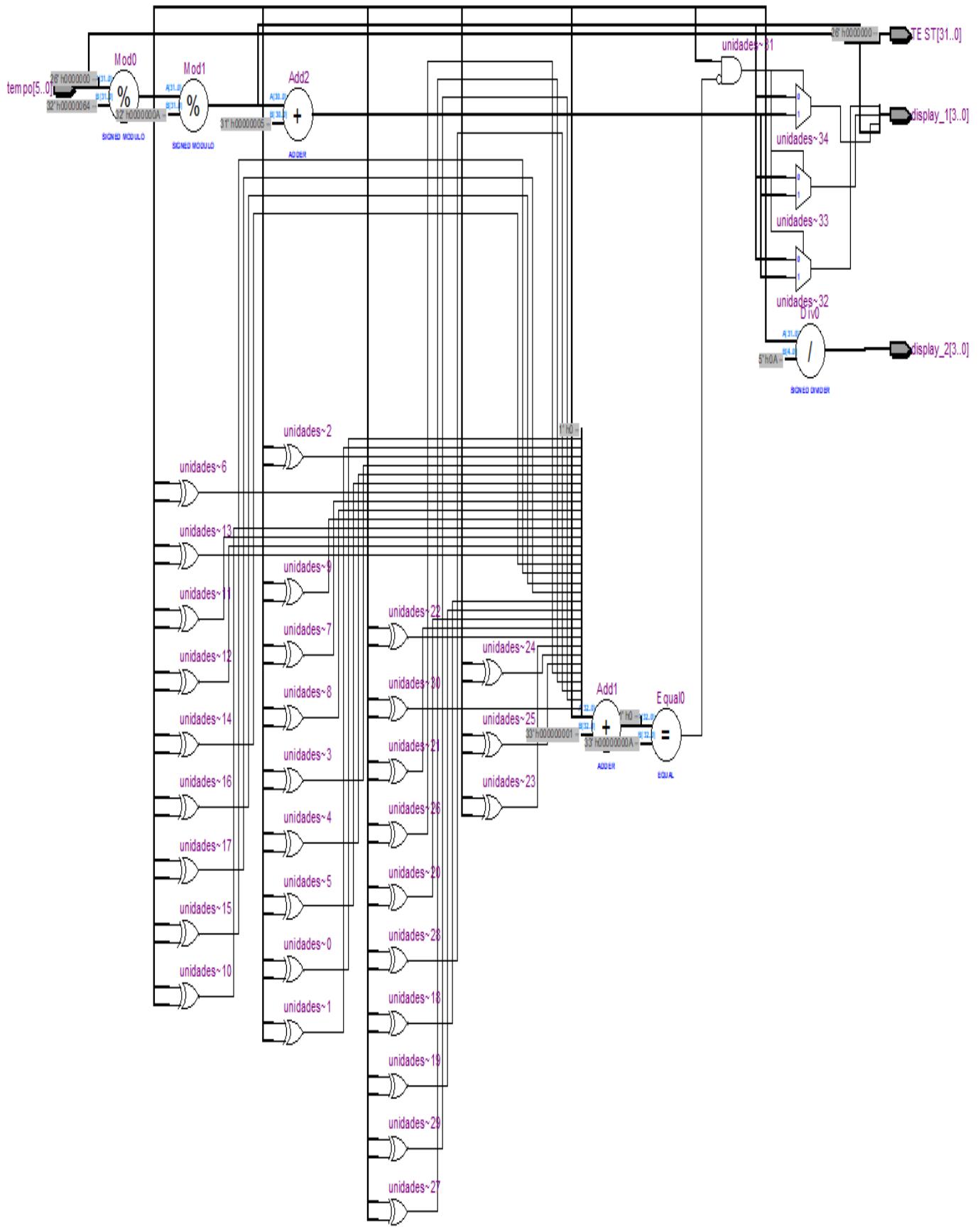
CONVERSOR PARA O DISPLAY

Implementa uma função de conversão binária para ser aplicada no display de 7 segmentos. Recebe um número binário de 6 bits e o converte separadamente em unidades e dezenas, enviando esses números para o display 1 e display 2 respectivamente.

CÓDIGO VHDL:

```
1  -- Projeto para um conversor numero binario 6 bits
2  -- para multiplos displays (2 displays no total)
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6  use ieee.std_logic_unsigned.all;
7
8  entity Conversor_Display is
9    Port (
10      tempo : in std_logic_vector(5 downto 0);
11      -- Saída para os displays de 7 segmentos
12      display_1 : out std_logic_vector(3 downto 0); --unidades
13      display_2 : out std_logic_vector(3 downto 0)  --dezenas
14    );
15 end Conversor_Display;
16
17 architecture RTL of Conversor_Display is
18
19  constant Maior      : integer :=100;
20  constant Menor      : integer :=10;
21  signal aux          : integer :=0;
22  signal total         : integer :=0;
23  signal dezenas       : integer :=0;
24  signal unidades       : integer :=0;
25
26 begin
27
28  total <= to_integer(unsigned(tempo));
29  display_1 <= std_logic_vector(to_unsigned(unidades, 4));
30  display_2 <= std_logic_vector(to_unsigned(dezenas, 4));
31  aux      <= total mod Maior;
32  dezenas   <= aux/Menor;
33  unidades <= aux mod Menor;
34
35 end RTL;
```

RTL VIEWER:



TESTBENCH:

```
1 library IEEE;
2 use ieee.numeric_std.all;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity TB_Conversor_Display is
6 end TB_Conversor_Display;
7
8 architecture RTL of TB_Conversor_Display is
9
10 component Conversor_Display is
11 Port (
12     tempo      : in std_logic_vector(5 downto 0);
13     display_1 : out std_logic_vector(3 downto 0);
14     display_2 : out std_logic_vector(3 downto 0)
15 );
16 end component;
17
18 signal fio_tempo : std_logic_vector(5 downto 0);
19 signal fio_unidades : std_logic_vector(3 downto 0);
20 signal fio_dezenas : std_logic_vector(3 downto 0);
21
22 begin
23
24     instance_Conversor : Conversor_Display
25     Port map(
26         tempo      => fio_tempo,
27         display_1 => fio_unidades,
28         display_2 => fio_dezenas
29     );
30     fio_tempo <= "UUUUUU", "110111" after 10 ns, "111111" after 20 ns,
31     "011110" after 30 ns, "001110" after 40 ns, "011011" after 50 ns;
32
33 end RTL;
```

SIMULAÇÃO:

	Msgs						
+/tb_conversor_display/fio_tempo	27	X	55	63	30	14	27
+/tb_conversor_display/fio_unidades	0111	0000	0101	0011	0000	0100	0111
+/tb_conversor_display/fio_dezenas	0010	0000	0101	0110	0011	0001	0010

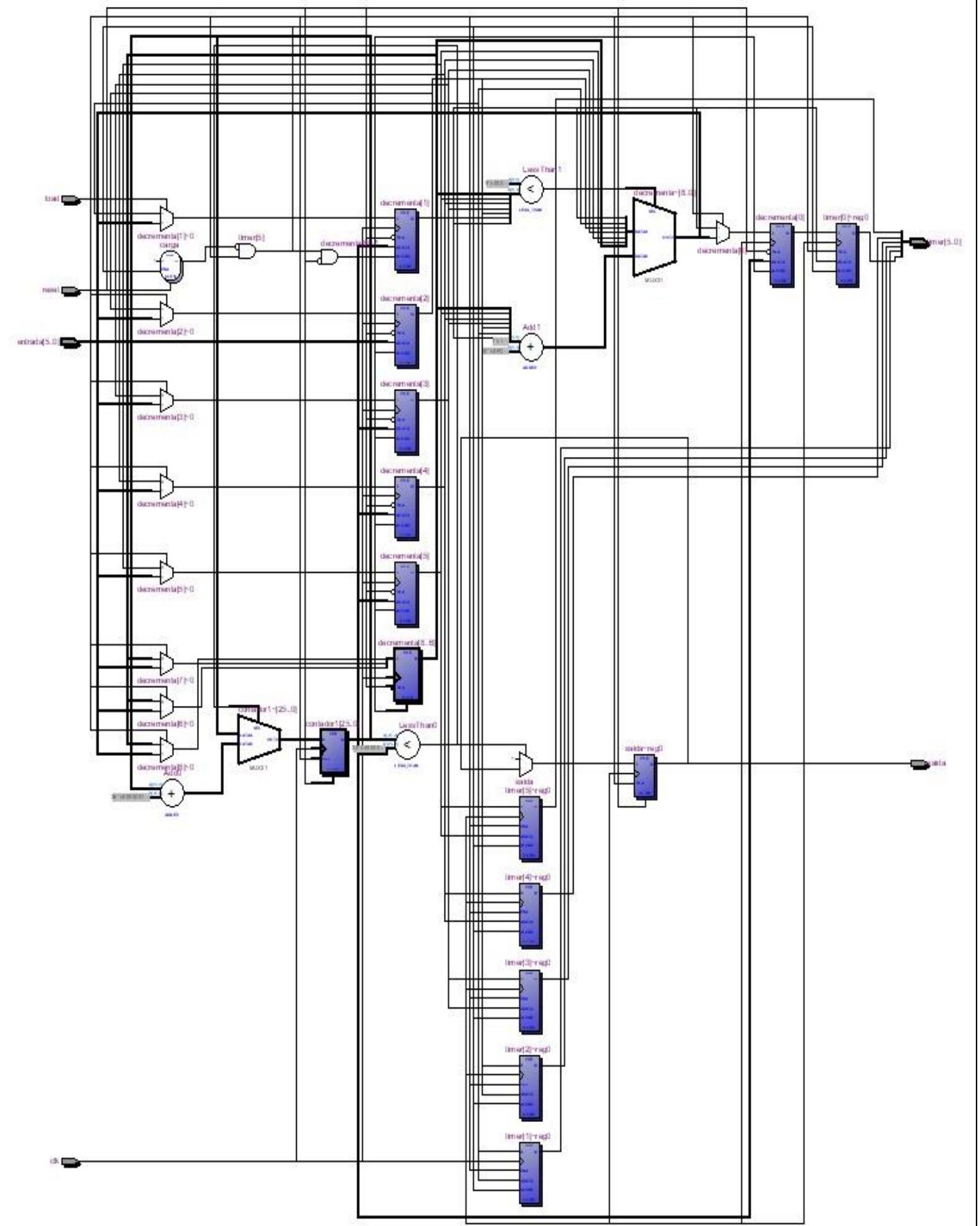
TEMPORIZADOR

Implementa um temporizador com contagem regressiva. O temporizador é composto por duas unidades principais: a unidade de controle e a unidade de contagem.
O temporizador tem quatro entradas e duas saídas.

CÓDIGO VHDL:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  entity Temporizador is
5    Port (
6      clk, reset, load      : in std_logic;
7      entrada                 : in std_logic_vector(5 downto 0);
8      saida                  : out std_logic := '0';
9      timer                  : out std_logic_vector(5 downto 0)
10     );
11 end entity Temporizador;
12 architecture RTL of Temporizador is
13   signal contadorl        : integer range 0 to 50000000 := 0;
14   signal decrementa       : integer range 0 to 500 := 0;
15   signal carga            : std_logic := '0';
16 begin
17   process(clk, reset, load)
18   begin
19     if reset = '1' then
20       contadorl <= 0;
21       saida <= '0';
22     elsif rising_edge(clk) and load = '1' then
23       if contadorl < (to_integer(unsigned(entrada))) then
24         contadorl <= contadorl + 1;
25       else
26         saida <= '1';
27       end if;
28     end if;
29   end process;
30   process(clk, reset, load)
31   begin
32     if reset = '1' then
33       carga <= '0';
34       timer <= (others => '0');
35     elsif (load = '1') then
36       if (carga = '0') then
37         decrementa <= to_integer(unsigned(entrada));
38         timer <= std_logic_vector(to_unsigned(decrementa, timer'length));
39         carga <= '1';
40       else
41         if (rising_edge(clk)) then
42           if(decrementa > 0) then
43             decrementa <= decrementa - 1;
44           end if;
45           timer <= std_logic_vector(to_unsigned(decrementa, timer'length));
46         end if;
47       end if;
48     end if;
49   end process;
50 end RTL;
```

RTL VIEWER:



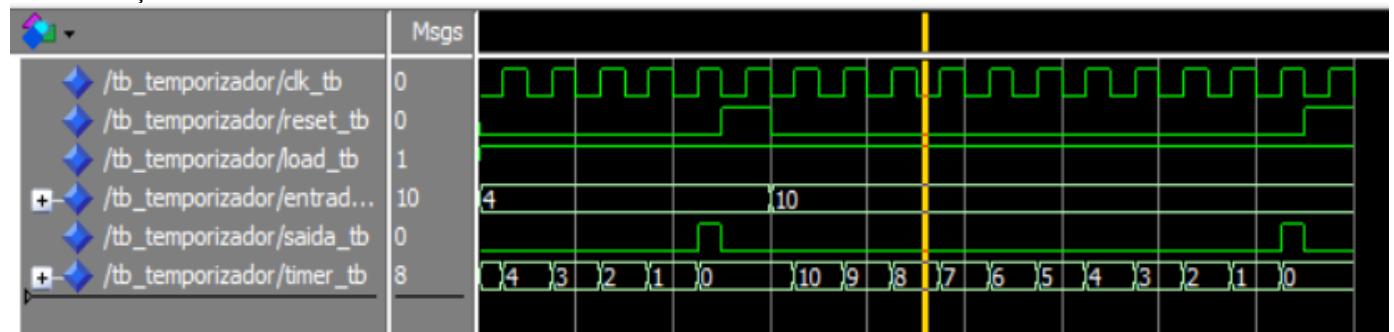
TESTBENCH:

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity TB_Temporizador is
5  end entity TB_Temporizador;
6
7  architecture RTL of TB_Temporizador is
8  component Temporizador is
9  port (
10    clk      : in std_logic;
11    reset    : in std_logic;
12    load     : in std_logic;
13    entrada  : in std_logic_vector(5 downto 0);
14    saida    : out std_logic;
15    timer    : out std_logic_vector(5 downto 0)
16  );
17 end component;
18 signal clk_tb      : std_logic := '0';
19 signal reset_tb    : std_logic;
20 signal load_tb    : std_logic;
21 signal entrada_tb : std_logic_vector(5 downto 0);
22 signal saida_tb   : std_logic;
23 signal timer_tb   : std_logic_vector(5 downto 0);
24 begin
25   My_Timer : Temporizador
26   port map (
27     clk      => clk_tb,
28     reset    => reset_tb,
29     load     => load_tb,
30     entrada  => entrada_tb,
31     saida    => saida_tb,
32     timer    => timer_tb
33   );
34   clk_tb <= not clk_tb after 500 ms;
35   process
36   begin
37     reset_tb <= '0'; load_tb <= '1'; entrada_tb <= "000100";
38     wait for 5 sec;
39     reset_tb <= '1';
40     wait for 1 sec;
41     reset_tb <= '0'; |entrada_tb <= "001010";
42     wait for 11 sec;
43     reset_tb <= '1';
44     wait;
45   end process;
46 end architecture RTL;

```

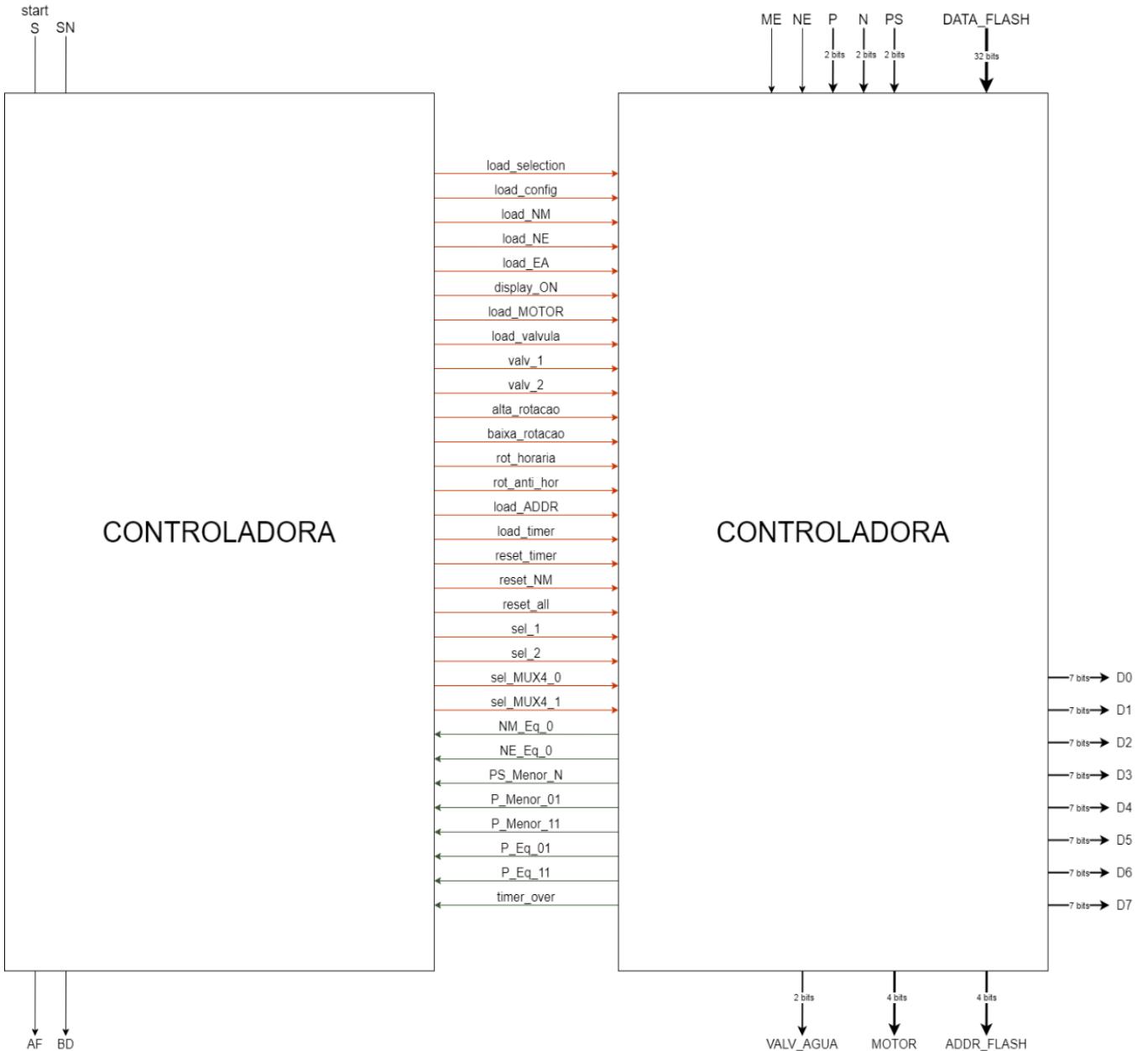
SIMULAÇÃO:



2.3. DESIGN DO BLOCO DE CONTROLE

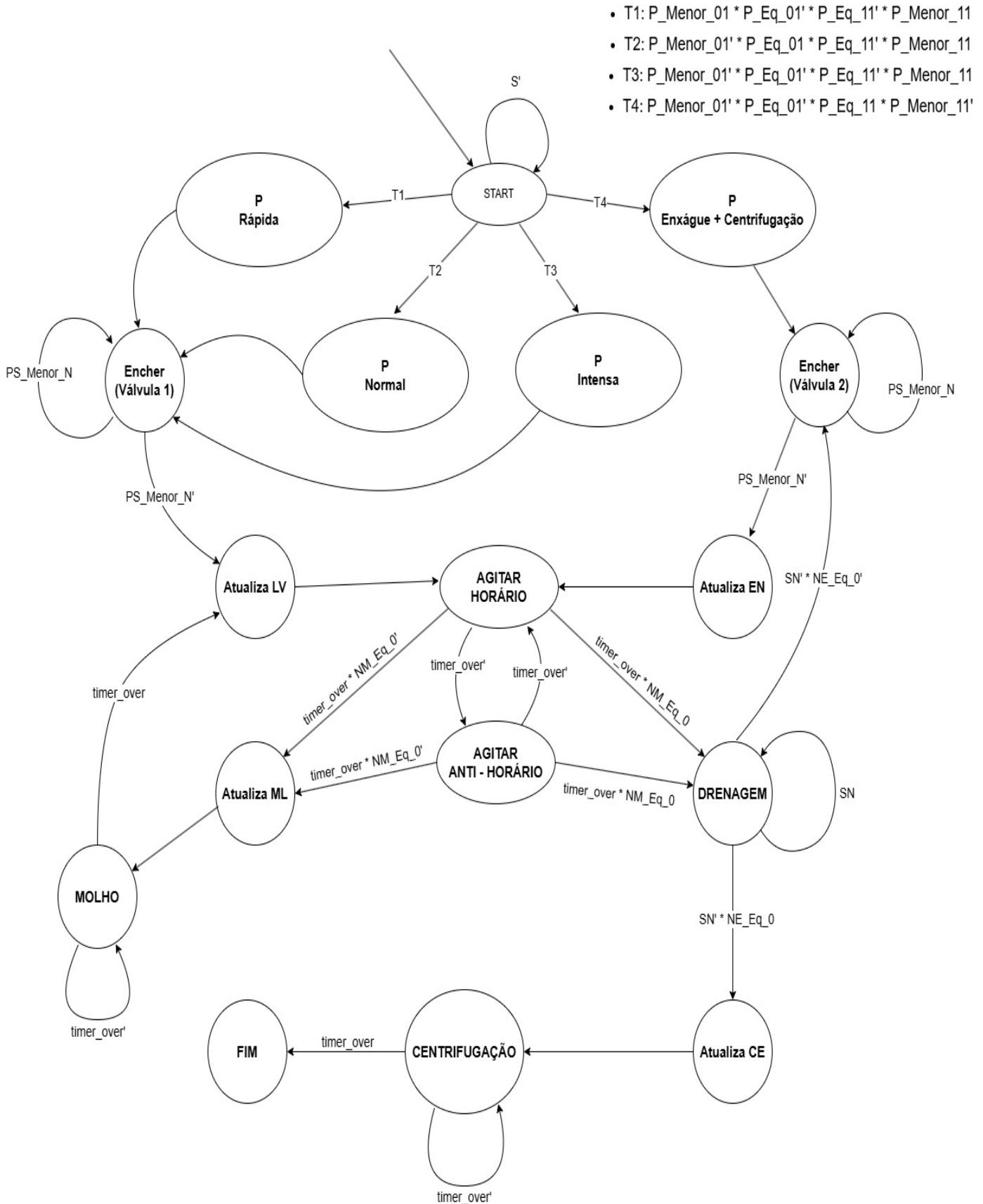
2.3.1. CONEXÃO DO DATAPATH À CONTROLADORA

A figura 4 a seguir mostra a conexão entre o Caminho de Dados e o Bloco de Controle:



2.3.2. FSM DE BAIXO NÍVEL DA CONTROLADORA

Descrição de algumas transições:



2.3.3. DESCRIÇÃO DA FSM DE BAIXO NÍVEL DA CONTROLADORA

Em **start**, permanecem ativos: `load_select`, para permitir a seleção do programa de lavagem; `load_ADDR_FLASH`, para comunicar com a memória flash, registrando o valor de `P` (programa) em 4 bits, completando com zeros, se tornando o endereço; `reset_all`, para resetar os componentes; e `reset_NM`, para resetar o número de molhos antes de começar um novo programa.

Dos **programas de lavagem**, $P=00 - P=11$, mantém-se ativos: `load_config`, `load_NM` e `load_NE`, para registrar as configurações dos programas. Adicionalmente, em $P=11$ (`P_Enxagua_Centrífuga`), `reset_NM` é ativado, para garantir que o programa não execute a etapa de molho, pois é objetivo desse programa que tal tarefa não seja executada.

Para cada um dos estados `encher_valvN`, ativa-se `load_valvula`, para registrar o acionamento, e a respectiva saída para a válvula N.

Em todos os estados `atualiza_ESTADO` temos `load_etapa_atual=1`, para registrar o número da etapa.

Para `atualiza_LV` (atualiza lavagem), estado seguinte ao acionamento das válvulas, mantém-se o `load_valvula`, para assegurar a transição de `valv_N` de 1 para 0, para ambas as válvulas. Além disso, fazemos com que `reset_timer=1` para zerar o tempo do timer

Em `atualiza_EN` (atualiza enxágue), mantemos `load_valvula` e `reset_timer`, pelos mesmos motivos; além disso, ativamos `sel_2`, para possibilitar a saída do decrementador para atualizar o número de enxágues NE (por isso também ativamos `load_NE`).

Para ambas as **rotações**, mantemos o `load_timer`, para registrar os tempos; ativamos `load_motor`, para alterar o valor de `baixa_rot_motor` para nível lógico alto; além disso, ativamos o sinal que indica o sentido de rotação (`rot_horaria` ou `rot_anti_hor`).

Em `atualiza_ML` (atualiza molho), logo depois de ativar o motor, mantemos `load_motor` para desativar os sinais do motor. Ainda, ativamos `load_NM` para atualizar o número de molhos NM. Usamos `sel_1` de forma análoga ao número de enxágues, mas agora para número de molhos. Usamos `sel_MUX4_1` para acessar o tempo de molhos.

Em **molho**, usamos `load_timer` e `sel_MUX4_1` para carregar o tempo de molho no temporizador.

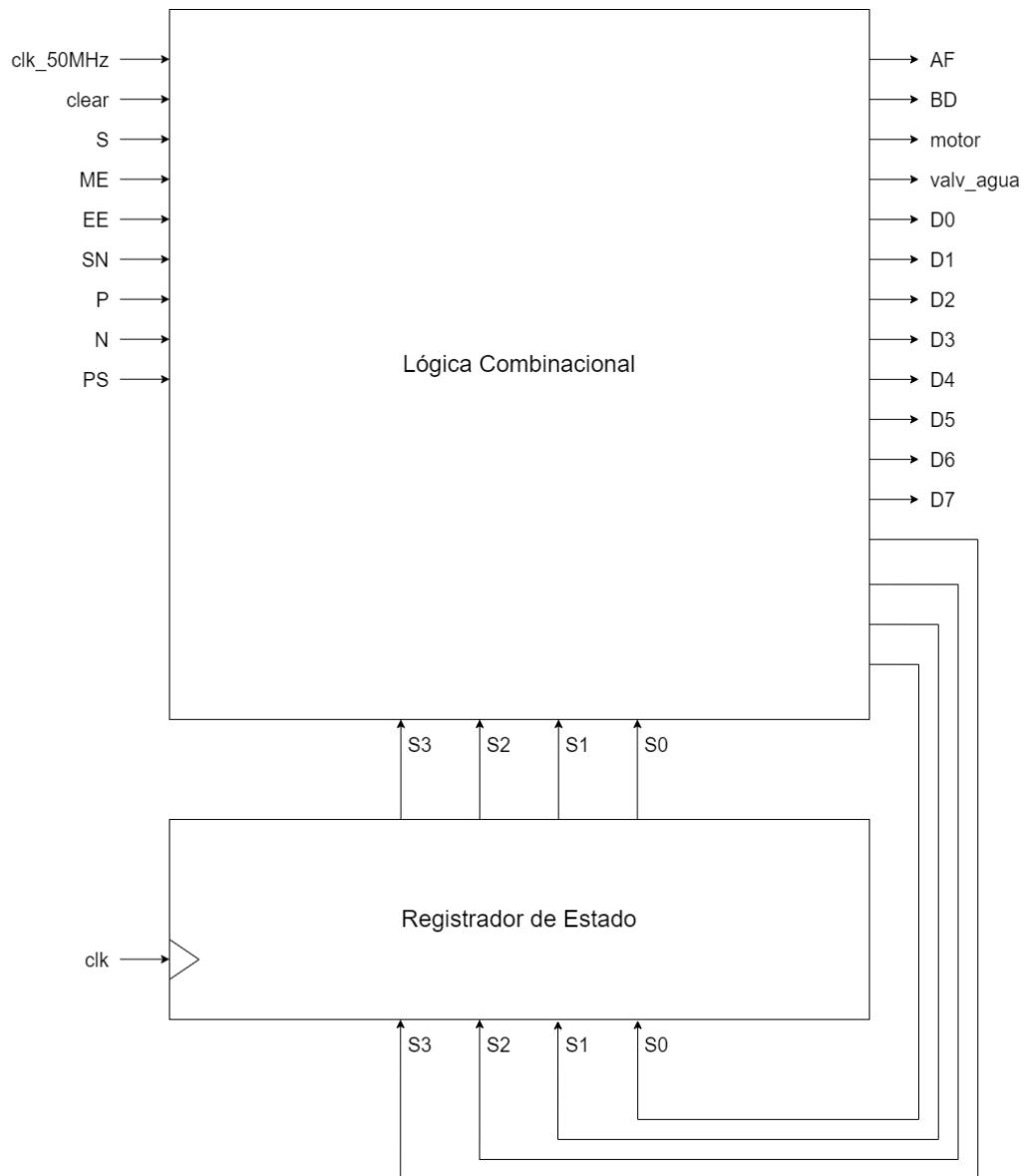
Em **drenagem**, mantemos o `load_MOTOR` para desativar o motor após o fim da agitação, e ativamos a bomba de drenagem BD.

Para `atualiza_CE` (atualiza centrifugação), assim como em `atualiza_ML`, usamos `reset_timer` para zerar o temporizador. Mantemos `sel_MUX4_1` ativado, mas agora acionamos também `sel_MUX4_0` para acessar o tempo de centrifugação.

No estado **centrifugação**, carregamos o tempo de acionamento do motor usando `load_`, `sel_MUX4_0` e `sel_MUX4_1`. Também acionamos o motor em modo 1001, ou seja, `load_motor`, `alta_rot_motor` e `rot_horaria`. Simultaneamente, acionamos a bomba de drenagem BD, para evacuar a água aspirada e o atuador do freio AF, para travar o agitador (rotor) e rotacionar apenas o cesto.

2.3.4. ARQUITETURA DO BLOCO DE CONTROLE

Para o processo de design e implementação do bloco de controle, foi feita a representação das conexões dos registradores de estado à lógica combinacional. Além disso, as entradas e saídas da FSM de operação baseada diretamente na interface com o usuário foram representadas no presente modelo.



2.4. DESIGN E INTEGRAÇÃO DO SISTEMA

2.4.1. CÓDIGOS VHDL

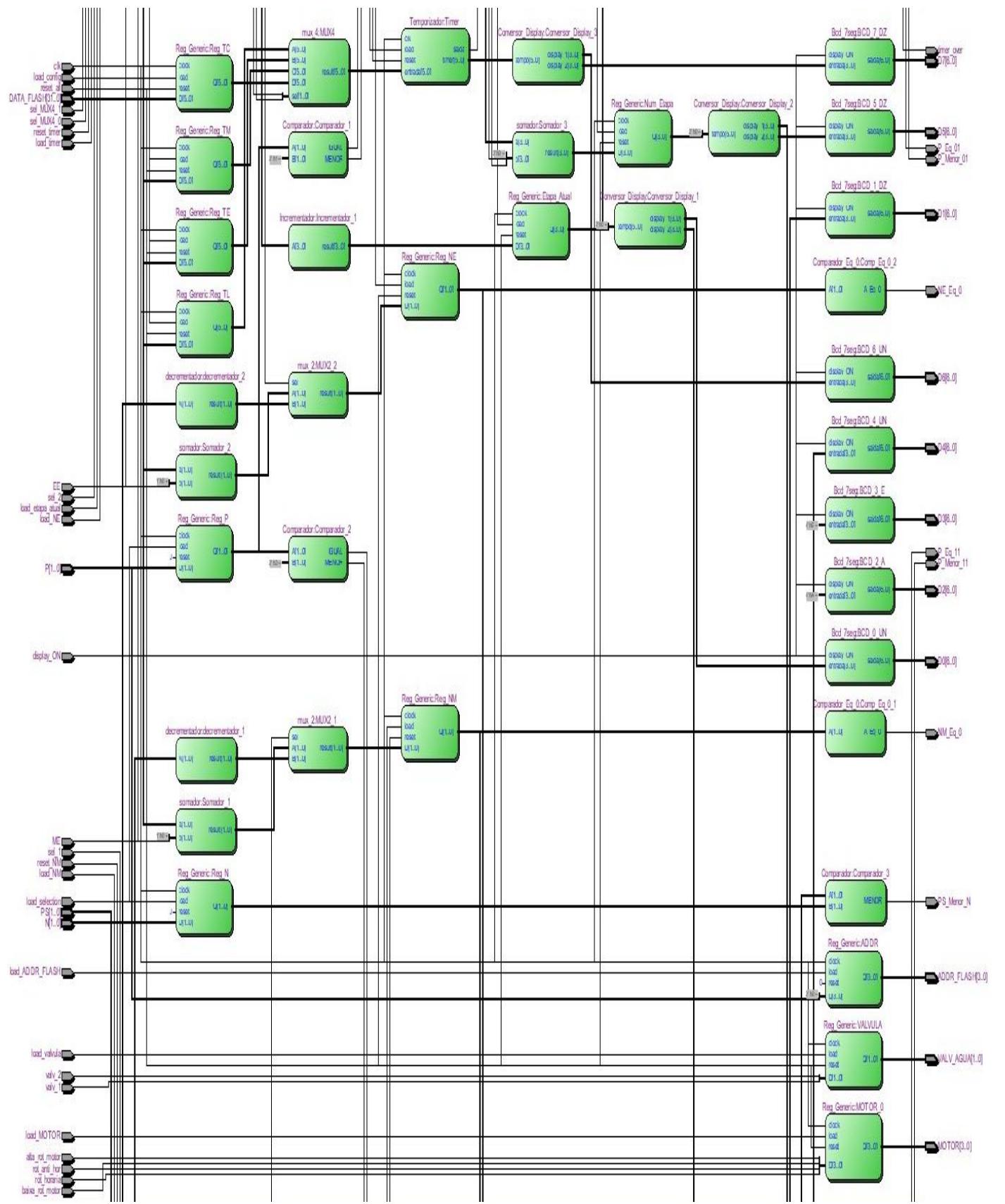
Os códigos em VHDL e arquivos do projeto encontram-se disponíveis em repositório:

CÓDIGO	LINK PARA O ARQUIVO NO GITHUB
Controladora	https://github.com/matheusmarcondes1/LabSD/blob/main/Controladora.vhd
Datapath	https://github.com/matheusmarcondes1/LabSD/blob/main/Datapath.vhd
CPU	https://github.com/matheusmarcondes1/LabSD/blob/main/CPU.vhd

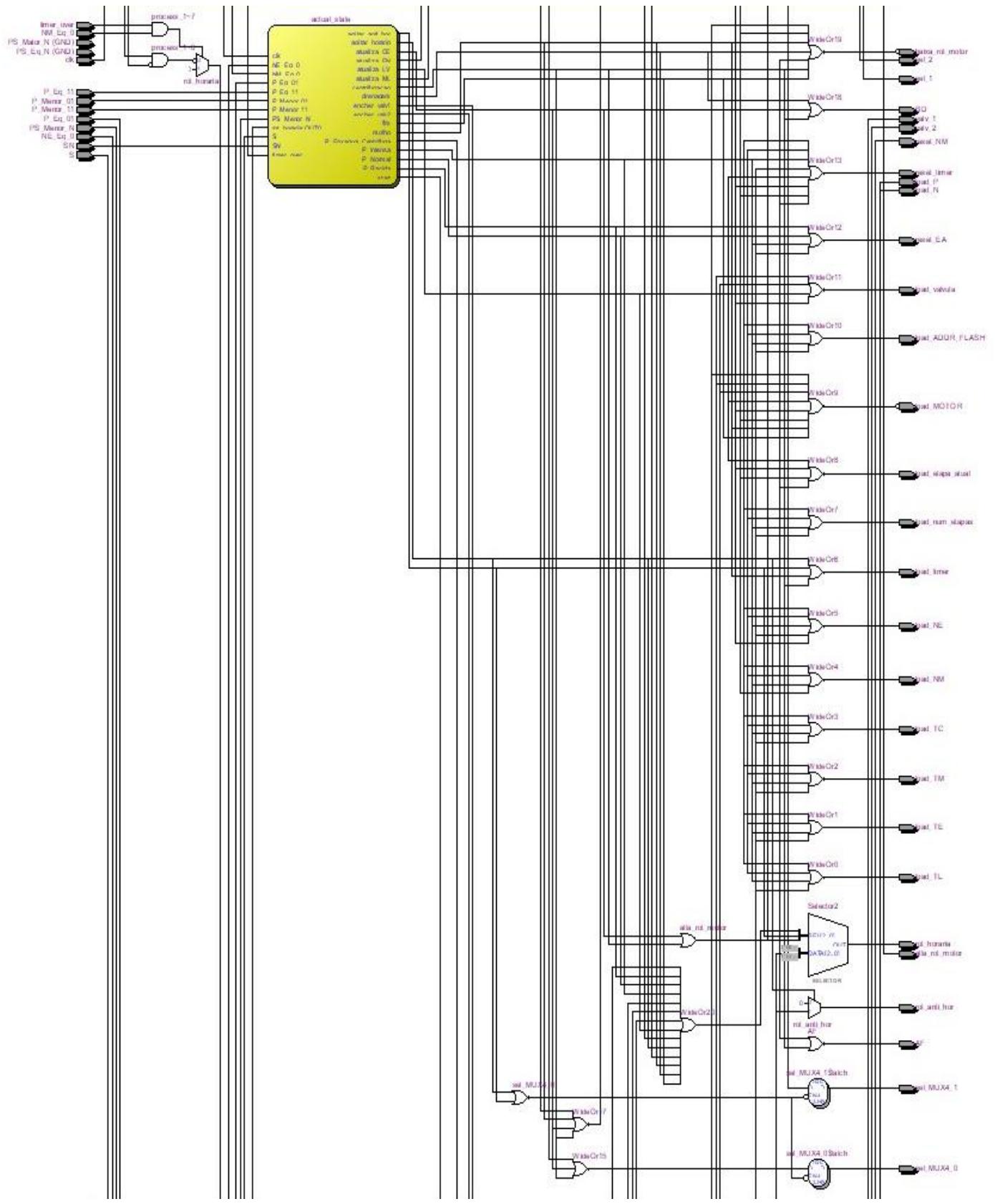
Para fins de organização da presente documentação, e para evitar complexidades que fugissem do escopo deste dossiê, optou-se por não incluir os códigos em VHDL neste arquivo, visto que estes encontram-se anexados e de fácil acesso, tanto em repositório online, quanto em arquivo zip.

2.4.2. RTL VIEWER

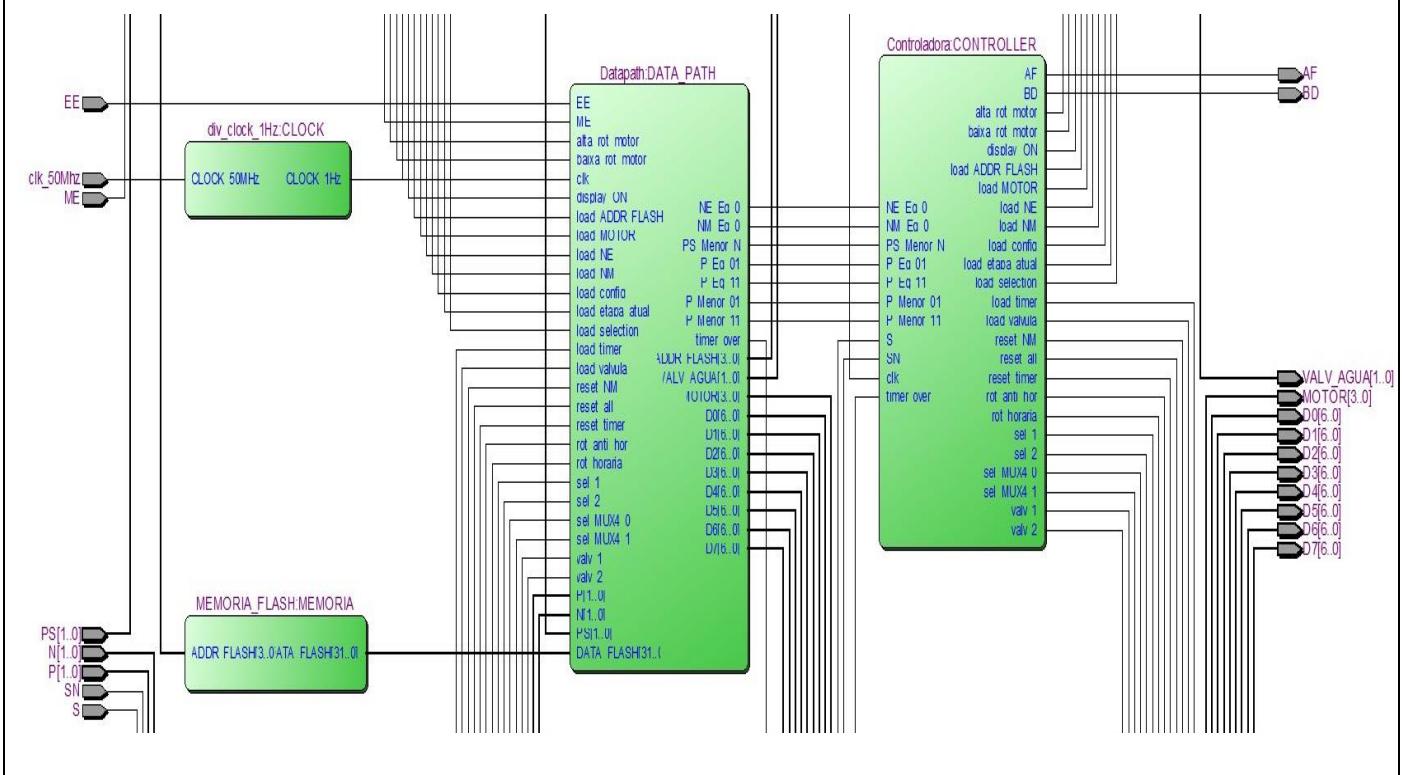
DATAPATH



CONTROLADORA



CPU



2.4.3. STATE MACHINE VIEWER

O diagrama abaixo representa a associação de estados da controladora, gerado pelo Quartus.

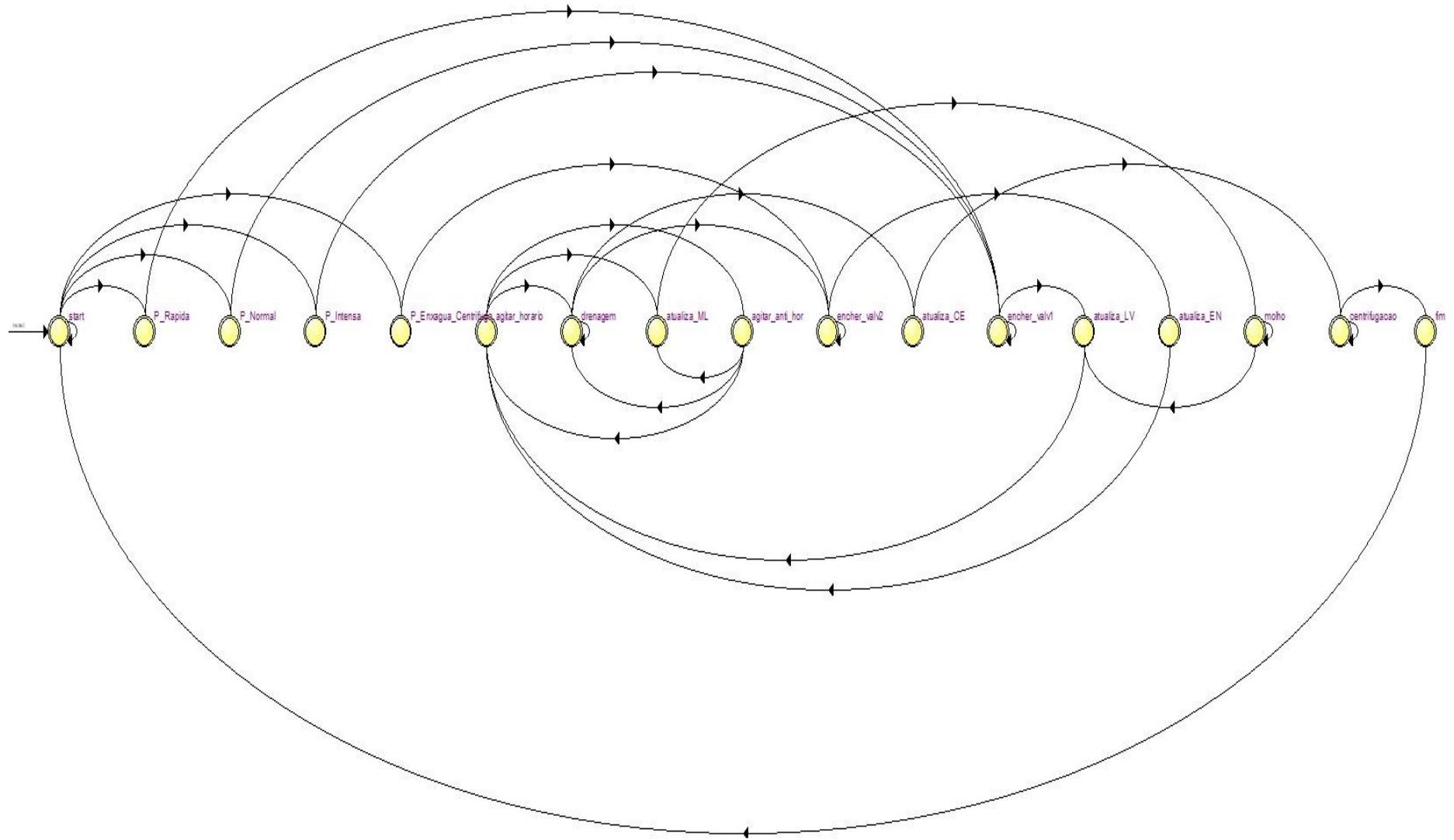


Figura 5 – State Machine Viewer

X
F
P

	Source State	Destination State	Condition
1	agitar_anti_hor	drenagem	(timer_over).(NM_Eq_0)
2	agitar_anti_hor	atualiza_DL	(timer_over).(!NM_Eq_0)
3	agitar_anti_hor	agitar_horario	(rot_horaria)
4	agitar_horario	drenagem	(timer_over).(NM_Eq_0)
5	agitar_horario	atualiza_DL	(timer_over).(!NM_Eq_0)
6	agitar_horario	agitar_anti_hor	(rot_horaria)
7	atualiza_CE	centrifugacao	
8	atualiza_EN	agitar_horario	
9	atualiza_LV	agitar_horario	
10	atualiza_DL	molho	
11	centrifugacao	fim	(timer_over)
12	centrifugacao	centrifugacao	(!timer_over)
13	drenagem	encher_valv2	(!SN).(!NE_Eq_0)
14	drenagem	drenagem	(SN)
15	drenagem	atualiza_CE	(!SN).(NE_Eq_0)
16	encher_valv1	encher_valv1	(PS_Menor_N)
17	encher_valv1	atualiza_LV	(!PS_Menor_N)
18	encher_valv2	encher_valv2	(PS_Menor_N)
19	encher_valv2	atualiza_EN	(!PS_Menor_N)
20	fim	start	
21	molho	molho	(!timer_over)
22	molho	atualiza_LV	(timer_over)
23	P_Enxagua_Centrifuga	encher_valv2	
24	P_Intensa	encher_valv1	
25	P_Normal	encher_valv1	
26	P_Rapida	encher_valv1	
27	start	start	(!S) + (S).(!P_Eq_11).(!P_Eq_01).(!P_Menor_01).(!P_Menor_11)
28	start	P_Rapida	(S).(P_Menor_01)
29	start	P_Normal	(S).(P_Eq_01).(!P_Menor_01)
30	start	P_Intensa	(S).(!P_Eq_01).(!P_Menor_01).(!P_Eq_11).(P_Menor_11)
31	start	P_Enxagua_Centrifuga	(S).(P_Eq_11).(!P_Eq_01).(!P_Menor_01)

State Table

Transitions \ Encoding

Figura 6 – Tabela de transições da Controladora

x	y	Name	fim	centrifugacao	atualiza_CE	drenagem	molho	atualiza_DL	agitador_anti_hor	agitador_horario	atualiza_EN	atualiza_LV	encher_valv2	encher_valv1	P_Enxagua_Centrifuga	P_Intensa	P_Normal	P_Rapida	start
1	start		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	P_Rapida		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
3	P_Normal		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
4	P_Intensa		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
5	P_Enxagua_Centrifuga		0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
6	encher_valv1		0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
7	encher_valv2		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
8	atualiza_LV		0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
9	atualiza_EN		0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
10	agitador_horario		0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
11	agitador_anti_hor		0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
12	atualiza_DL		0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
13	molho		0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
14	drenagem		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
15	atualiza_CE		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
16	centrifugacao		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
17	fim		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figura 7 – Tabela de codificações da Controladora

2.4.4. MEMÓRIA FLASH

SIMULAÇÃO DA MEMÓRIA FLASH

Bloco “SIMULA MEMÓRIA” utilizado como uma forma de simular o comportamento pré-definido desejado da memória flash.

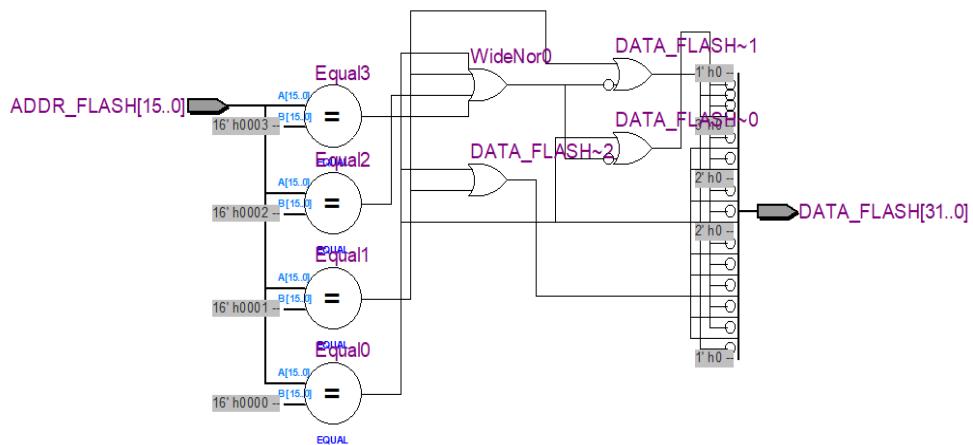
CÓDIGO VHDL:

```

1  -- Projeto de registrador tamanho genérico
2  -- com entrada reset e load síncronas
3
4  LIBRARY IEEE;
5  use ieee.std_logic_1164.all;
6
7  entity MEMORIA_FLASH is
8    port(
9      ADDR_FLASH : in std_logic_vector (3 downto 0);
10     DATA_FLASH : out std_logic_vector (31 downto 0)
11   );
12 end MEMORIA_FLASH;
13
14 architecture RTL of MEMORIA_FLASH is
15
16 begin
17   with ADDR_FLASH select
18     DATA_FLASH <=  "0101010100010100111001000001010" when "0000",
19                           "01110110000111010100001010001111" when "0001",
20                           "10001010001001011001001100010100" when "0010",
21                           "001110000001110000000001010000000" when "0011",
22                           "000000000000000000000000000000000000000000000000" when OTHERS;
23 end RTL;
24
25 --RAPIDO - LV=10, ML=15, EN=8, CE=5, NE=NM=1, NUM_ETAPAS=5
26 --NORMAL - LV=15, ML=20, EN=10, CE=7, NE=1 e NM=2, NUM_ETAPAS=5
27 --INTENSA - LV=20, ML=25, EN=12, CE=9, NE=2 e NM=2, NUM_ETAPAS=5
28 --ENX+CENT- LV=0, ML=0, EN=10, CE=7, NE=2 e NM=0, NUM_ETAPAS=5

```

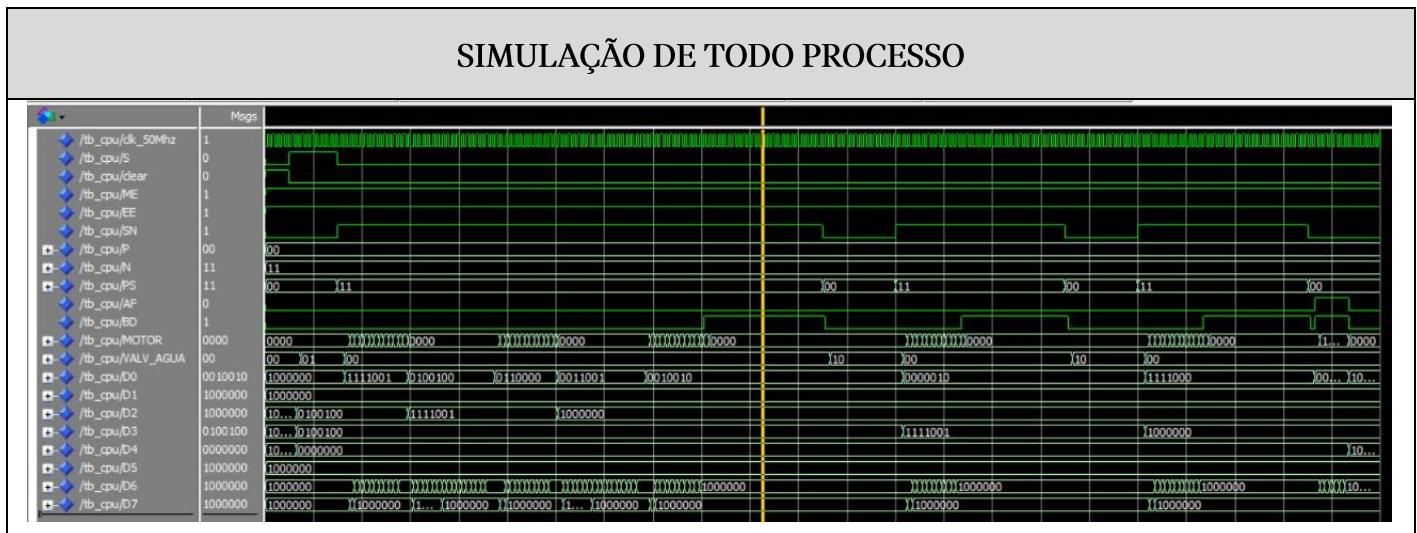
RTL VIEWER:



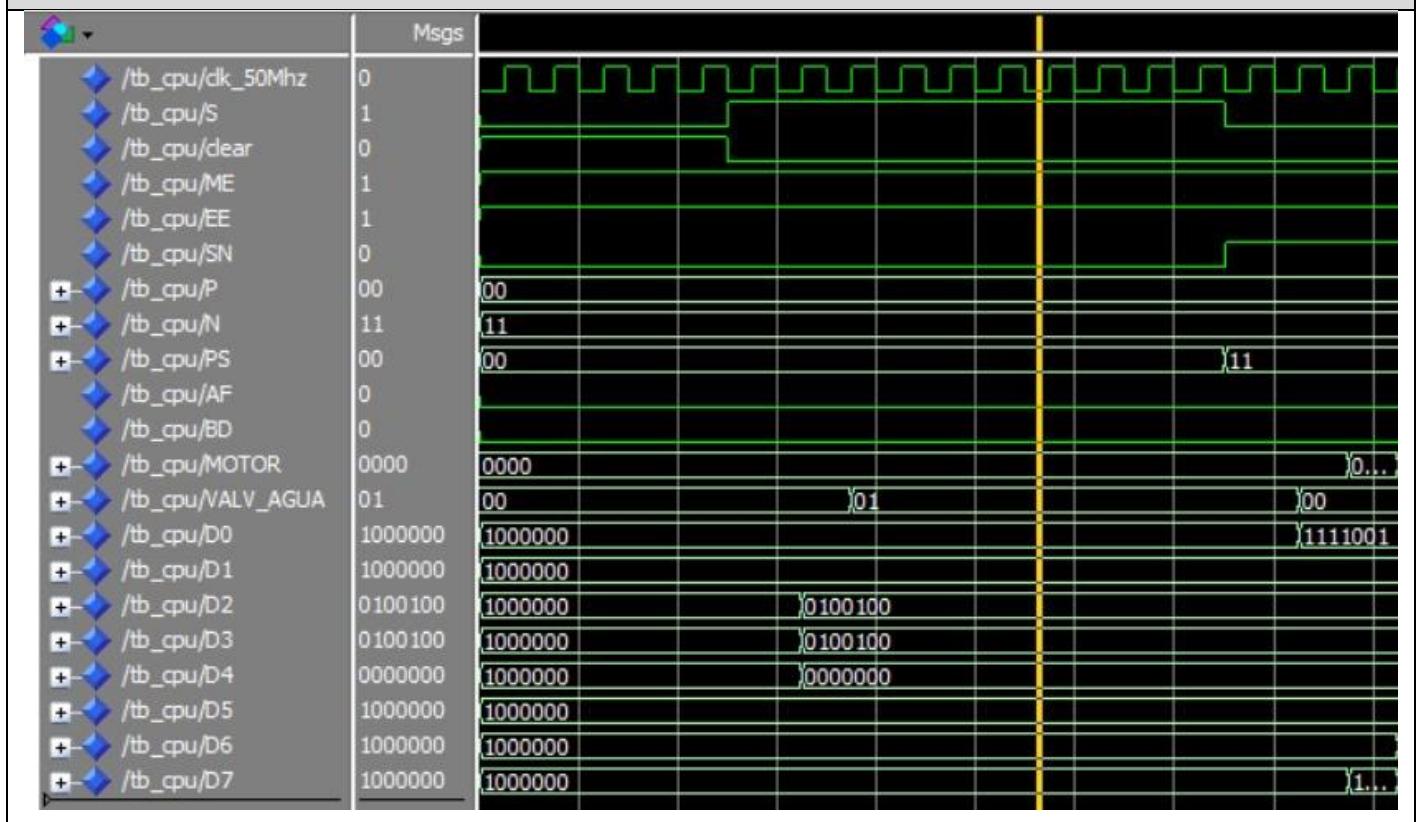
3. SIMULAÇÃO DO SISTEMA

3.1. SIMULAÇÃO VIRTUAL

Faremos agora a simulação, completa e por etapas, de um possível cenário: uma lavagem rápida com molho e enxágue extra.

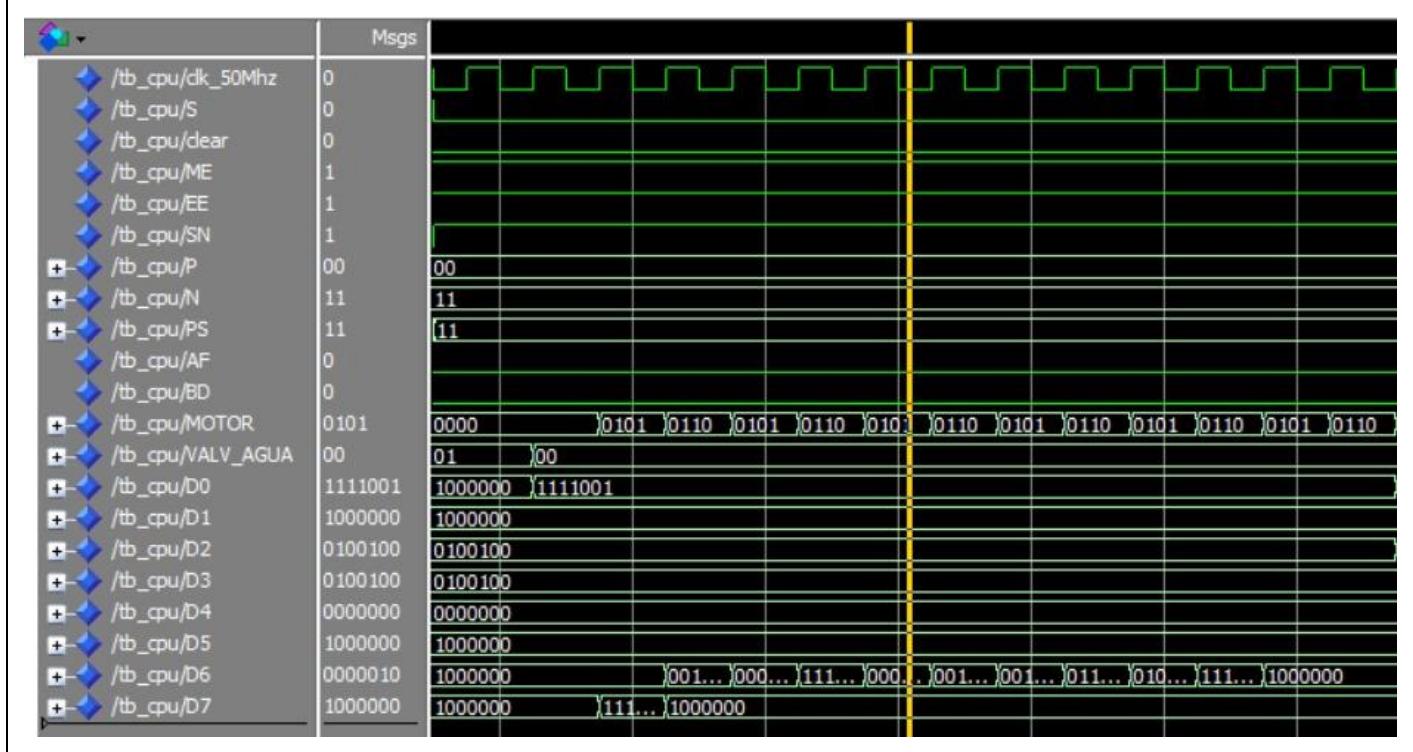


SIMULAÇÃO DO INÍCIO: VÁLVULA ENCHENDO ATÉ DESLIGAR



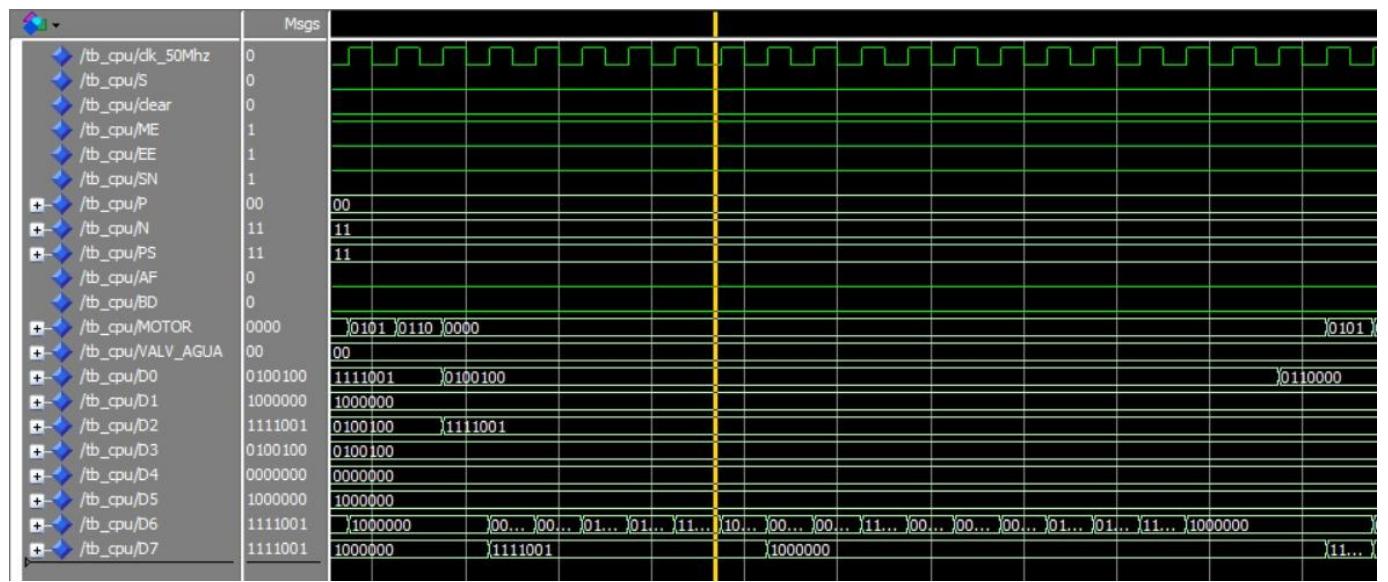
Aguarda start. Assim que é iniciada, o sistema puxa os dados da memória e a válvula 1 é acionada. Ela desarma quando o pressostato indica que atingiu o nível d'água selecionado.

SIMULAÇÃO DO MOTOR ALTERNANDO



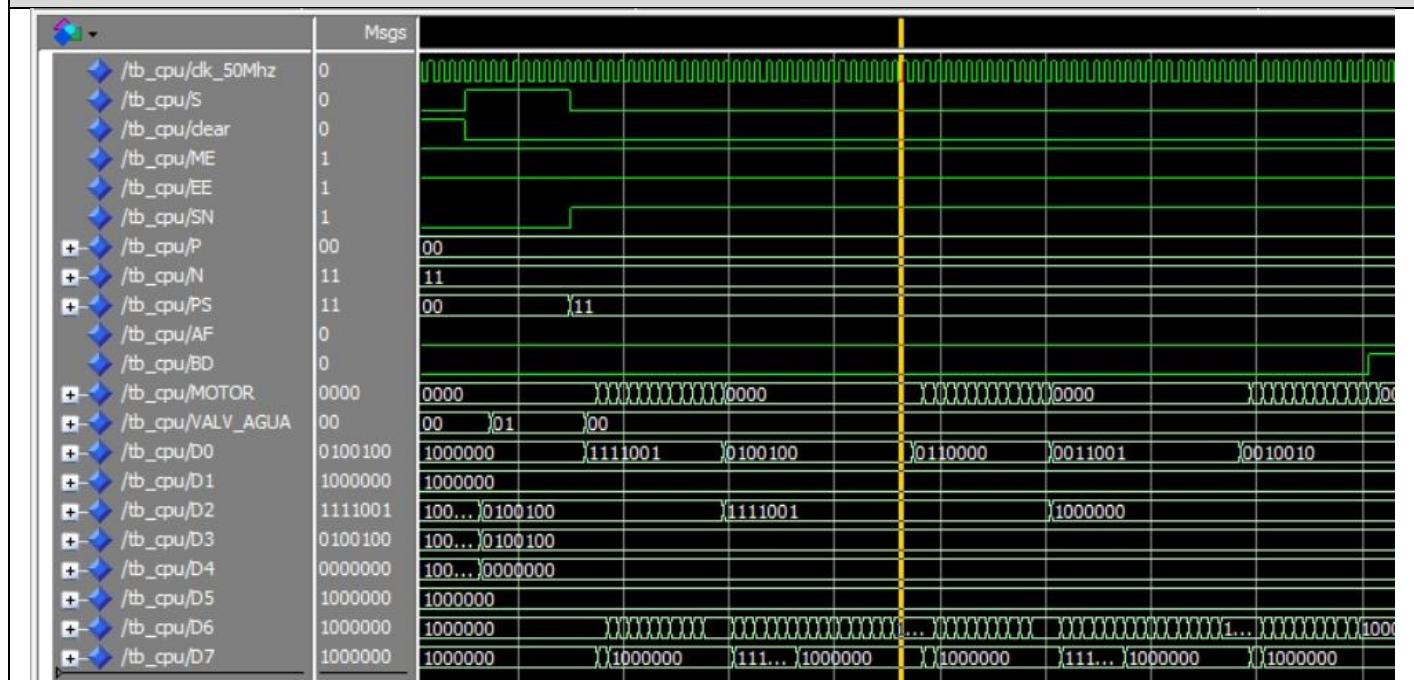
Após a válvula desligar, motor começa a trabalhar alternadamente no processo de agitação

SIMULAÇÃO DA CONTAGEM REGRESSIVA: MOLHO (MOTOR DESLIGADO)



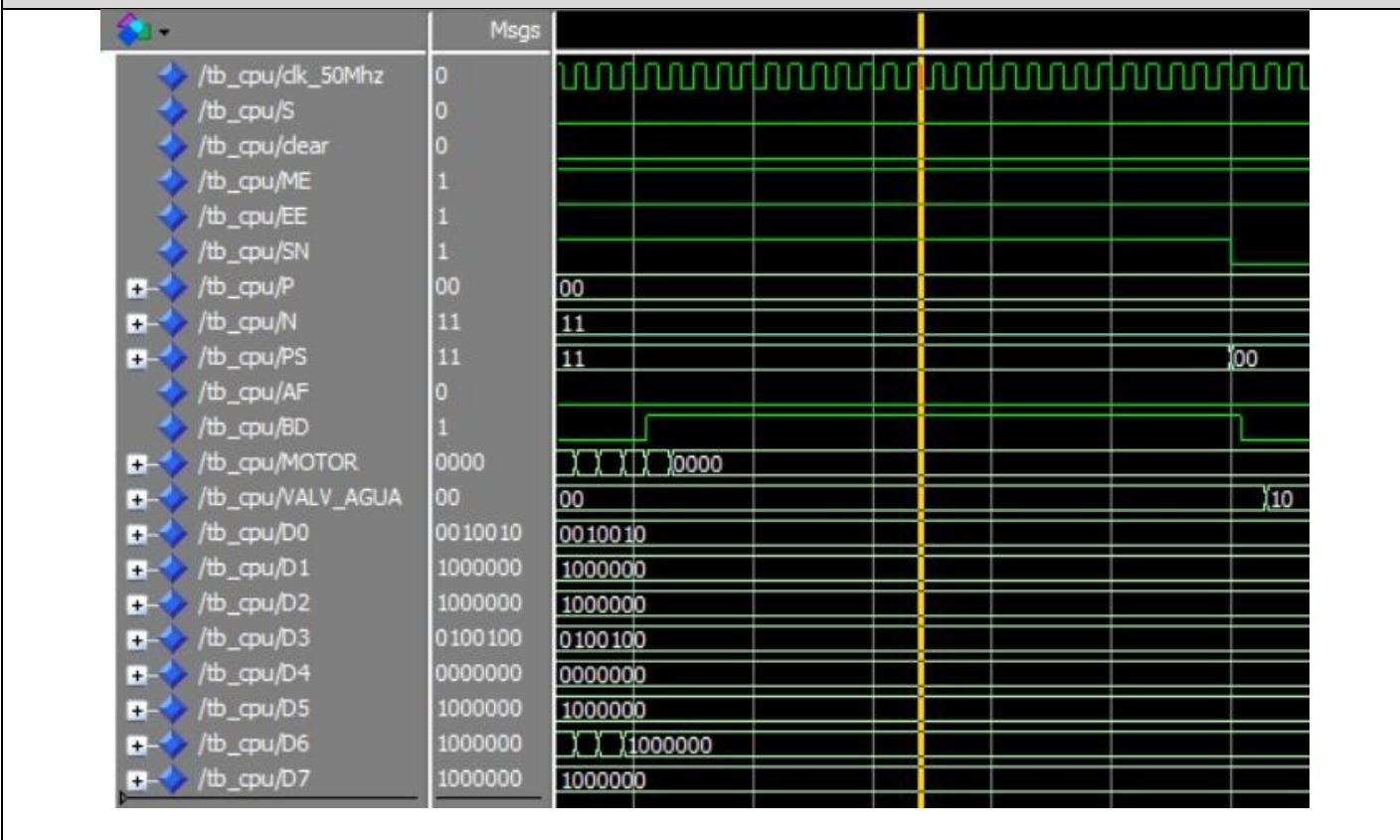
Motor desliga e etapa de molho inicia. Os valores do display mostram o tempo de molho, enquanto o motor permanece em 0000.

SIMULAÇÃO DA INTERCALAÇÃO ENTRE AGITAÇÃO E MOLHO



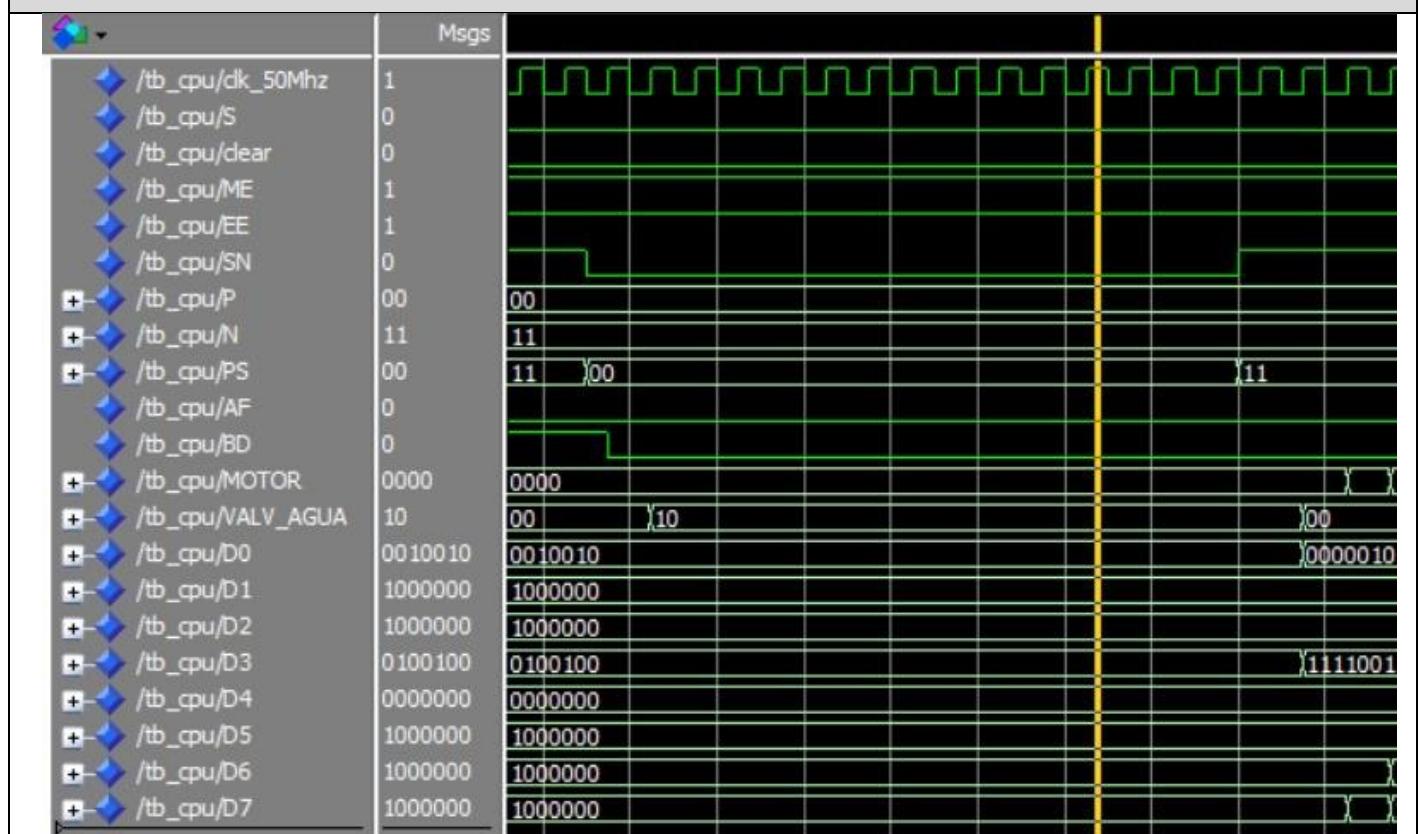
Sistema alterna entre agitação e molho até que número de molhos termine.

SIMULAÇÃO DO FIM DA AGITAÇÃO E INÍCIO DA DRENAGEM ATÉ O DESARME DA BOMBA PELO SENSOR DE NÍVEL



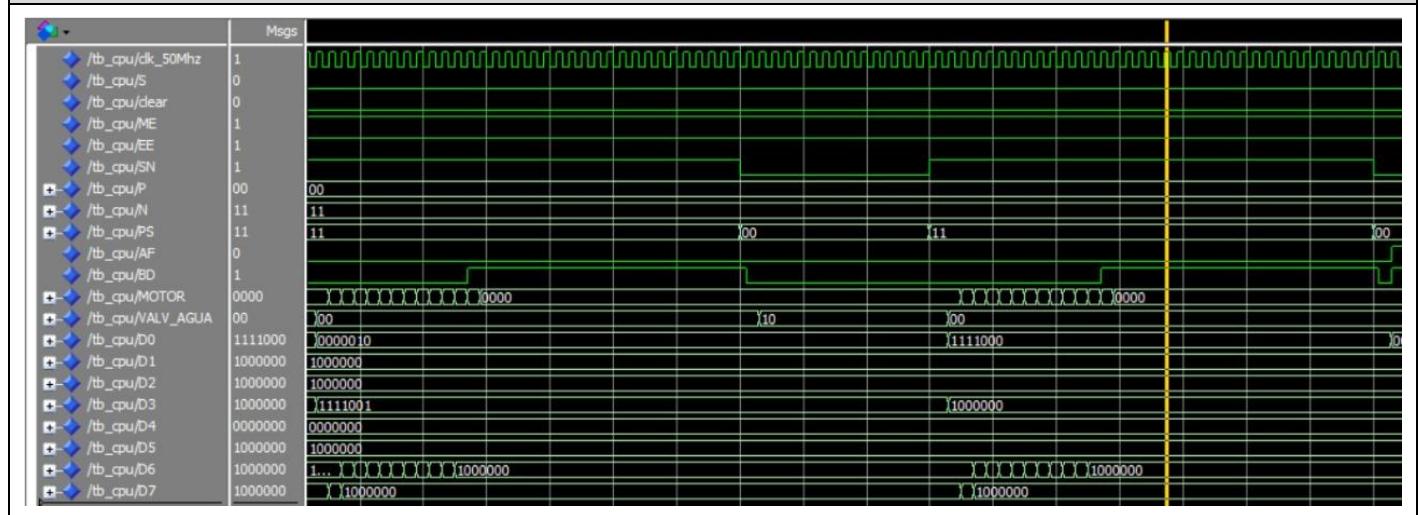
Processo de agitação chega ao fim e drenagem inicia, até que o nível d'água seja nulo, para então ser encerrada.

SIMULAÇÃO DO DESARME DA VÁLVULA DE ENTRADA D'ÁGUA PELO PRESSOSTATO



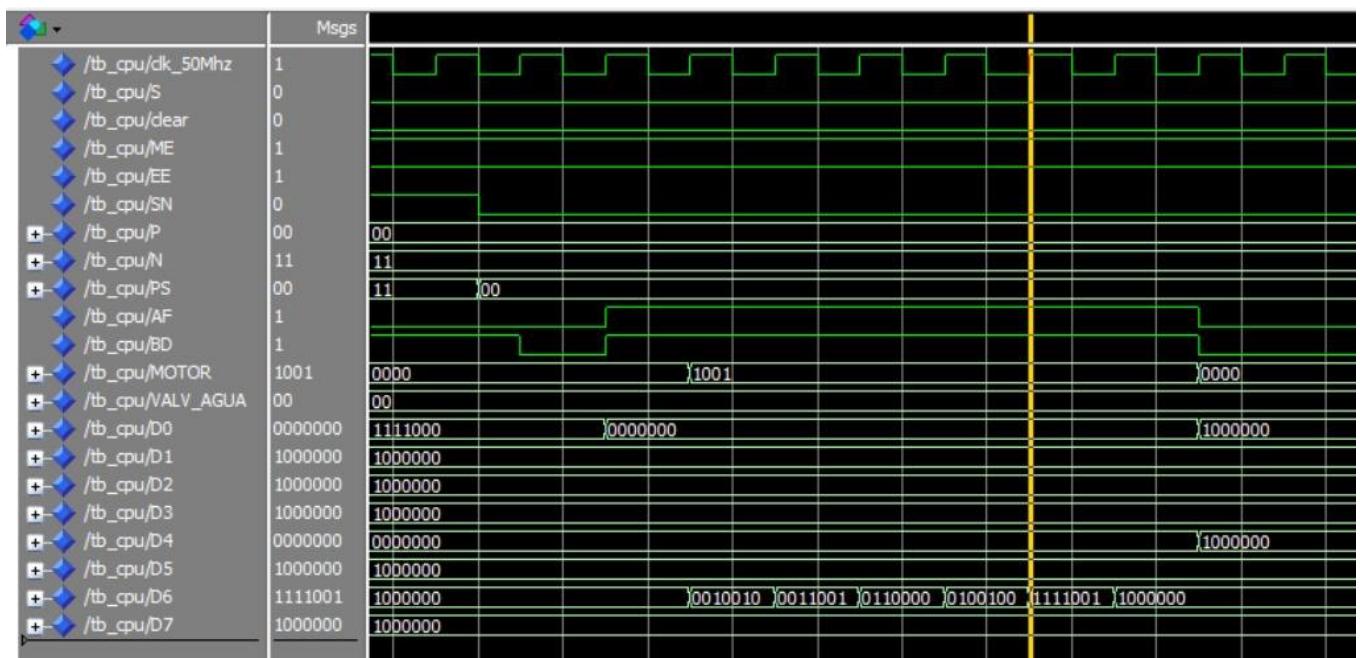
Drenagem encerra e logo após isso, válvula 2 é acionada.

SIMULAÇÃO DO INÍCIO DA CENTRIFUGAÇÃO ATÉ FINALIZAR



Processos de agitação do enxágue intercala com drenagem sempre que o tempo de agitação termina, até que o número de enxágues termine.

SIMULAÇÃO DO FIM DA LAVAGEM RETORNANDO AO START



Última drenagem é finalizada e todo o processo de centrifugação, que envolve atuador do freio AF e bomba de drenagem BD simultaneamente ativados, junto com alta rotação do motor no sentido horário até o fim da lavagem.

3.2.SIMULAÇÃO NO KIT FPGA

Segue o diagrama da representação funcional no kit, para o pin planner indexado conforme projetado:

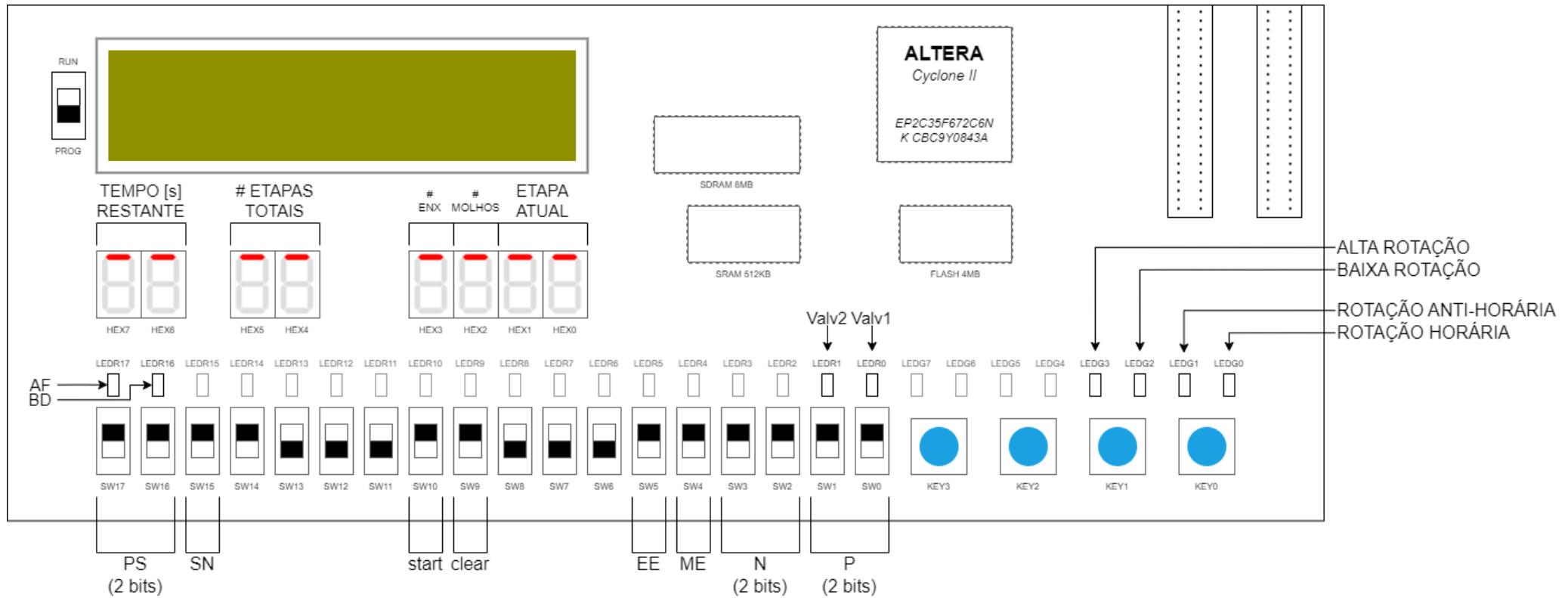


Figura 8 – Interface do usuário

4. CONCLUSÃO

O desenvolvimento de um projeto RTL de um sistema conhecido, simples e objetivo como uma máquina lavadora constitui uma excelente oportunidade para aprendizado sobre o processo de concepção de sistemas embarcados, habilidade fundamental no ensino de Engenharia. A didática permitiu a aplicação dos conhecimentos passados ao longo deste curso, e a integração, que não se limitava, com conhecimentos de outras disciplinas. Não apenas isso, mas também se combinava de forma excepcional com habilidades práticas para a implementação de sistemas digitais e soft skills.

Ao longo do projeto, destaca-se o trabalho da habilidade de observar um sistema do cotidiano e compreendê-lo na forma de uma máquina. A metodologia RTL se mostrou, mais do que aprendido em sala de aula, uma ferramenta de descrição precisa de sistemas digitais compatível e favorável com a necessidade de visão clara do fluxo de dados e atuadores.

As simulações funcionais se mostraram imprescindíveis para garantir o bom comportamento dos componentes do sistema. À medida que a complexidade do projeto aumentava, e inevitavelmente, surgiam respostas inesperadas do sistema, era constante a checagem do funcionamento de cada bloco por meio dos Testbenches na tentativa de localizar o erro.

A integração dos módulos do sistema permitiu uma compreensão maior sobre a distinção de funções de cada parte, e com isso, fortaleceu a ideia sobre a atuação da Controladora, do Caminho de Dados e da CPU. Nesse projeto, cumprimos o objetivo de implementar tal integração de forma eficiente, evidenciando a importância da compatibilidade entre os componentes do sistema. A síntese do projeto no FPGA, ainda, demonstrou a viabilidade prática do sistema desenvolvido.

Uma das questões que foram levadas em conta, foi a possibilidade de expansão do sistema, que não exigiria uma escalada de complexidade. Por exemplo, poderíamos facilmente implementar mais dois programas de lavagem, pois a modularização do sistema torna prático adicionar mais alguns poucos estados para isso. O endereçamento da memória flash permite mais dois bits a serem usados para rotular tais programas, e os componentes são o suficiente para construir inúmeras operações.

Fundamentalmente, este projeto não apenas alcançou todos os objetivos propostos, mas também propiciou uma experiência valiosa teórica e prática. Essa experiência destaca a importância de projetos e aulas que envolvem bom planejamento, e seu impacto positivo no currículo da Engenharia, proporcionando uma conexão essencial entre teoria e prática.

5. REFERÊNCIAS BIBLIOGRÁFICAS

VAHID, Frank. Sistemas Digitais: projeto, otimização e HDLs. 1^a Edição. Porto Alegre: Bookman 2008.

NA -S106F1 ® MANUAL DO USUÁRIO MÁQUINA DE Lavar Roupa. [s.l.: s.n., s.d.]. Disponível em: <<https://www.panasonic.com/content/dam/Panasonic/br/pt/PDF/NA-S106F1.pdf>>. Acesso em: 11 jul. 2024.

6. APÊNDICE A – REPOSITÓRIO

ARQUIVO	LINK PARA O ARQUIVO NO GITHUB
Projeto completo	https://github.com/matheusmarcondes1/LabSD
Controladora	https://github.com/matheusmarcondes1/LabSD/blob/main/Controladora.vhd
Datapath	https://github.com/matheusmarcondes1/LabSD/blob/main/Datapath.vhd
CPU	https://github.com/matheusmarcondes1/LabSD/blob/main/CPU.vhd
Arquivo rar	https://github.com/matheusmarcondes1/LabSD/blob/main/PROJETO%20FINAL.rar