



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA

ALLYSSON DE ANDRADE SILVA
GABRIEL CAVALHEIRO FRANCISCO
ISAAC DE LYRA JUNIOR
LUCAS BATISTA DA FONSECA

Sistemas Digitais
Grupo 04 - Problema 01 - Implementação

Natal/RN

2021



SISTEMAS DIGITAIS
GRUPO 04 - PROBLEMA 01 - IMPLEMENTAÇÃO

ALLYSSON DE ANDRADE SILVA
GABRIEL CAVALHEIRO FRANCISCO
ISAAC DE LYRA JUNIOR
LUCAS BATISTA DA FONSECA

Relatório do grupo 4, segunda semana, visando implementar a Secretária Eletrônica que foi projetado inicialmente pelo grupo 4 da primeira semana.

Docente: SAMAHERNI MORAIS DIAS

Natal/RN

2021

Sumário

	Lista de Imagens	1
1	INTRODUÇÃO	2
1.1	Objetivo.	2
1.1.1	Objetivo Geral.	2
1.1.2	Objetivos Específicos.	3
2	IMPLEMENTAÇÃO	4
2.1	Correções do projeto	4
2.1.1	Tabela de Transição e Máquina de Estados Corrigidos	4
2.1.2	Bloco Operacional Corrigido	6
2.2	Bloco de Controle	8
2.3	Display	9
2.4	Temporizadores	10
2.4.1	2 segundos e 5 segundos.	10
2.4.2	60 segundos	11
2.5	Contador de Escrita/Leitura	12
2.6	Bloco de Registro	13
2.6.1	Contador das Amostras	14
2.6.2	Memória RAM	15
2.7	Bloco Operacional	16
3	RESULTADOS	18
4	CONCLUSÃO	21
	REFERÊNCIAS	22
	ANEXO A – CÓDIGO VHDL	26

Lista de Imagens

Figura 1 – Interface homem-máquina.	2
Figura 2 – Elementos da interface homem-máquina no sistema digital.	2
Figura 3 – Máquina de Estados Corrigida	4
Figura 4 – Tabela transição de estados.	5
Figura 5 – Habilitador de contagem.	6
Figura 6 – Bloco Operacional antigo	7
Figura 7 – Bloco Operacional	7
Figura 8 – Tags da máquina de estados.	8
Figura 9 – Definição de estados.	8
Figura 10 – Exemplo da implementação do estado <i>Tela Inicial</i>	8
Figura 11 – Código dos 7 segmentos do display.	9
Figura 12 – Displays recebendo o valor.	9
Figura 13 – Código dos temporizadores 2seg e 5seg.	10
Figura 14 – Código do temporizador de 60 segundos.	11
Figura 15 – Código de leitura.	12
Figura 16 – Código de escrita.	13
Figura 17 – Código do contador de amostras.	14
Figura 18 – Código da memória RAM.	15
Figura 19 – Projeto do bloco operacional.	16
Figura 20 – Entidade do bloco operacional.	16
Figura 21 – 25 Blocos de registro.	17
Figura 22 – Port map dos blocos de registro.	17

1 INTRODUÇÃO

1.1 Objetivo.

1.1.1 Objetivo Geral.

A secretária eletrônica é um recurso que permite a gravação de mensagens mediante a ausência de algum indivíduo para atender o telefone. Em empresas é um meio útil na captação de demandas realizadas fora do horário comercial, já nas casas de família possibilita a gravação de recados ou lembretes por familiares e amigos.

Diante disso, o seguinte trabalho tem como objetivo projetar um sistema digital que possibilite o funcionamento de uma secretária eletrônica. Esta poderá salvar até 25 mensagens, onde cada mensagem tem, no mínimo, 10 amostras do conversor A/D que dizem respeito à amplitude de um sinal de tensão de um microfone que capta a voz humana. Cada uma dessas amostras possui 8 bits.

A interface homem-máquina é a mostrada na Figura 1, na Figura 2 temos a descrição de cada um dos seus elementos.

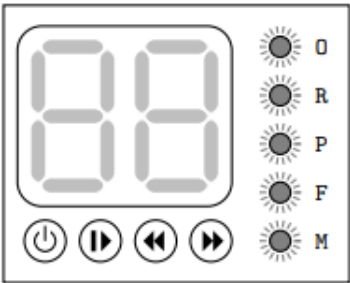


Figura 1 – Interface homem-máquina.







Elemento	Descrição
	Display principal (exibição de valor entre 0 e 25)
	Botão ON (inicializar/repouso/reset - tipo <i>pushbutton</i>)
	Botão PLAY (reprodução/pausa - tipo <i>pushbutton</i>)
	Botão BACK (navegação decremental - tipo <i>pushbutton</i>)
	Botão NEXT (navegação incremental - tipo <i>pushbutton</i>)
	Led (O: em funcionamento; R: gravando; P: reproduzindo; F: cheio; M: tem mensagem)

Figura 2 – Elementos da interface homem-máquina no sistema digital.

1.1.2 Objetivos Específicos.

O sistema tem um funcionamento pré-definido que utiliza, ou não, os botões para mudar as funções e a interface do sistema. Dessa forma, a secretária eletrônica está desligada quando todos LEDs e o display estão apagados, para ligar o sistema o pushbutton ON deve ser pressionado, nesse momento o led O é ativado e permanecerá assim sempre que o circuito integrado estiver ativo, além disso, o display será ativado exibindo o número de mensagens guardadas. Esta é a interface da tela inicial.

Se, da tela inicial, o pushbutton ON for pressionado ou nenhum outro botão seja pressionado por 60 segundos o sistema entrará em repouso, assim, o led O permanecerá acionado mas o display ficará desligado, se houver mensagens gravadas o led M continuará acionado, assim como, o led F permanecerá acionado se a secretária eletrônica estiver cheia de mensagens, os demais leds ficarão apagados.

A gravação das mensagens será sinalizada pelo led R ativo, nesse momento o display ficará desligado. Após a gravação da mensagem o sistema voltará a ficar em repouso. Para reproduzir todas as mensagens armazenadas, o sistema deve estar na tela inicial (caso esteja em repouso, basta pressionar o pushbutton ON) a qual estará exibindo o número total de mensagens, ao pressionar o pushbutton PLAY todas as mensagens serão reproduzidas, da mais antiga à mais nova. Dessa forma, o display ficará exibindo qual mensagem está sendo reproduzida no momento e o led P ficará ativo durante toda a reprodução. Quando a última mensagem for reproduzida, o led P voltará a ficar desligado e o sistema voltará para a tela inicial.

Há a opção de não ouvir todas as mensagens de uma vez e selecionar quais ouvir, para isso há os pushbuttons BACK e NEXT que permitem a navegação das mensagens. Quando a mensagem for escolhida e o pushbutton PLAY for pressionado, somente a mensagem selecionada será reproduzida. Durante esse processo, o led P permanecerá acionado. Caso haja o interesse de ouvir outra mensagem é só repetir o processo mencionado, caso queira sair da tela de reprodução é só pressionar o pushbutton ON e ir para tela de repouso.

Para apagar todas as mensagens da secretária eletrônica, o usuário precisa pressionar o pushbutton ON por 2 segundos. A confirmação do reset será dado pela exibição de '00' no display e dos leds F e M desligados. Por fim, para desligar o sistema o usuário precisa pressionar o pushbutton ON por 5 segundos.

Diante de tal demanda, o relatório a seguir irá propor a implementação do sistema, anteriormente projetado pelo antigo grupo 4, utilizando os conceitos de máquina de estados; projeto RTL e memória RAM.

Figura 3 – Máquina de Estados Corrigida

O principal problema que encontramos durante a implementação da maquina de estados foi referente a quando os contadores referentes as 25 mensagens seriam acionados, para que assim pudéssemos direcionar a leitura ou escrita a o novo componente de memoria, visto que neste projeto cada mensagem ocupa uma memoria do tipo RAM, pois durante a leitura e escrita de cada uma das 10 amostras que formam as mensagens os contadores continuavam acionados o que acarretava a escrita ou leitura de cada amostra em uma mensagem diferente.

Para corrigir esse problema o grupo decidiu adicionar novos estados, os quais chamamos de Process_r1, Process_r2 e Process_w, os quais são responsáveis por habilitar a contagem para redirecionar para o proximo slot de mensagem após o fim da escrita ou da leitura no caso dos Process_w e Process_r2 respectivamente, e durante o estado de navegação, onde o usuário da secretária eletrônica seleciona qual das mensagens irá reproduzir por meio do Process_r1, o qual habilita o contador up/down; Em todas as variáveis Process adicionadas, permanecemos por apenas um pulso de clock, o que nos proporciona apenas um incremento ou decremento por vez, solucionando assim o problema inicial que encontramos.

Outro problema que o grupo pode identificar foi a não garantia de se reproduzir a partir da primeira mensagem quanto o usuário pressionasse o botão play estando na tela inicial, o que pela descrição dos parâmetros que o projeto precisava atender disponibilizadas pelo professor, deveria reproduzir todas as mensagens salvas em ordem crescente a partir da primeira, para isso foi adicionado o estado Reseta Contador na transição do estado inicial ao estado reprodução total, e nesse novo estados resetamos o contador de leitura para o valor inicial. Quando no estado reprodução total a maquina fica alternando entre este e ouvir tudo, tendo o estado Preocess_r2 como o responsável por somar 1 no numero da mensagem a ser lida após a execução anterior até a chegada do momento onde o numero da mensagem que executada seja igual ao numero total de mensagens, acionando a flag comp_eq a qual nos redireciona a tela inicial.

Após as alterações feitas na maquina de estados o grupo refez a tabela de transição como uma maquina do tipo Moore, a qual pode ser vista na figura 4.

VARIÁVEIS DE ESTADO																								
INPUT												OUTPUT												
ESTADO ATUAL	B_ON	PLAY	REC	COMP_EQ	REC	B_CLOCK	F	T0	T1	T2	ESTADO FUTURO	LED_0	LED_1	LED_2	REF_W	REF_R	REF_S	DISPLAY	HC_R	HC_W	MEMORY	REV		
DESLIGADO	1	0	0	0	0	0	0	0	0	0	DESLIGADO	0	0	0	0	0	0	0	0	0	0	0		
	0	1	0	0	0	0	0	0	0	0	TELA INICIAL	1	0	0	0	0	0	1	0	0	0	0		
	0	0	0	0	0	0	0	0	0	0	RESE	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	DESLIGADO	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPOUSO	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	PROCESS_R1	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	RESE CONTADOR	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	TELA INICIAL	1	0	0	0	0	0	1	0	0	0			
REPRODUÇÃO TOTAL	1	0	0	0	0	0	0	0	0	0	OUVIR TUDO	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	OUVIR UM	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	TELA INICIAL	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPRODUÇÃO INDIVIDUAL	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	PROCESS_R1	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	OUVIR UM	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPOUSO	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	DESLIGADO	1	0	0	0	0	0	0	0	0	0			
REPOUSO	0	0	0	0	0	0	0	0	0	0	GRAVAR	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPOUSO	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	TELA INICIAL	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	OUVIR TUDO	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	OUVIR UM	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	PROCESS_R1	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	OUVIR UM	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPRODUÇÃO INDIVIDUAL	1	0	0	0	0	0	1	0	0	0			
OUVIR UM	0	0	0	0	0	0	0	0	0	0	TELA INICIAL	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	GRAVAR	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	PROCESS_R1	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	PROCESS_R2	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	RESET CONTADOR	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPRODUÇÃO TOTAL	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPOUSO	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	DESLIGADO	1	0	0	0	0	0	0	0	0	0			
RESET	0	0	0	0	0	0	0	0	0	0	TELA INICIAL	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	PROCESS_R1	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	PROCESS_R2	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPRODUÇÃO INDIVIDUAL	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPRODUÇÃO TOTAL	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	REPOUSO	1	0	0	0	0	0	1	0	0	0			
	0	0	0	0	0	0	0	0	0	0	OUVIR TUDO	1	0	0	0	0	0	0	0	0	0			
	0	0	0	0	0	0	0	0	0	0	OUVIR UM	1	0	0	0	0	0	0	0	0	0			

Figura 4 – Tabela transição de estados.

2.1.2 Bloco Operacional Corrigido

No bloco operacional teve alterações em relação as variáveis habilitadoras de contagem e de memória. Na Figura 5 pode ser visto como a variável de habilitar contagem era definida antes, onde era utilizado as saídas do bloco de controle Led_P e Memory, isto por si só fazia a liberação de contagem em momentos que não deveria, pois nos estados de "Ouvir Tudo" e "Ouvir Um" essas saídas eram definidas em nível lógico alto, ou seja, enquanto era lida uma mensagem, eu alterava também o endereço da mensagem que eu estava lendo a cada pulso de clock, fazendo com que minha saída fosse apenas a primeira amostra de todas as mensagens armazenadas na máquina. Para solucionar esse problema toda essa lógica foi substituída por duas novas saídas do bloco de controle denominadas de HC_R e HC_W, para habilitar contagem de leitura e escrita, respectivamente, por apenas um pulso de clock.

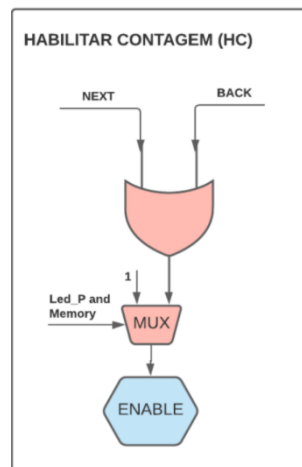


Figura 5 – Habilitador de contagem.

Ainda em relação ao bloco operacional, é possível ver na Figura 6 o bloco operacional projetado, tal bloco foi modificado e dividido em dois, o motivo da divisão é que, na implementação ficou confuso a utilização das saídas dos contadores, logo, para resolver isso foi separa o bloco operacional projetado em um bloco responsável por fornecer o endereço ao bloco de registro de 25 memórias e um bloco responsável por fornecer o número de mensagens ao display.

Além da separação já mencionada e das variáveis habilitadoras dos contadores de leitura e escrita, houve também uma mudança no *reset* do contador de leitura, o grupo achou desnecessário a utilização de um mux e definiu uma saída do bloco de controle apenas para resetar o contador, denominada de *RST_R*. O resultado da separação e das modificações mencionadas é possível visualizar na Figura 7a e 7b.

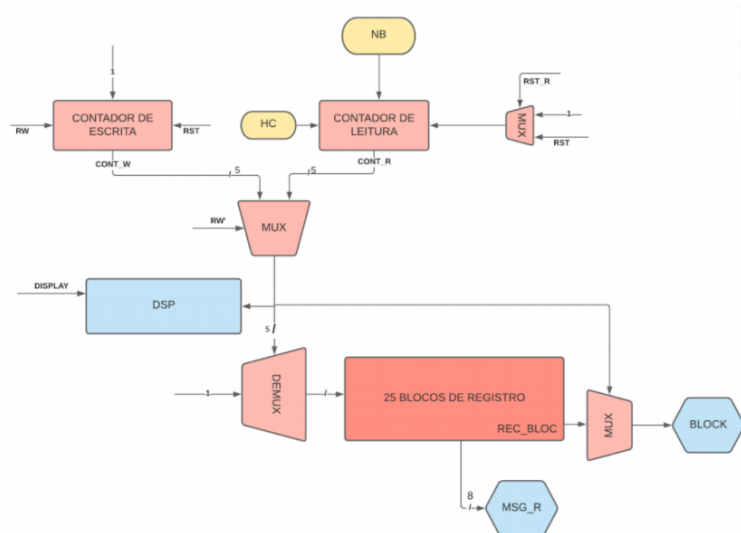
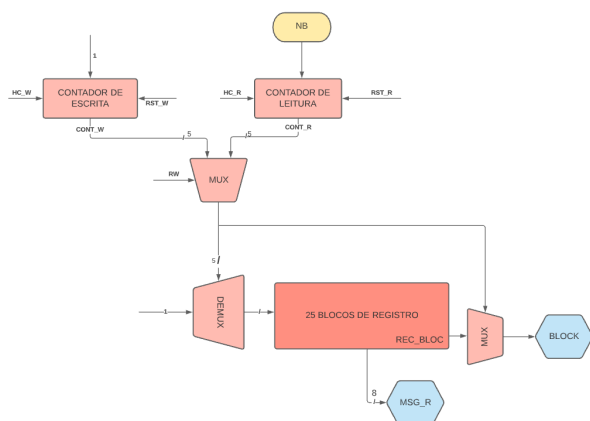
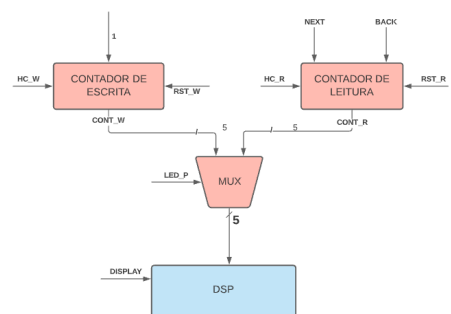


Figura 6 – Bloco Operacional antigo



(a) Bloco responsável pela escrita e leitura.



(b) Bloco do display.

Figura 7 – Bloco Operacional

2.2 Bloco de Controle

No bloco de controle, primeiro criamos uma tag para cada um dos estados, inclusive os novos estados que foram adicionados na parte de correção.

```
--DE: DESLIGADO
--TI: TELA INICIAL
--RT: REPRODUÇÃO TOTAL
--RI: REPRODUÇÃO INDIVIDUAL
--RE: REPOUSO
--OT: OUVIR TUDO
--OU: OUVIR UM
--RS: RESET
--GV: GRAVAR
--PR1: PROCESS_R1
--PR2: PROCESS_R2
--PW: PROCESS_W
--RC: RESET CONTADOR
```

Figura 8 – Tags da máquina de estados.

Então definimos a entidade, baseada em como está no projeto, com todas as 12 entradas e 11 saídas, além do clock. Dentro da arquitetura definimos cada uma dessas tags como um tipo de estado e também definimos um sinal para o estado atual e o próximo.

```
type state_type is (DE, TI, RT, RI, RE, OT, OU, RS, GV, PR1, PR2, PW, RC);
signal y_present , y_next : state_type ;
```

Figura 9 – Definição de estados.

Na implementação de cada estado, utilizamos o comando de *CASE* e dentro de cada um, aplicamos a lógica booleana baseado na tabela de transição de estados, por exemplo temos na figura 10 a implementação do estado de Tela Inicial.

```
when TI =>
  if (B_ON = '0' and PLAY = '0' and MU = '0' and T60 = '0' and T5 = '0' and T2 = '0') then y_next <= TI;
  elsif (B_ON = '0' and T5 = '0' and T2 = '1') then y_next <= RS;
  elsif (B_ON = '0' and T5 = '1' and T2 = '0') then y_next <= DE;
  elsif (B_ON = '0' and PLAY = '0' and MU = '0' and T60 = '1' and T5 = '0' and T2 = '0') then y_next <= RE;
  elsif (B_ON = '0' and PLAY = '0' and MU = '1' and M = '1' and T5 = '0' and T2 = '0') then y_next <= PR1;
  elsif (B_ON = '0' and PLAY = '1' and M = '1' and T5 = '0' and T2 = '0') then y_next <= RC;
  elsif (B_ON = '0' and M = '0' and T60 = '0' and T5 = '0' and T2 = '0') then y_next <= TI;
  elsif (B_ON = '1') THEN y_next <= RE; end if;
```

Figura 10 – Exemplo da implementação do estado *Tela Inicial*

2.3 Display

Na parte do display, primeiro fizemos um código da lógica dos 7 segmentos, para o valor que entrar na entrada I, acionar corretamente o vetor de saída O correspondente aos segmentos.

```
entity D7SEG is
    port (I: in std_logic_vector (3 downto 0);
          O: out std_logic_vector (6 downto 0));
end D7SEG;

architecture logic of D7SEG is
    signal A, B, C, D, NA, NB, NC, ND: std_logic;
begin
    A <= I(3);
    B <= I(2);
    C <= I(1);
    D <= I(0);
    NA <= not A;
    NB <= not B;
    NC <= not C;
    ND <= not D;

    O(0) <= C or A or (NB and ND) or (B and D);
    O(1) <= NB or (NC and ND) or (C and D);
    O(2) <= NC or D or B;
    O(3) <= (NB and ND) or (NB and C) or (C and ND) or (B and NC and D);
    O(4) <= (NB and ND) or (C and ND);
    O(5) <= A or (NC and ND) or (B and NC) or (B and ND);
    O(6) <= A or (NB and C) or (C and ND) or (B and NC);
end logic;
```

Figura 11 – Código dos 7 segmentos do display.

Então no bloco operacional fizemos uma lógica para cada um dos displays receber um sinal do valor que eles devem mostrar na tela.

```
D7S1: decod_bcd_8bit port map(ss_msg_r,s_binbcd);
dsp1<=s_binbcd(3 downto 0);
dsp2<=s_binbcd(7 downto 4);
```

Figura 12 – Displays recebendo o valor.

2.4 Temporizadores

Seguindo o projeto que foi proposto, temos que implementar a lógica de *Reset*, *Desligar* e entrar em repouso, utilizando os temporizadores de 2 segundos, 5 segundos e 60 segundos respectivamente.

2.4.1 2 segundos e 5 segundos.

Para isso, começamos implementando a lógica de 2 segundos e 5 segundos na mesma entidade, tendo em vista que os dois recebem a mesma entrada, que é o botão de liga/desliga.

Nesse código temos uma variável do tipo inteiro que vai ser nosso contador, incrementando a cada borda de subida do *clock* enquanto o botão liga/desliga estiver pressionado. No momento em que o botão não for mais pressionado, o contador irá para zero. Então temos as condições para acionar as flags que são bem triviais, quando o contador for maior que 2 e menor que 5, a flag de 2 segundos irá acionar e quando o contador ficar maior que 5 a flag de 2 segundos vai para zero e a de 5 segundos aciona.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Timers_2_5 is
5  port (
6      clk, b_on      :    in std_logic;
7      timer2s, timer5s:    out std_logic
8  );
9  end Timers_2_5;
10
11 architecture hardware of Timers_2_5 is
12 begin
13 process(clk, b_on)
14 variable counter      :    integer range 0 to 5;
15 begin
16     if (clk = '1' and b_on = '1') then
17         counter := counter + 1;
18     elsif (b_on = '0') then
19         counter := 0;
20
21     end if;
22
23     if (counter > 1 and counter < 4) then
24         timer2s <= '1';
25     else timer2s <= '0';
26     end if;
27
28     if (counter = 4) then
29         timer5s <= '1';
30     else timer5s <= '0';
31     end if;
32
33 end process;
34 end hardware;
```

Figura 13 – Código dos temporizadores 2seg e 5seg.

2.4.2 60 segundos

Como o temporizador de 60 segundos está ligado com a inatividade do usuário, então colocamos como entrada todos os botões. Sendo assim, para essa entidade o a variável de contador vai estar sempre ativa, sendo incrementada em cada borda de subida do *clock* e quando algum botão for apertado, o contador vai voltar a ser 0. Já para a condição da flag, quando o contador chegar a 60, quer dizer que o sistema inativo durante 60 segundos, sendo assim nós acionamos a flag do temporizador.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Timer_60 is
5  port (
6      clk, b_on, b_play, b_next, b_back      :    in std_logic;
7      timer60s :    out std_logic
8  );
9  end Timer_60;
10
11 architecture hardware of Timer_60 is
12 begin
13 process(clk)
14 variable counter      :    integer range 0 to 80;
15
16 begin
17     if (clk = '1' and b_on = '0' and b_play = '0' and b_next = '0' and b_back = '0') then
18         counter := counter + 1;
19     elsif (b_on = '1' or b_play = '1' or b_next = '1' or b_back = '1') then
20         counter := 0;
21
22     end if;
23
24     if (counter > 59 and counter < 61) then
25         timer60s <= '1';
26     else
27         timer60s <= '0';
28     end if;
29
30 end process;
31 end hardware;
```

Figura 14 – Código do temporizador de 60 segundos.

2.5 Contador de Escrita/Leitura

Como dito anteriormente, durante implementação da leitura, percebemos para o usuário ficar percorrendo as mensagens, apenas o sinal do NB não daria para cobrir todas as situações, então decidimos nessa parte também alterar o projeto e colocar, ao invés de NB, o sinal de N e o de B (botão Next e botão Back).

No código temos o contador *COUNTER* incrementando ou decrementando, baseado nos botões N e B. Então convertemos o contador para um vetor de 5 bits e colocamos na saída (qual dos 25 registradores será ser lido).

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity CONT_L is
6  port (CLK,LOAD,CLEAR, N, B: IN STD_LOGIC;
7        CONT_W: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
8        COMP_EQ: OUT STD_LOGIC;
9        SAIDÄ: OUT STD_LOGIC_VECTOR (4 DOWNTO 0));
10 end entity;
11
12 architecture logica of CONT_L is
13
14     SIGNAL COUNT : STD_LOGIC_VECTOR(4 DOWNTO 0);
15     SIGNAL CNT_W : INTEGER RANGE 0 TO 24;
16
17 begin
18
19     PROCESS (CLK,LOAD,CLEAR)
20     VARIABLE COUNTER : INTEGER RANGE 0 TO 24;
21     BEGIN
22         CNT_W <= to_integer(unsigned(CONT_W));
23
24         IF (CLEAR = '1' AND CLK = '1' and CLK'event) THEN
25             COUNTER := 0;
26         ELSIF (CLK = '1' and CLK'event AND LOAD = '1' and N = '0' and B = '0' and (COUNTER < CNT_W)) THEN
27             COUNTER := COUNTER + 1;
28         ELSIF (CLK = '1' and CLK'event AND LOAD = '1' and N = '1' and (COUNTER < CNT_W)) THEN
29             COUNTER := COUNTER + 1;
30         ELSIF (CLK = '1' and CLK'event AND LOAD = '1' and B = '1' and (COUNTER /= 0)) THEN
31             COUNTER := COUNTER - 1;
32         ELSIF (COUNTER = 24) THEN
33             COUNTER := 0;
34         END IF;
35
36         SAIDÄ <= std_logic_vector(to_unsigned(COUNTER, COUNT'LENGTH));
37
38         IF (COUNTER = CNT_W) THEN
39             COMP_EQ <= '1';
40         ELSE
41             COMP_EQ <= '0';
42         END IF;
43
44     END PROCESS;
45 end logica;
```

Figura 15 – Código de leitura.

Já no código de escrita, como não depende dos botões de Next ou Back, ele apenas segue o contador da quantidade atual de mensagens e vai incrementando sempre que uma nova mensagem é gravada. Com a exceção de quando está no máximo, que aí o contador não vai incrementar e nem gravar, só vai sair desse estado quando a máquina entrar no estado de *Reset* e zerar as mensagens.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity CONT_ESC is
6  port (CLK,LOAD,CLEAR: IN STD_LOGIC;
7        SAIDA: OUT STD_LOGIC_VECTOR (4 DOWNTO 0));
8  end entity;
9
10 architecture logica of CONT_ESC is
11
12     SIGNAL COUNT : STD_LOGIC_VECTOR(4 DOWNTO 0);
13
14 begin
15     PROCESS (CLK,LOAD,CLEAR)
16     VARIABLE COUNTER : INTEGER RANGE 0 TO 24;
17     BEGIN
18         IF (CLEAR = '1' AND CLK = '1' and CLK'event) THEN
19             COUNTER := 0;
20         ELSIF (CLK = '1' and CLK'event AND LOAD = '1') THEN
21             COUNTER := COUNTER + 1;
22         ELSIF (COUNTER = 24) THEN
23             COUNTER := 0;
24         END IF;
25         SAIDA <= std_logic_vector(to_unsigned(COUNTER, COUNT'LENGTH));
26     END PROCESS;
27 end logica;

```

Figura 16 – Código de escrita.

2.6 Bloco de Registro

Para implementar a memória da forma como o projeto pedia, precisamos desenvolver 25 blocos de registros, isso é, um contador que vai recebendo as amostras e mandando a informação como um endereçamento para o bloco de memória RAM, por fim um comparador para saber quando o sistema recebeu todas as amostras da mensagem (no projeto ficou definido que cada mensagem sempre receberia 10 amostras, então o comparador está para 'igual a 10'), quando a condição for atendida, vai acionar uma flag de que acabou a mensagem.

2.6.1 Contador das Amostras

Implementamos o contador que vai ser incrementado até 10 enquanto estiver com o *enable* em 1. Por praticidade, já incorporamos o comparador dentro do nosso contador, então na arquitetura, temos uma comparação que quando o contador for igual a 9, acionamos a flag de *block* do bloco de registro. Enquanto isso não ocorre e o contador está incrementando, estamos enviando como saída para o bloco de memória ram o valor do contador, ou seja, o endereço que será gravada/liga a amostra.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity contador10 is
6  port(clk, en, clr:   in std_logic;
7        b_block: out std_logic;
8        o: out std_logic_vector (3 downto 0));
9  end contador10;
10
11 architecture logic of contador10 is
12 begin
13     process(clk, clr)
14     variable counter: integer range 0 to 9;
15     begin
16
17         if(clr = '1' and clk'event and clk = '1') then
18             counter := 0;
19         elsif (counter = 9) then
20             counter := 0;
21         elsif(en='1' and clk'event and clk = '1') then
22             counter := counter + 1;
23         end if;
24
25         if (counter = 9) then
26             b_block <= '1';
27         else b_block <= '0';
28         end if;
29
30         o <= std_logic_vector(to_unsigned(counter, o'length));
31     end process;
32 end logic;
```

Figura 17 – Código do contador de amostras.

2.6.2 Memória RAM

No código podemos ver que temos tanto um *data* de entrada (*datain*) como um *data* de saída (*dataout*), identificando a informação que será escrita ou lida na memória, respectivamente. a entrada *rw* quando está em zero, o bloco entende que será usado para leitura, quando está em um será escrita. Também temos um *en* que é o enable do bloco. Por fim a entrada *addr* é justamente o endereço que vamos receber do contador, onde vamos gravar cada amostra.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity s_ram is
6  port (
7      clk          : in std_logic ;
8      en           : in std_logic ;
9      rw           : in std_logic ;
10     addr          : in std_logic_vector (3 downto 0);
11     datain        : in std_logic_vector (7 downto 0);
12     dataout       : out std_logic_vector (7 downto 0));
13 end s_ram ;
14
15 architecture logica of s_ram is
16
17     type ram is array (0 to 9) of std_logic_vector (7 downto 0);
18     signal memoria : ram ;
19     signal r_addr : std_logic_vector (0 to 3);
20
21 begin
22
23     process ( clk )
24     begin
25         if rising_edge ( clk ) then
26             if (en = '1' and rw = '1') then
27                 memoria ( to_integer ( unsigned ( addr ) ) ) <= datain;
28             elsif (en = '1' and rw = '0') then
29                 --r_addr <= addr ;
30                 dataout <= memoria (to_integer(unsigned(addr)));
31             end if;
32         end if;
33     end process ;
34     --dataout <= memoria (to_integer(unsigned(r_addr)));
35 end logica ;
36
```

Figura 18 – Código da memória RAM.

A arquitetura é bem trivial, tanto a situação de escrita e leitura só serão acionadas quando o *clock* estiver em borda de subida e o *enable* ativo. Então dependendo do valor de *rw* vamos ter a situação de escrita ou de leitura.

2.7 Bloco Operacional

Finalmente temos que unir todas as entradas, saídas e sinais no bloco operacional.

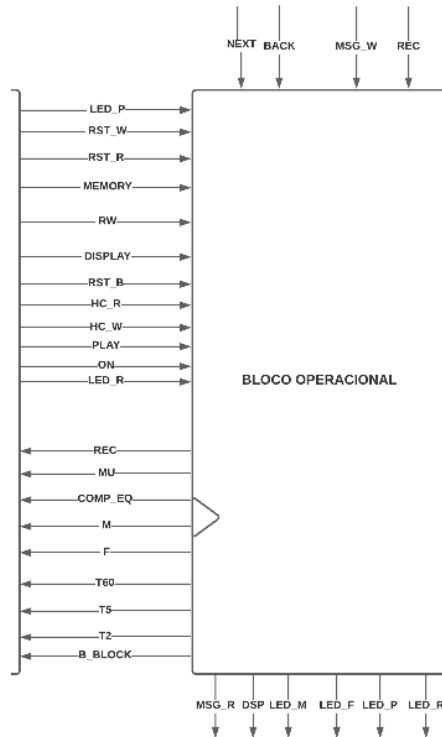


Figura 19 – Projeto do bloco operacional.

Declaramos a entidade do bloco operacional e adicionamos todas as 16 entradas e 17 saídas além do clock.

```
entity bloco_op is
port (clk, rst_w, rst_r, memory, rw, display, rst_b, hc_r, hc_w, b_next, b_back, rec_i, b_on, b_play, led_r_i, led_p_i: in std_logic;
      msg_w: in std_logic_vector (7 downto 0);
      rec_o, mu, comp_eq, m, f, t60, t5, t2, b_block, led_m, led_f, led_p, led_r: out std_logic;
      dsp: out std_logic_vector (4 downto 0);
      dsp1, dsp2: out std_logic_vector(3 downto 0);
      msg_r: out std_logic_vector(7 downto 0)
);
end entity;
```

Figura 20 – Entidade do bloco operacional.

Então adicionamos os componentes que foram implementados anteriormente: Bloco de registro, temporizador de 60 segundos, temporizador de 2 e 5 segundos, contador de escrita, contador de leitura e o decodificador BCD para o display.

```

CASE (mux_rw) is
  when "00000" => dmux(0) <= '1';
  when "00001" => dmux(1) <= '1';
  when "00010" => dmux(2) <= '1';
  when "00011" => dmux(3) <= '1';
  when "00100" => dmux(4) <= '1';
  when "00101" => dmux(5) <= '1';
  when "00110" => dmux(6) <= '1';
  when "00111" => dmux(7) <= '1';
  when "01000" => dmux(8) <= '1';
  when "01001" => dmux(9) <= '1';
  when "01010" => dmux(10) <= '1';
  when "01011" => dmux(11) <= '1';
  when "01100" => dmux(12) <= '1';

```

(a) Case do *dmux*.

```

CASE (mux_rw) is
  when "00000" => b_block <= sb_block(0);
  when "00001" => b_block <= sb_block(1);
  when "00010" => b_block <= sb_block(2);
  when "00011" => b_block <= sb_block(3);
  when "00100" => b_block <= sb_block(4);
  when "00101" => b_block <= sb_block(5);
  when "00110" => b_block <= sb_block(6);
  when "00111" => b_block <= sb_block(7);
  when "01000" => b_block <= sb_block(8);
  when "01001" => b_block <= sb_block(9);
  when "01010" => b_block <= sb_block(10);
  when "01011" => b_block <= sb_block(11);
  when "01100" => b_block <= sb_block(12);
  when "01101" => b_block <= sb_block(13);
  when "01110" => b_block <= sb_block(14);
  when "01111" => b_block <= sb_block(15);

```

(b) Case do *block*.

Figura 21 – 25 Blocos de registro.

Seguindo o projeto, tivemos que implementar os 25 blocos de registros na arquitetura. Então criamos um sinal *dmux* que é um vetor de 25 bits (Figura 21a) e então criamos um case para que, dependendo do valor recebido pelo MUX do RW, vamos setar um bit do *dmux* para 1, que representa o bloco de registrador que vai ser escrito ou lido. Da mesma forma também criamos um case para acionar o sinal de *block* de cada bloco de registro, como podemos ver na figura 21b e também tivemos que implementar os 25 enables.

Por fim, criamos o port map de cada bloco de registro, usando os sinais e variáveis que implementamos anteriormente.

```

BR1: blocor port map (enable(0), clk, rst_b, memory, rw, msg_w, sb_block(0), s_msg_r);
BR2: blocor port map (enable(1), clk, rst_b, memory, rw, msg_w, sb_block(1), s_msg_r);
BR3: blocor port map (enable(2), clk, rst_b, memory, rw, msg_w, sb_block(2), s_msg_r);
BR4: blocor port map (enable(3), clk, rst_b, memory, rw, msg_w, sb_block(3), s_msg_r);
BR5: blocor port map (enable(4), clk, rst_b, memory, rw, msg_w, sb_block(4), s_msg_r);
BR6: blocor port map (enable(5), clk, rst_b, memory, rw, msg_w, sb_block(5), s_msg_r);
BR7: blocor port map (enable(6), clk, rst_b, memory, rw, msg_w, sb_block(6), s_msg_r);
BR8: blocor port map (enable(7), clk, rst_b, memory, rw, msg_w, sb_block(7), s_msg_r);
BR9: blocor port map (enable(8), clk, rst_b, memory, rw, msg_w, sb_block(8), s_msg_r);

```

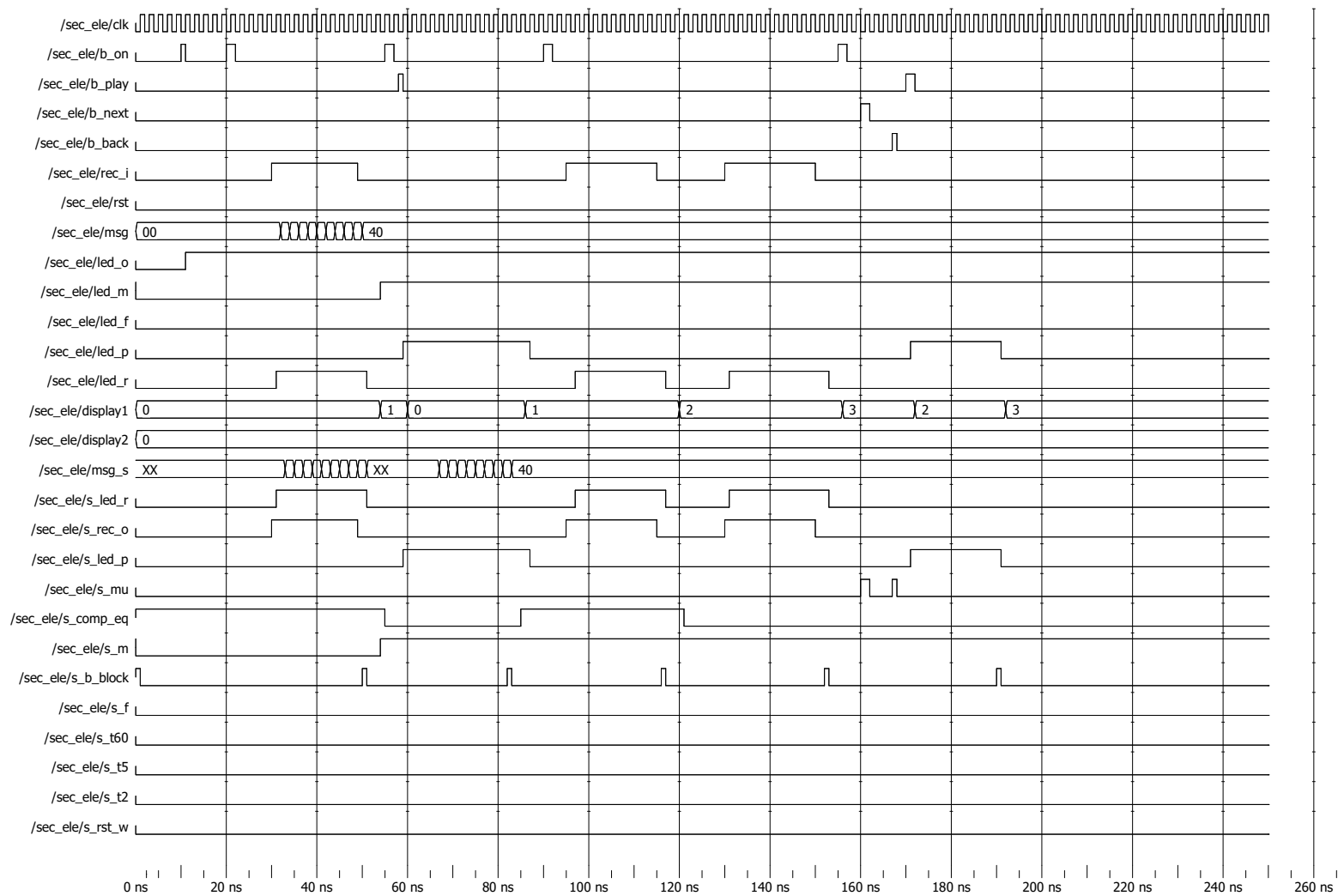
Figura 22 – Port map dos blocos de registro.

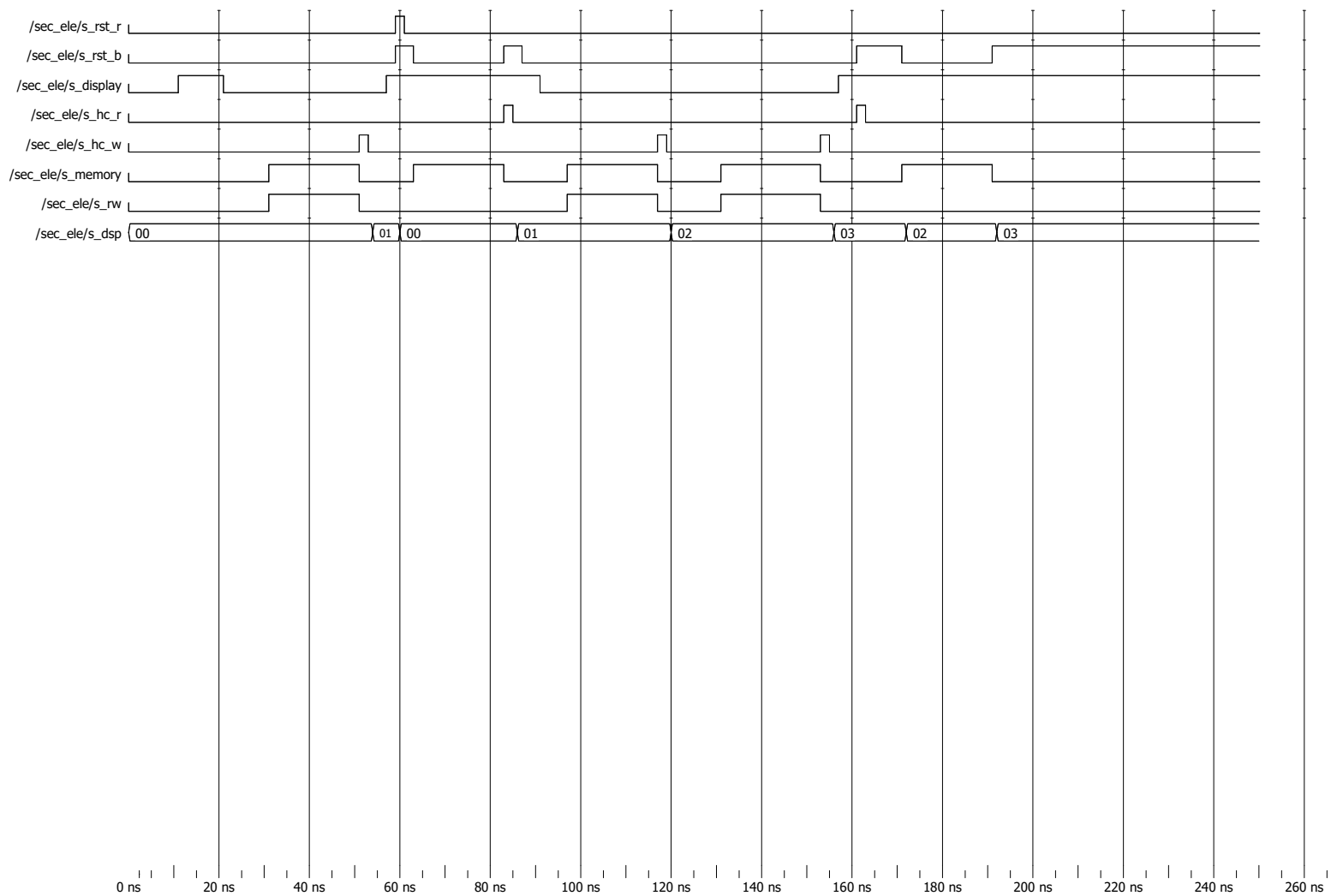
3 Resultados

Durante a simulação de todo o projeto, conseguimos observar que colocando valores nos botões de entrada, obtinhamos corretamente os valores de saída tanto do display quanto dos LEDs, então a parte do bloco de controle está funcionando corretamente.

Já a parte do bloco operacional, a gravação de mensagens está funcionando perfeitamente. Porém a leitura das mensagens estava aparecendo um bug que quando o sinal deveria ir para 1, ele ficava como *undentified*.

Na simulação que vamos apresentar a seguir, tentamos cobrir a maior parte das situações propostas no problema. Todas as trocas de estados e as funcionalidades do bloco de registro.





4 Conclusão

No decorrer da implementação, foi necessário voltar várias vezes para o projeto e fazer algumas modificações e ajustes. Porém conseguimos implementar tudo o que foi proposto no projeto.

No bloco de comando, conseguimos ver durante as simulações que estava funcionando perfeitamente, todas as transições de estados estavam funcionais.

Já no bloco operacional, tivemos muitos problemas com a implementação de 25 blocos de registro. Conseguimos gravar mensagens nos blocos com algumas dificuldades, porém a parte de leitura realmente não estava funcional nas simulações. Como dito nos resultados, os sinais estavam indo para *undefined*.

Referências

VAHID, F. *Sistemas Digitais: Projeto, Otimização e HDLs*. [S.l.]: Artmed Bookman, 2008.

ANEXO A - Relato Semanal

Líder: Lucas Batista da Fonseca

A.1 Equipe

Função do integrante	Discente
Redator	Gabriel Cavalheiro Francisco
Debatedor	Isaac de Lyra Junior
Videomaker	Allysson de Andrade Silva
Auxiliar	-

Fonte: Elaborado pelos autores.

A.2 Defina o problema

O problema consiste em projetar uma secretária eletrônica, a qual deve guardar 25 mensagens que possui no 10 amostras e cada amostra tem tamanho de 8 bits, além disso a máquina deve reproduzir a mensagem gravada, enviando para o conversor D/A amostra por amostra, a quantidade de mensagens deve aparecer na no display de dois dígitos quando estiver ligado e na tela inicial, além disso deve aparecer a mensagem que está sendo reproduzida quando estiver reproduzindo a mesma, ela possui também 4 botões que respectivamente liga/desliga/repousa, play, mensagem seguinte e mensagem anterior. possui também 6 leds (O: em funcionamento; R: gravando; P: reproduzindo; F:cheio; M: tem mensagem).

A.3 Registro do brainstorming

Foram realizadas reuniões diárias para delimitação do que deveria ser feito e para resolução de problemas.

A primeira reunião foi na terça-feira logo após a aula, nela foram definidas as funções de cada membro e foi criado uma tabela para marcar os horários das reuniões seguintes.

LUCAS							LEGENDA
Horarios	Quarta	Quinta	Sexta	Sabado	Domingo	SEGUNDA	VAGO
M12							Ocupado
M34							Reunioes
M56							Caso necessario
T12							
T34							
T56							
N12							
N34							
ISAAC, O Lindo							LEGENDA
Horarios	Quarta	Quinta	Sexta	Sabado	Domingo	SEGUNDA	VAGO
M12							Ocupado
M34							Reunioes
M56							Caso necessario
T12							
T34							
T56							
N12							
N34							
Allysson							LEGENDA
Horarios	Quarta	Quinta	Sexta	Sabado	Domingo	SEGUNDA	VAGO
M12							Ocupado
M34							Reunioes
M56							Caso necessario
T12							
T34							
T56							
N12							
N34							
GABRIEL							LEGENDA
Horarios	Quarta	Quinta	Sexta	Sabado	Domingo	SEGUNDA	VAGO
M12							Ocupado
M34							Reunioes
M56							Caso necessario
T12							
T34							
T56							
N12							
N34							

No dia seguinte , decidimos começar a implementar os temporizadores, os membros trouxeram suas sugestões de circuito e as primeiras implementações dos temporizadores foram realizadas. Na reunião seguinte foram identificadas algumas inconsistências na MDE e consequentemente na tabela verdade, primeiras correções foram feitas. O restante da semana foi dedicado à correção de erros de projeto e implementação. Por fim, no último dia houve a correção de erros da máquina, entretanto a máquina não ficou 100% finalizada.

A.4 Pontos-chaves

O ponto-chave é entender como a memória ram funciona, suas entradas e saídas.

Entender o funcionamento de um conversor A/D e D/A.

Entender que nesse projeto foram utilizadas 25 memórias ram com capacidade de 10 amostras de 8 bits.

Entender a máquina de estado e o bloco operacional.

A.5 Questões de pesquisa

Os tópicos mais relevantes da pesquisa foram: Memória RAM; Conversor A/D e D/A; projeto RTL; bloco de controle; bloco operacional; máquina de alto e baixo nível.

A.6 Planejamento da pesquisa

A idealização foi que no primeiro dia todos estivessem a par de todo o projeto a ser implementado. Após isso foi planejado implementar as partes do bloco operacional que não possuem relação com a máquina de estado, no terceiro dia implementar a máquina de estado e o bloco operacional, no quarto dia, implementar a RTL, e os últimos dias estarem livres para escrever o relatório e correção do código se necessário.

ANEXO A – Código VHDL

```
1
library ieee;
3 use ieee.std_logic_1164.all;

5 entity Timers_2_5 is
port (
7   clk, b_on : in std_logic;
   timer2s, timer5s: out std_logic
9 );
end Timers_2_5;

11
architecture hardware of Timers_2_5 is
13 begin
   process(clk, b_on)
15 variable counter : integer range 0 to 5;
begin
17   if (clk = '1' and b_on = '1') then
       counter := counter + 1;
19   elsif (b_on = '0') then
       counter := 0;

21
       end if;

23
       if (counter > 1 and counter < 4) then
25         timer2s <= '1';
       else timer2s <= '0';
27   end if;

29   if (counter = 4) then
       timer5s <= '1';
31   else timer5s <= '0';
       end if;

33

35 end process;
end hardware;

37


---


39
library ieee;
41 use ieee.std_logic_1164.all;

43 entity Timer_60 is
```

```

45 port (
    clk, b_on, b_play, b_next, b_back    : in std_logic;
    timer60s : out std_logic
47 );
end Timer _60;

49
architecture hardware of Timer _60 is
51 begin
    process(clk)
53 variable counter : integer range 0 to 80;

55 begin
    if (clk = '1' and b_on = '0' and b_play = '0' and b_next = '0' and b
        _back = '0') then
57         counter := counter + 1;
    elsif (b_on = '1' or b_play = '1' or b_next = '1' or b_back = '1') then
59         counter := 0;

61     end if;

63     if (counter > 59 and counter < 61) then
        timer60s <= '1';
65     else
        timer60s <= '0';
67     end if;

69 end process;
end hardware;

71


---


73
library ieee;
75 use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
77
entity s_ram is
79 port (
    clk      : in std_logic ;
81 en       : in std_logic ;
    rw       : in std_logic ;
83 addr     : in std_logic_vector (3 downto 0);
    datain   : in std_logic_vector (7 downto 0);
85 dataout   : out std_logic_vector (7 downto 0));
end s_ram ;

87
architecture logica of s_ram is
89

```

```

type ram is array (0 to 9) of std_logic_vector (7 downto 0);
191 signal memoria : ram ;
signal r_addr : std_logic_vector (0 to 3);
193
begin
195
process ( clk )
197   begin
      if rising_edge ( clk ) then
199         if (en = '1' and rw = '1') then
            memoria ( to_integer ( unsigned ( addr ))) <= datain;
201         elsif (en = '1' and rw = '0') then
            --r_addr <= addr ;
203             dataout <= memoria (to_integer(unsigned(addr)));
            end if;
205         end if;
      end process ;
207       --dataout <= memoria (to_integer(unsigned(r_addr)));
end logica ;
209


---


211
library ieee ;
213 use ieee.std_logic_1164.all;

215 --DE: DESLIGADO
--TI: TELA INICIAL
217 --RT: REPRODUCAO TOTAL
--RI: REPRODUCAO INDIVIDUAL
219 --RE: REPOUSO
--OT: OUVIR TUDO
221 --OU: OUVIR UM
--RS: RESET
223 --GV: GRAVAR
--PR1: PROCESS_R1
225 --PR2: PROCESS_R2
--PW: PROCESS_W
227 --RC: RESET CONTADOR

229 entity mde_b is
      port ( CLK , RST , B_ON, PLAY, MU, COMP_EQ, REC, M, B_BLOCK, F, T60, T5
            , T2: in std_logic ;
231             LED_O, LED_R, LED_P, RST_W, RST_R, RST_B, DISPLAY, HC_R, HC_W,
            MEMORY, RW: out std_logic );
end mde_b ;
233
architecture logica of mde_b is

```

```

135     type state_type is (DE, TI, RT, RI, RE, OT, OU, RS, GV, PR1, PR2, PW,
136                          RC);
137     signal y_present , y_next : state_type ;
138
139 begin
140     process (B_ON, PLAY, MU, COMP_EQ, REC, M, B_BLOCK, F, T60, T5, T2,
141              y_present )
142     begin
143         case y_present is
144         when DE =>
145             if B_ON = '0' then y_next <= DE;
146             else y_next <= TI; end if;
147         when TI =>
148             if (B_ON = '0' and PLAY = '0' and MU = '0' and T60 = '0' and T5
149                = '0' and T2 = '0') then y_next <= TI;
150             elsif (B_ON = '0' and T5 = '0' and T2 = '1') then y_next <= RS;
151             elsif (B_ON = '0' and T5 = '1' and T2 = '0') then y_next <= DE;
152             elsif (B_ON = '0' and PLAY = '0' and MU = '0' and T60 = '1' and
153                    T5 = '0' and T2 = '0') then y_next <= RE;
154             elsif (B_ON = '0' and PLAY = '0' and MU = '1' and M = '1' and
155                    T5 = '0' and T2 = '0') then y_next <= PR1;
156             elsif (B_ON = '0' and PLAY = '1' and M = '1' and T5 = '0' and
157                    T2 = '0') then y_next <= RC;
158             elsif (B_ON = '0' and M = '0' and T60 = '0' and T5 = '0' and T2
159                    = '0') then y_next <= TI;
160             elsif (B_ON = '1') THEN y_next <= RE; end if;
161         when RT =>
162             if (B_ON = '0' and COMP_EQ = '0') then y_next <= OT;
163             elsif (B_ON = '0' and COMP_EQ = '1') then y_next <= TI;
164             elsif (B_ON = '1') then y_next <= RE; end if;
165         when RI =>
166             if (B_ON = '0' and PLAY = '0' and MU = '0') then y_next <= RI;
167             elsif (B_ON = '0' and PLAY = '0' and MU = '1') then y_next <=
168                 PR1 ;
169             elsif (B_ON = '0' and PLAY = '1') then y_next <= OU ;
170             elsif (B_ON = '1') then y_next <= RE; end if;
171         when RE =>
172             if (B_ON = '0' and REC = '0' and T5 = '0') then y_next <= RE;
173             elsif (B_ON = '0' and REC = '0' and T5 = '1') then y_next <= DE
174                 ;
175             elsif (B_ON = '0' and REC = '1' and F = '0') then y_next <= GV;
176             elsif (B_ON = '0' and REC = '1' and F = '1') then y_next <= RE;
177             elsif (B_ON = '1' and REC = '0' and T5 = '0') then y_next <= TI
178                 ;
179             elsif (B_ON = '1' and REC = '1' and F = '0' and T5 = '0') then
180                 y_next <= GV;
181             elsif (B_ON = '1' and REC = '1' and F = '1' and T5 = '0') then
182                 y_next <= TI; end if;

```



```

171         when OT =>
            if B_BLOCK = '0' then y_next <= OT;
            else y_next <= PR2; end if;
173         when OU =>
            if B_BLOCK = '0' then y_next <= OU;
175             else y_next <= RI; end if;
            when RS =>
177                 y_next <= TI;
            when GV =>
179                 if B_BLOCK = '0' then y_next <= GV;
                    else y_next <= PW; end if;
181             when PR1 =>
                    y_next <= RI;
183             when PR2 =>
                    y_next <= RT;
185             when PW =>
                    y_next <= RE;
187             when RC =>
                    y_next <= RT;
189         end case;
    end process;

191
193 process (CLK , RST )
    begin
        if RST = '1' then
195             y_present <= DE ;
            elsif ( CLK'event and CLK = '1') then
197                 y_present <= y_next;
            end if;
199     end process ;

201 LED_O <= '0' when y_present = DE else '1';
    LED_R <= '1' when (y_present = GV or y_present = PW ) else '0';
203 LED_P <= '1' when (y_present = RT or y_present = OT or y_present = OU
        or y_present = PR1 or y_present = PR2 or y_present = RC) else '0';
    RST_W <= '1' when y_present = RS else '0';
205 RST_R <= '1' when (y_present = RS or y_present = RC) else '0';
    RST_B <= '1' when (y_present = RT or y_present = RI or y_present = RS
        or y_present = PR1 or y_present = PR2 or y_present = RC) else '0';
207 DISPLAY <= '0' when (y_present = DE or y_present = RE or y_present = GV
        or y_present = PW) else '1';
    HC_R <= '1' when (y_present = PR1 or y_present = PR2) else '0';
209 HC W <= '1' when y_present = PW else '0';
    MEMORY <= '1' when (y_present = OU or y_present = OT or y_present = GV)
        else '0';
211 RW <= '1' when y_present = GV else '0';

```

```

213 end logica ;

215


---


217 library ieee;
218 use ieee.std_logic_1164.all;
219 use ieee.numeric_std.all;

221 entity contador10 is
222     port (clk, en, clr: in std_logic;
223           b_block: out std_logic;
224           o: out std_logic_vector (3 downto 0));
225 end contador10;

227 architecture logic of contador10 is
228     begin
229         process (clk, en, clr)
230             variable counter: integer range 0 to 9;
231             begin
232
233                 if (clr = '1' and clk'event and clk = '1') then
234                     counter := 0;
235             elsif (counter = 9) then
236                 counter := 0;
237             elsif (en='1' and clk'event and clk = '1') then
238                 counter := counter + 1;
239             end if;

240
241                 if (counter = 9) then
242                     b_block <= '1';
243                 else b_block <= '0';
244                 end if;

245
246                 o <= std_logic_vector(to_unsigned(counter, o'length));
247             end process;
248         end logic;
249


---


251
252 library ieee;
253 use ieee.std_logic_1164.all;
254 use ieee.numeric_std.all;

255
256 entity CONT_ESC is
257     port (CLK,LOAD,CLEAR: IN STD_LOGIC;
258           SAIDA: OUT STD_LOGIC_VECTOR (4 DOWNT0 0));
259 end entity;

```

```

261 architecture logica of CONT_ESC is
263     SIGNAL COUNT : STD_LOGIC_VECTOR(4 DOWNT0 0);
265 begin
266     PROCESS (CLK,LOAD,CLEAR)
267     VARIABLE COUNTER : INTEGER RANGE 0 TO 24;
268     BEGIN
269         IF (CLEAR = '1' AND CLK = '1' and CLK'event) THEN
270             COUNTER := 0;
271         ELSIF (CLK = '1' and CLK'event AND LOAD = '1') THEN
272             COUNTER := COUNTER + 1;
273         ELSIF (COUNTER = 24) THEN
274             COUNTER := 0;
275         END IF;
276         SAIDA <= std_logic_vector(to_unsigned(COUNTER, COUNT'LENGTH));
277     END PROCESS;
278 end logica;
279
280
281
282 library ieee;
283 use ieee.std_logic_1164.all;
284 use ieee.numeric_std.all;
285
286 entity CONT_L is
287     port (CLK,LOAD,CLEAR, N, B: IN STD_LOGIC;
288           CONT_W: IN STD_LOGIC_VECTOR (4 DOWNT0 0);
289           COMP_EQ: OUT STD_LOGIC;
290           SAIDA: OUT STD_LOGIC_VECTOR (4 DOWNT0 0));
291 end entity;
292
293 architecture logica of CONT_L is
295     SIGNAL COUNT : STD_LOGIC_VECTOR(4 DOWNT0 0);
296     SIGNAL CNT_W : INTEGER RANGE 0 TO 24;
297
298 begin
299
300     PROCESS (CLK,LOAD,CLEAR)
301     VARIABLE COUNTER : INTEGER RANGE 0 TO 24;
302     BEGIN
303         CNT_W <= to_integer(unsigned(CONT_W));
304
305         IF (CLEAR = '1' AND CLK = '1' and CLK'event) THEN
306             COUNTER := 0;

```

```

307     ELSIF (CLK = '1' and CLK'event AND LOAD = '1' and N = '0' and B = '0'
           and (COUNTER < CNT_W)) THEN
           COUNTER := COUNTER + 1;
309     ELSIF (CLK = '1' and CLK'event AND LOAD = '1' and N = '1' and (COUNTER <
           CNT_W)) THEN
           COUNTER := COUNTER + 1;
311     ELSIF (CLK = '1' and CLK'event AND LOAD = '1' and B = '1' and (COUNTER
           /= 0)) THEN
           COUNTER := COUNTER - 1;
313     ELSIF (COUNTER = 24) THEN
           COUNTER := 0;
315     END IF;

317     SAIDA <= std_logic_vector(to_unsigned(COUNTER, COUNT'LENGTH));

319     IF (COUNTER = CNT_W) THEN
           COMP_EQ <= '1';
321     ELSE
           COMP_EQ <= '0';
323     END IF;

325     END PROCESS;
end logica;
327

329
331 library ieee;
333 use ieee.std_logic_1164.all;
335 use ieee.numeric_std.all;
337
339 — DISPLAY 7 SEGMENTOS —
341
343 entity D7SEG is
345     port (I: in std_logic_vector (3 downto 0);
           O: out std_logic_vector (6 downto 0));
347 end D7SEG;

349 architecture logic of D7SEG is
           signal A, B, C, D, NA, NB, NC, ND: std_logic;
345 begin
           A <= I(3);
347           B <= I(2);
           C <= I(1);
349           D <= I(0);

```

```

351     NA <= not A;
      NB <= not B;
353     NC <= not C;
      ND <= not D;
355
      O(0) <= C or A or (NB and ND) or (B and D);
357
      O(1) <= NB or (NC and ND) or (C and D);
359
      O(2) <= NC or D or B;
361
      O(3) <= (NB and ND) or (NB and C) or (C and ND) or (B and NC and D);
363
      O(4) <= (NB and ND) or (C and ND);
365
      O(5) <= A or (NC and ND) or (B and NC) or (B and ND);
367
      O(6) <= A or (NB and C) or (C and ND) or (B and NC);
369
end logic;
371
library ieee;
373 use ieee.std_logic_1164.all;
      use ieee.numeric_std.all;
375 -----
      — BLOCO Bin-BCD —
377 -----
379 entity bloco_bin_bcd is
      port (A: in std_logic_vector (3 downto 0);
381          S: out std_logic_vector(3 downto 0));
end bloco_bin_bcd;
383
architecture logic of bloco_bin_bcd is
385 begin
387     S(0) <= (A(3) and (not A(0))) or ((not A(3)) and (not A(2)) and A(0)) or
              (A(2) and A(1) and (not A(0)));
389
      S(1) <= ((not A(2)) and A(1)) or (A(1) and A(0)) or (A(3) and (not A(0))
              );
391
      S(2) <= (A(3) and A(0)) or (A(2) and (not A(1)) and (not A(0)));
393
      S(3) <= A(3) or (A(2) and A(0)) or (A(2) and A(1));
395 end logic;

```

```

397 library ieee;
    use ieee.std_logic_1164.all;
399 use ieee.numeric_std.all;

=====
401 — DECODIFICADOR Bin-BCD (8 std_logics) —
=====

403
    entity decod_bcd_8bit is
405     port(bin: in std_logic_vector(5 downto 0);
          bcd: out std_logic_vector (7 downto 0));
407 end decod_bcd_8bit;

409 architecture logic of decod_bcd_8bit is

411 component bloco_bin_bcd
    port(A: in std_logic_vector(3 downto 0);
413         S: out std_logic_vector(3 downto 0));
end component;

415
    signal B1_A, B1_S, B2_A, B2_S, B3_A, B3_S, B4_A, B4_S,
417         B5_A, B5_S, B6_A, B6_S, B7_A, B7_S: std_logic_vector(3 downto 0);

419 begin
    B1_A(3) <= '0';
421    B1_A(2) <= '0';
    B1_A(1) <= '0';
423    B1_A(0) <= bin(5);

425    B1: bloco_bin_bcd port map(B1_A, B1_S);

427    B2_A(3) <= B1_S(2);
    B2_A(2) <= B1_S(1);
429    B2_A(1) <= B1_S(0);
    B2_A(0) <= bin(4);

431
    B2: bloco_bin_bcd port map(B2_A, B2_S);

433
    B3_A(3) <= B2_S(2);
435    B3_A(2) <= B2_S(1);
    B3_A(1) <= B2_S(0);
437    B3_A(0) <= bin(3);

439    B3: bloco_bin_bcd port map(B3_A, B3_S);

441
    B4_A(3) <= B3_S(2);
    B4_A(2) <= B3_S(1);

```

```

443 B4_A(1) <= B3_S(0);
    B4_A(0) <= bin(2);
445
    B4: bloco_bin_bcd port map(B4_A, B4_S);
447
    B5_A(3) <= '0';
449 B5_A(2) <= B1_S(3);
    B5_A(1) <= B2_S(3);
451 B5_A(0) <= B3_S(3);

453 B5: bloco_bin_bcd port map(B5_A, B5_S);

455 B6_A(3) <= B4_S(2);
    B6_A(2) <= B4_S(1);
457 B6_A(1) <= B4_S(0);
    B6_A(0) <= bin(1);
459
    B6: bloco_bin_bcd port map(B6_A, B6_S);
461
    B7_A(3) <= B5_S(2);
463 B7_A(2) <= B5_S(1);
    B7_A(1) <= B5_S(0);
465 B7_A(0) <= B4_S(3);

467 B7: bloco_bin_bcd port map(B7_A, B7_S);

469
    bcd(7) <= B7_S(2);
471 bcd(6) <= B7_S(1);
    bcd(5) <= B7_S(0);
473 bcd(4) <= B6_S(3);
    bcd(3) <= B6_S(2);
475 bcd(2) <= B6_S(1);
    bcd(1) <= B6_S(0);
477 bcd(0) <= bin(0);

479 end logic;

481


---


483 use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
485
    entity bloco_op is
487 port (clk, rst_w, rst_r, memory, rw, display, rst_b, hc_r, hc_w, b_next,
        b_back, rec_i, b_on, b_play, led_r_i, led_p_i: in std_logic;
        msg_w: in std_logic_vector (7 downto 0);

```

```

489     rec_o, mu, comp_eq, m, f, t60, t5, t2, b_block, led_m, led_f, led_p,
        led_r: out std_logic;
    dsp: out std_logic_vector (4 downto 0);
491     dsp1,dsp2: out std_logic_vector(3 downto 0);
    msg_r: out std_logic_vector(7 downto 0)
493 );
end entity;
495
architecture ckt of bloco_op is
497     component blocor is
        port (
499         en, clk, rst_b, memory, rw: in std_logic;
        data_i: in std_logic_vector (7 downto 0);
501         rec_block: out std_logic;
        data_o: out std_logic_vector(7 downto 0)
503         );
    end component;
505     component Timer_60 is
        port (
507         clk, b_on, b_play, b_next, b_back : in std_logic;
        timer60s : out std_logic
509         );
    end component;
511
    component Timers_2_5 is
513         port (
            clk, b_on : in std_logic;
515             timer2s, timer5s: out std_logic
        );
517     end component;
    component CONT_ESC is
519         port (
            CLK,LOAD,CLEAR: IN STD_LOGIC;
521             SAIDA: OUT STD_LOGIC_VECTOR (4 DOWNTIO 0)
        );
523     end component;
    component decod_bcd_8bit is
525         port(bin: in STD_LOGIC_VECTOR(5 downto 0);
            bcd: out STD_LOGIC_VECTOR (7 downto 0)
527         );
    end component;
529     component CONT_L is
        port (CLK,LOAD,CLEAR, N, B: IN STD_LOGIC;
531             CONT_W: IN STD_LOGIC_VECTOR (4 DOWNTIO 0);
            COMP_EQ: OUT STD_LOGIC;
533             SAIDA: OUT STD_LOGIC_VECTOR (4 DOWNTIO 0));
    end component;

```



```

535 signal cont_w, cont_r, mux_rw: std_logic_vector(4 downto 0);
537 signal dmux, sb_block : std_logic_vector (24 downto 0);
signal ss_msg_r : std_logic_vector (5 downto 0);
539 signal s_binbcd, s_msg_r : std_logic_vector (7 downto 0);
signal enable: std_logic_vector (24 downto 0);

541 begin
543 P1: CONT_ESC port map( clk , hc_w, rst_w, cont_w );
P2: CONT_L port map( clk , hc_r, rst_r , b_next , b_back , cont_w , comp_eq , cont_r );

545 PROCESS( clk )

547 BEGIN
549 IF (rw = '1') THEN
mux_rw <= cont_w;
551 ELSE
mux_rw <= cont_r;
553 END IF;

555 IF (led_p_i = '1') THEN
dsp <= cont_r;
557 ELSE
dsp <= cont_w;
559 END IF;

561 CASE (mux_rw) is
when "00000" => dmux(0) <= '1';
563 when "00001" => dmux(1) <= '1';
when "00010" => dmux(2) <= '1';
565 when "00011" => dmux(3) <= '1';
when "00100" => dmux(4) <= '1';
567 when "00101" => dmux(5) <= '1';
when "00110" => dmux(6) <= '1';
569 when "00111" => dmux(7) <= '1';
when "01000" => dmux(8) <= '1';
571 when "01001" => dmux(9) <= '1';
when "01010" => dmux(10) <= '1';
573 when "01011" => dmux(11) <= '1';
when "01100" => dmux(12) <= '1';
575 when "01101" => dmux(13) <= '1';
when "01110" => dmux(14) <= '1';
577 when "01111" => dmux(15) <= '1';
when "10000" => dmux(16) <= '1';
579 when "10001" => dmux(17) <= '1';
when "10010" => dmux(18) <= '1';
581 when "10011" => dmux(19) <= '1';

```

```

583     when "10100" => dmux(20) <= '1';
    when "10101" => dmux(21) <= '1';
    when "10110" => dmux(22) <= '1';
585     when "10111" => dmux(23) <= '1';
    when "11000" => dmux(24) <= '1';
587     when others => dmux <= "XXXXXXXXXXXXXXXXXXXXXXXXX";
END CASE;

589
CASE (mux_rw) is
591     when "00000" => b_block <= sb_block(0);
    when "00001" => b_block <= sb_block(1);
593     when "00010" => b_block <= sb_block(2);
    when "00011" => b_block <= sb_block(3);
595     when "00100" => b_block <= sb_block(4);
    when "00101" => b_block <= sb_block(5);
597     when "00110" => b_block <= sb_block(6);
    when "00111" => b_block <= sb_block(7);
599     when "01000" => b_block <= sb_block(8);
    when "01001" => b_block <= sb_block(9);
601     when "01010" => b_block <= sb_block(10);
    when "01011" => b_block <= sb_block(11);
603     when "01100" => b_block <= sb_block(12);
    when "01101" => b_block <= sb_block(13);
605     when "01110" => b_block <= sb_block(14);
    when "01111" => b_block <= sb_block(15);
607     when "10000" => b_block <= sb_block(16);
    when "10001" => b_block <= sb_block(17);
609     when "10010" => b_block <= sb_block(18);
    when "10011" => b_block <= sb_block(19);
611     when "10100" => b_block <= sb_block(20);
    when "10101" => b_block <= sb_block(21);
613     when "10110" => b_block <= sb_block(22);
    when "10111" => b_block <= sb_block(23);
615     when "11000" => b_block <= sb_block(24);
    when others => b_block <= 'X';
617 END CASE;

619 IF (cont_w = "11000") then
    f <= '1';
621     led_f <= '1';
ELSE
623     f <= '0';
    led_f <= '0';
625 END IF;
IF (cont_w /= "00000") THEN
627     m <= '1';
    led_m <= '1';

```

```

629     ELSE
        m <= '0';
631     led_m <= '0';
    END IF;
633     IF (b_next='1' or b_back='1') THEN
        mu <= '1';
635     ELSE
        mu <= '0';
637     END IF;

639 END PROCESS;

641 enable(0) <= (dmux(0) and (led_p_i or led_r_i));
enable(1) <= (dmux(1) and (led_p_i or led_r_i));
643 enable(2) <= (dmux(2) and (led_p_i or led_r_i));
enable(3) <= (dmux(3) and (led_p_i or led_r_i));
645 enable(4) <= (dmux(4) and (led_p_i or led_r_i)) ;
enable(5) <= (dmux(5) and (led_p_i or led_r_i)) ;
647 enable(6) <= (dmux(6) and (led_p_i or led_r_i)) ;
enable(7) <= (dmux(7) and (led_p_i or led_r_i)) ;
649 enable(8) <= (dmux(8) and (led_p_i or led_r_i)) ;
enable(9) <= (dmux(9) and (led_p_i or led_r_i)) ;
651 enable(10) <= (dmux(10) and (led_p_i or led_r_i)) ;
enable(11) <= (dmux(11) and (led_p_i or led_r_i)) ;
653 enable(12) <= (dmux(12) and (led_p_i or led_r_i));
enable(13) <= (dmux(13) and (led_p_i or led_r_i)) ;
655 enable(14) <= (dmux(14) and (led_p_i or led_r_i)) ;
enable(15) <= (dmux(15) and (led_p_i or led_r_i)) ;
657 enable(16) <= (dmux(16) and (led_p_i or led_r_i));
enable(17) <= (dmux(17) and (led_p_i or led_r_i)) ;
659 enable(18) <= (dmux(18) and (led_p_i or led_r_i)) ;
enable(19) <= (dmux(19) and (led_p_i or led_r_i)) ;
661 enable(20) <= (dmux(20) and (led_p_i or led_r_i)) ;
enable(21) <= (dmux(21) and (led_p_i or led_r_i)) ;
663 enable(22) <= (dmux(22) and (led_p_i or led_r_i)) ;
enable(23) <= (dmux(23) and (led_p_i or led_r_i)) ;
665 enable(24) <= (dmux(24) and (led_p_i or led_r_i)) ;

667

669 BR1: blocor port map (enable(0), clk, rst_b, memory, rw, msg_w, sb_block(0),
    s_msg_r);
BR2: blocor port map (enable(1), clk, rst_b, memory, rw, msg_w, sb_block(1),
    s_msg_r);
671 BR3: blocor port map (enable(2), clk, rst_b, memory, rw, msg_w, sb_block(2),
    s_msg_r);

```

```

BR4: blocor port map ( enable(3) , clk , rst_b , memory , rw , msg_w , sb_block(3) ,
    s_msg_r );
673 BR5: blocor port map ( enable(4) , clk , rst_b , memory , rw , msg_w , sb_block(4) ,
    s_msg_r );
BR6: blocor port map ( enable(5) , clk , rst_b , memory , rw , msg_w , sb_block(5) ,
    s_msg_r );
675 BR7: blocor port map ( enable(6) , clk , rst_b , memory , rw , msg_w , sb_block(6) ,
    s_msg_r );
BR8: blocor port map ( enable(7) , clk , rst_b , memory , rw , msg_w , sb_block(7) ,
    s_msg_r );
677 BR9: blocor port map ( enable(8) , clk , rst_b , memory , rw , msg_w , sb_block(8) ,
    s_msg_r );
BR10: blocor port map ( enable(9) , clk , rst_b , memory , rw , msg_w , sb_block(9) ,
    s_msg_r );
679 BR11: blocor port map ( enable(10) , clk , rst_b , memory , rw , msg_w , sb_block(10) ,
    s_msg_r );
BR12: blocor port map ( enable(11) , clk , rst_b , memory , rw , msg_w , sb_block(11) ,
    s_msg_r );
681 BR13: blocor port map ( enable(12) , clk , rst_b , memory , rw , msg_w , sb_block(12) ,
    s_msg_r );
BR14: blocor port map ( enable(13) , clk , rst_b , memory , rw , msg_w , sb_block(13) ,
    s_msg_r );
683 BR15: blocor port map ( enable(14) , clk , rst_b , memory , rw , msg_w , sb_block(14) ,
    s_msg_r );
BR16: blocor port map ( enable(15) , clk , rst_b , memory , rw , msg_w , sb_block(15) ,
    s_msg_r );
685 BR17: blocor port map ( enable(16) , clk , rst_b , memory , rw , msg_w , sb_block(16) ,
    s_msg_r );
BR18: blocor port map ( enable(17) , clk , rst_b , memory , rw , msg_w , sb_block(17) ,
    s_msg_r );
687 BR19: blocor port map ( enable(18) , clk , rst_b , memory , rw , msg_w , sb_block(18) ,
    s_msg_r );
BR20: blocor port map ( enable(19) , clk , rst_b , memory , rw , msg_w , sb_block(19) ,
    s_msg_r );
689 BR21: blocor port map ( enable(20) , clk , rst_b , memory , rw , msg_w , sb_block(20) ,
    s_msg_r );
BR22: blocor port map ( enable(21) , clk , rst_b , memory , rw , msg_w , sb_block(21) ,
    s_msg_r );
691 BR23: blocor port map ( enable(22) , clk , rst_b , memory , rw , msg_w , sb_block(22) ,
    s_msg_r );
BR24: blocor port map ( enable(23) , clk , rst_b , memory , rw , msg_w , sb_block(23) ,
    s_msg_r );
693 BR25: blocor port map ( enable(24) , clk , rst_b , memory , rw , msg_w , sb_block(24) ,
    s_msg_r );

695 led_r <= led_r_i;
led_p <= led_p_i;

```

```

697     msg_r <= s_msg_r;
699     ss_msg_r <= s_msg_r(5 downto 0);
    Timer60: Timer_60 port map (clk ,b_on,b_play ,b_next ,b_back ,t60);
701     Timer5: Timers_2_5 port map (clk ,b_on,t2 ,t5);

703     D7S1: decod_bcd_8bit port map(ss_msg_r,s_binbcd);
    dsp1<=s_binbcd(3 downto 0);
705     dsp2<=s_binbcd(7 downto 4);

707 end ckt;

709
-----

711 library ieee;
    use ieee.std_logic_1164.all;
713 use ieee.numeric_std.all;

715 entity blocor is
    port (en, clk , rst_b, memory, rw: in std_logic;
717         data_i: in std_logic_vector (7 downto 0);
        rec_block: out std_logic;
719         data_o: out std_logic_vector(7 downto 0));
end entity;

721
architecture ckt of blocor is
723     component contador10 is
        port(clk , en, clr: in std_logic;
725             b_block: out std_logic;
             o: out std_logic_vector (3 downto 0));
727 end component;

        component ram_s is
729         port (
            clk      : in std_logic ;
731             en      : in std_logic ;
            rw       : in std_logic ;
733             addr    : in std_logic_vector (3 downto 0);
            datain   : in std_logic_vector (7 downto 0);
735             dataout  : out std_logic_vector (7 downto 0));
        end component;

737
    signal sig_addr: std_logic_vector(3 downto 0);
739
    begin

741
    z1: contador10 port map(clk ,en ,rst_b ,rec_block ,sig_addr);
743

```

```

745 z2: ram_s port map (clk ,memory,rw,sig_addr ,data_i ,data_o);
747
749
751 library ieee;
753 use ieee.std_logic_1164.all;
755 use ieee.numeric_std.all;
757
759 entity sec_ele is
761     port (clk ,b_on,b_play ,b_next,b_back,rec_i ,rst: in std_logic;
763           msg: in std_logic_vector (7 downto 0);
765           led_o,led_m,led_f,led_p,led_r: out std_logic;
767           display1 ,display2: out std_logic_vector (3 downto 0);
769           msg_s: out std_logic_vector (7 downto 0));
771 end entity;
773
775 architecture ckt of sec_ele is
777     component bloco_op is
779         port (clk , rst_w, rst_r, memory, rw, display , rst_b, hc_r, hc_w, b_next,
781             b_back, rec_i, b_on,b_play,led_r_i,led_p_i: in std_logic;
783             msg_w: in std_logic_vector (7 downto 0);
785             rec_o, mu, comp_eq, m, f, t60, t5, t2, b_block, led_m, led_f, led_p,
787             led_r: out std_logic;
789             dsp: out std_logic_vector (4 downto 0);
791             dsp1,dsp2: out std_logic_vector(3 downto 0);
793             msg_r: out std_logic_vector(7 downto 0)
795             );
797     end component;
799
801     component mde_b is
803         port ( CLK , RST , B_ON, PLAY, MU, COMP_EQ, REC, M, B_BLOCK, F, T60, T5
805             , T2: in std_logic ;
807             LED_O, LED_R, LED_P, RST_W, RST_R, RST_B, DISPLAY, HC_R, HC_W,
809             MEMORY, RW: out std_logic );
811     end component;
813
815     signal s_led_r,s_rec_o,s_led_p,s_mu,s_comp_eq,s_m,s_b_block,s_f,s_t60,s_t5,
817         s_t2,s_rst_w,s_rst_r,s_rst_b,s_display ,s_hc_r,s_hc_w,s_memory,s_rw :
819         std_logic;
821     signal s_dsp : std_logic_vector (4 downto 0);
823     begin

```

```

MDE1: mde_b port map (clk , rst , b_on, b_play, s_mu, s_comp_eq, rec_i, s_m
    , s_b_block, s_f, s_t60, s_t5, s_t2,
785     led_o, s_led_r, s_led_p, s_rst_w, s_rst_r, s_rst_b, s_display,
        s_hc_r, s_hc_w, s_memory, s_rw);

787 Bloco_op1: bloco_op port map (clk , s_rst_w, s_rst_r, s_memory, s_rw,
    s_display, s_rst_b, s_hc_r, s_hc_w, b_next,
    b_back, rec_i, b_on, b_play, s_led_r, s_led_p, msg, s_rec_o, s_mu,
    s_comp_eq, s_m, s_f, s_t60, s_t5, s_t2, s_b_block, led_m, led_f,
789     led_p, led_r, s_dsp, display1, display2, msg_s);

791     led_p <= s_led_p;

793
end ckt;

```