UNIVERSIDADE FEDERAL DE PERNAMBUCO

Departamento de Eletrônica e Sistemas

Relógio digital com despertador

Projeto 2

Isaac Neves Farias

Matheus José Araújo Oliveira

Pedro Henrique de Albuquerque Gomes

Recife, 04 de junho de 2021

Sumário

1 Introdução

O objetivo deste projeto é a aplicação prática dos conceitos vistos em aula, e nesse caso o desenvolvimento de sistemas em AHDL. Tendo isto em mente o projeto consiste em um relógio digital com alarme programável desenvolvido totalmente na linguagem AHDL (Altera Hardware Description Language).

O relógio digital deve contar até 24:00, utilizando as medidas em horas e minutos. O relógio deve possuir um modo de ajuste que é selecionado após o ativamento de uma chave, nesse mesmo modo botões são utilizados para alterar o valor do dígito (incrementando um) e para selecionar o dígito a ser ajustado. Para sair do modo ajuste normal e entrar no modo ajuste do alarme basta chegar ao fim dos dígitos ajustados. No modo ajuste do alarme o horário do alarme deve ser selecionado do mesmo modo que no ajuste anterior. O relógio deve iniciar às 00:01.

O relógio digital foi desenvolvido no software Quartus e implementado na placa Cyclone IV.

Sobre a estrutura deste relatório, a seção 2, desenvolvimento, aprofunda nos passos tomados para o relógio funcionar nas especificações solicitadas. A seção é dividida em quatro etapas após uma visão geral do problema, em que abordam: relógio contador, modo ajuste, alarme e exibição. Uma divisão esquemática pode ser observada na figura 01.

Além disso, na seção 3, manual de operação, é explicado a interface do usuário do relógio digital implementado na placa Cyclone IV. Resultados & discussão, seção 4, traz a avaliação do produto final e discute sobre as dificuldades encontradas pela equipe durante o desenvolvimento do projeto. Por fim, na seção 5, conclusão, pensamentos finais são expressos sobre o projeto como um todo.

2 Desenvolvimento

A solução desenvolvida foi dividida em 4 partes:

1. Relógio contador

A partir do clock da placa, é utilizado um sistema redutor de frequência para se ajustar com a contagem do relógio. O clock tratado é enviado para contadores (frequência de 1 Hz) que são conectados entre si, de modo que o sistema contabilize de 00:01 até 24:00.

2. Controle e ajuste

No controle e ajuste é especificado de que modo cada botão funciona e como é implementado o modo de ajuste, determinando os horários de ajuste do relógio e do alarme.

3. Alarme

No módulo alarme é recebido o horário de despertar do ajustes e então é comparado esse horário com o relógio no modo normal, é também nesse módulo que acontece a ativação do sinal sonoro e visual no momento que é ativado o alarme

4. Exibição

Para a exibição do horário no display de 7 segmentos, é utilizado um sistema multiplexador que alterne o display selecionado para que todos os dígitos sejam apresentados ao mesmo tempo. Além disso, é utilizado um decodificador para cada número de 0 a 9.

Para os casos de que os dígitos piscam, ao mesmo tempo ou em apenas um dígito, são aplicadas condições especiais referentes ao modo ajuste e alarme. O fato de piscar decorre do apagamento de alguns dígitos em determinada frequência .

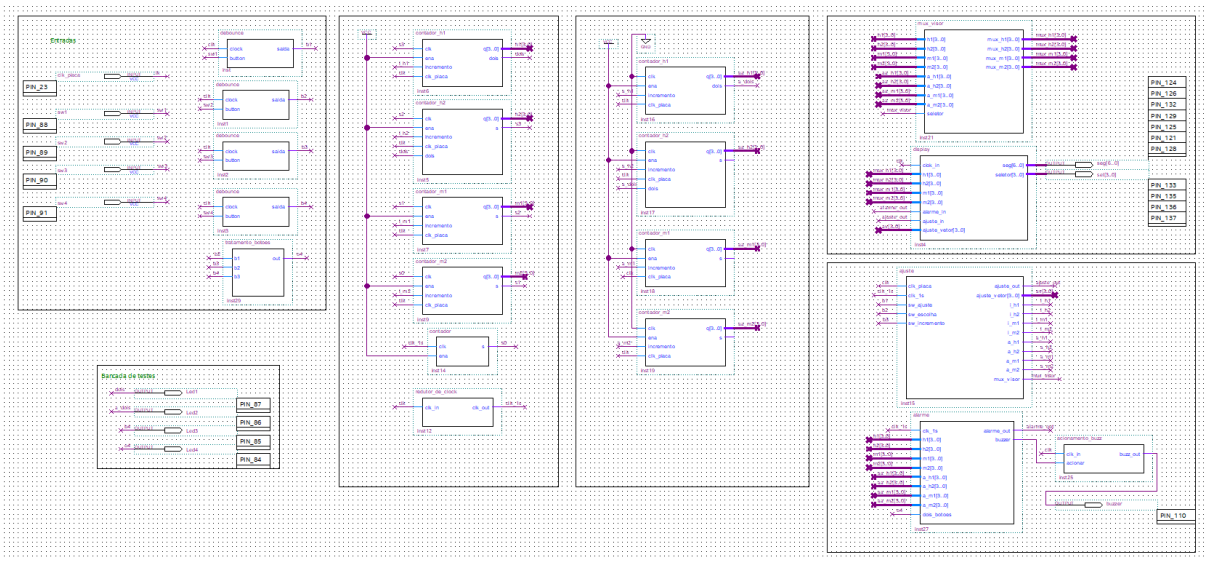

Figura 01: Projeto completo dividido nas partes do desenvolvimento

2.1 Relógio contador

2.1.1 Redução da frequência do *clock*

Para o sistema redutor de frequência, o que é aplicado de fato é apenas um contador interno que quando alcança determinado número muda a variável de saída (funcionando também como um clock), isso ocorre de acordo com o necessário. Como o clock inicial é de 50 MHz, então ao determinar o limite de contagem em 50 MHz reduzimos a frequência até 1 Hz, todo o processo pode ser observado na Figura 02.

```
SUBDESIGN redutor_de_clock
(
    clk_in : INPUT;
    clk_out : OUTPUT;
)
VARIABLE
    q[24..0]    : DFF; -- /25M(25bits) clock resultante de 2Hz

    aux : TFF;


BEGIN

    q[].clk=clk_in;
    aux.clk=clk_in;

    clk_out=aux.q;

    CASE q[].q IS
        WHEN 25000000 => --mudança de estado a cada 25M (1/2 clock)

            aux.t=VCC;

            q[].d=0; --zera o contador de 25M

        WHEN OTHERS =>

            aux.t=GND;

            q[].d=q[].q+1; -- incremento

        END CASE;

END;
```

Figura 02: Código em *AHDL* do redutor de frequência

Foi utilizado mais de um divisor de *clock,* pois para ligação do display e para executar o pisca precisaríamos de um clock maior que um segundo, porém menor que 50 MHz, isso será apresentado posteriormente.

2.1.2 Contador crescente

Para a implementação do contador até 24 horas foram utilizados 5 contadores em modo cascata de modo que:

1. Contador módulo 60, para os segundos o qual já recebe o clock tratado de 1 Hz.

2. Contador módulo 10, para contar os minutos de modo unitário, e que recebe o *clock* apenas quando o contador 1 reseta.
3. Contador módulo 6, para contar os minutos decimais, e que recebe o *clock* apenas quando o contador 2 reseta.
4. Contador módulo 10 (e módulo 4 para o caso do último dígito ser 2) , para contar as horas de modo unitário, e que recebe o *clock* apenas quando o contador 3 reseta.
5. contador módulo 3, para contar as horas decimais, o qual recebe o *clock* apenas quando o contador 4 reseta.

Para uma verificação sucinta da implementação segue na Figura 03 o modelo base utilizado no desenvolvimento do contador módulo 10.

É importante perceber também que no código abaixo estão presentes condicionamentos de contagem, e não apenas o contador em si e por isso um apresenta um tamanho extenso.

```
SUBDESIGN contador_m2
(
   clk, ena, incremento: INPUT;
    clk_placa : INPUT;
    setup_in : INPUT = VCC;

   q[3..0] : OUTPUT;
    s : OUTPUT;
)

VARIABLE
    count[3..0] : DFF;

    saux : DFF;

    primeira : DFF;

    setup : DFF;


BEGIN

    DEFAULTS
        setup.d=GND;
    END DEFAULTS;

    count[].clk=clk_placa;
    saux.clk=clk_placa;
    primeira.clk=clk_placa;
    setup.clk=clk_placa;

    s=saux.q;

    q[]=count[].q;
```

```
IF ena THEN


    IF ((clk==VCC # incremento==GND) & primeira.q==VCC) #setup.q==GND THEN

        setup.d=VCC;

        IF count[].q==9 THEN

            count[].d=0;

            primeira.d=GND;
            saux.d=VCC;

        ELSE

            count[].d=count[].q+1;

            primeira.d=GND;
            saux.d=GND;
        END IF;
    ELSE

        primeira.d=primeira.q;

        count[].d=count[].q;

        setup.d=setup.q;

    END IF;


    IF clk==GND & incremento==VCC THEN

        primeira.d=VCC;

    END IF;
```

Figura 03: Código em *AHDL* do contador módulo 10

2.2 Controle e modo ajuste

2.2.2 Entradas de controle

2.2.2.1 Botões & *Debounce*

Na aplicação, o *debounce* funciona auxiliando os botões do sistemas. Ele evita a ocorrência de oscilação de sinal no circuito. A Figura 04 mostra o esquemático externo dos *debounces,* botões e a chave do sistema.

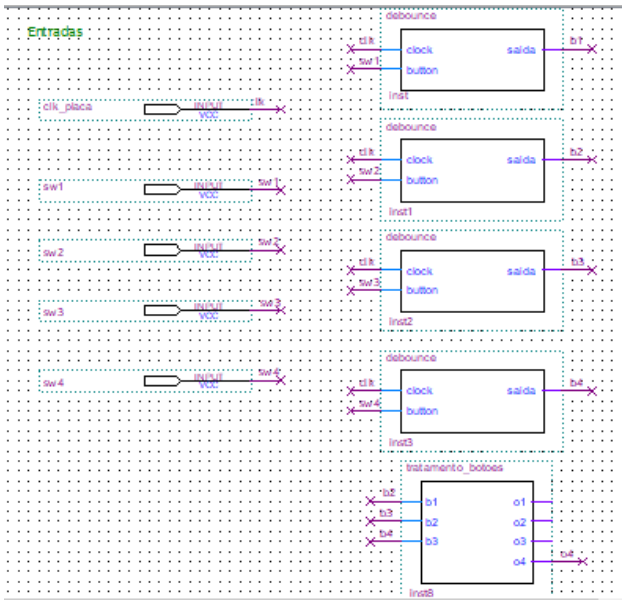

Figura 04: Esquemático externo dos botões e do *clock*

O *debounce* utilizado garante que vários pulsos (oscilações do sinal de entrada do botão) tenham como saída, dentro do *clock* de entrada, apenas uma mudança de posição da saída, o esquema implementado para o debounce está na Figura 04.

```
SUBDESIGN debounce
(
    clock, button :INPUT;
    saida :OUTPUT;
)

VARIABLE
    count[18..0] :DFF; --500.000(19bits) [100Hz ou 10ms]
    ff[1..0] : DFF;
    aux : DFF;
    counter_set: NODE;

BEGIN
    counter_set = ff[1].q $ ff[0].q; --XOR para controlar a contagem do debounce
    count[].clk = clock;
    ff[].clk = clock;
    aux.clk = clock;

    -- o ff[] garante que o botao não ta na fase de inicial, nem na final, de pressionamento.

    ff[0].d = button;
    ff[1].d = ff[0].q; --atraso de 1 clock
    saida = aux.q;

    IF (counter_set == 1) THEN --se os valores dos FF's forem diferentes, a contagem começa de novo
        count[].d = 0;
    ELSE --as saidas dos FlipFlops iguais indicam que a contagem pode começar (10ms para o debounce de um botão)
        IF (count[].q == 500000) THEN
            count[].d = 0;
            aux.d = ff[1].q;
        ELSE
            count[].d = count[].q + 1;
            aux.d = aux.q;
        END IF;
    END IF;
END;
```

Figura 04: Circuito de debounce em *AHDL*.

O bloco tratamento de botões é responsável apenas para identificar quando dois botões são pressionados ao mesmo tempo, esse esquema é verificado na Figura 05 abaixo.

```
SUBDESIGN tratamento_botoes
(
   b1, b2, b3 : INPUT;
   o1, o2, o3, o4 : OUTPUT;


)
VARIABLE
aux : NODE;


BEGIN

-- saida o4 corresponde ao aperto de dois botões
-- como ja usamos o debounce antes, podemos fazer a logica direta, sem atrasos

   IF ((b1==GND & b2==GND) # (b1==GND & b3==GND) # (b2==GND & b3==GND)) THEN

      o4=GND;

   ELSE
      o4=VCC;

   END IF;

END;
```

Figura 05: Tratamento de botões em *AHDL*.

2.2.2.2 Modo ajuste

É um módulo mais complexo e consiste em um bloco com a propriedade de ajuste do horário e alarme do equipamento.

Esse módulo possui como entrada um clock de um segundo, clock da placa e as entradas de controle (botões e *switch*), e tem como saída o horário de ajuste do alarme, do horário, seletor do display e indicadores de alteração no ajuste (piscar).

Na Figura 06, é mostrado o sistema externo do bloco de ajuste.

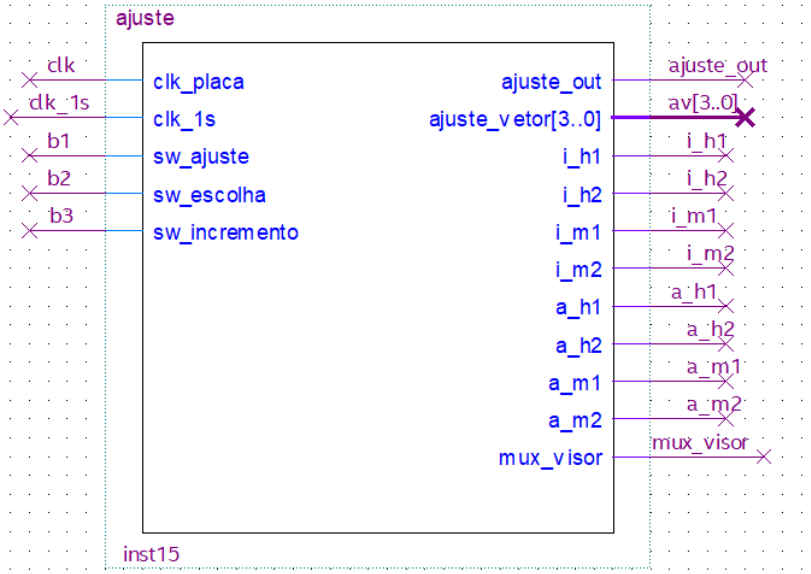

Figura 06: Bloco ajuste.

Quando a chave for acionada, o relógio entrará no modo de ajuste e enviará um sinal (*ajuste_out*) para o *display* piscar na frequência de 1Hz (*clk_1s*). Estando no modo de ajuste o sinal de entrada *sw_escolha* (vindo de um botão) incrementa um contador de 3 bits. Este determina para qual contador externo será enviado o sinal de incremento definido pela entrada *sw_incremento* (também vindo de um botão).

O bit mais significativo desse contador determina se o contador externo ajustado será do relógio ou do alarme, além de enviar um sinal (*mux_out*) para o multiplexador do visor (*mux_visor*). Por fim, os dois algarismos menos significativos do contador também definem o sinal de saída *ajuste_vector*, que vai para o *display* indicando qual digito piscar.

A implementação da lógica citada acima pode ser verificada nas Figuras 7, 8 e 9 abaixo.

```
VARIABLE

   aux, aux2 : NODE;
   primeira_esc, primeira_inc : DFF;
   aux_vetor[2..0] : DFF; -- de 0-7 (3bits)

BEGIN

   ajuste_out=aux;

   --avisa quando é pra mostrar o alarme e quando é pra mostrar a hr normal
   mux_visor= aux_vetor[2].q;

   aux_vetor[].clk=clk_placa;
   primeira_inc.clk=clk_placa;
   primeira_esc.clk=clk_placa;

   IF sw_ajuste==GND THEN

      aux=clk_1s; --faz o digito piscar

      IF sw_escolha==GND & primeira_esc==VCC THEN -- controla o digito que pisca

         IF aux_vetor[].q==7 THEN

            aux_vetor[].d=0;

            primeira_esc=GND;

         ELSE
            aux_vetor[].d=aux_vetor[].q+1;

            primeira_esc=GND;
         END IF;

      ELSE
         aux_vetor[].d=aux_vetor[].q;
      END IF;


      IF sw_escolha==VCC THEN --usado para pular apenas 1 digito quando descer o botao

            primeira_esc=VCC;

      END IF;
```

Figura 7: Lógica do bloco de ajuste em AHDL mostrando o contador de 3 bits

```
----------------------------------------------

  --incremento dos somadores

     IF sw_incremento==GND & aux_vetor[].q==B"000" THEN

        i_h1=GND;

     ELSE

        i_h1=VCC;

     END IF;

     IF sw_incremento==GND & aux_vetor[].q==B"001" THEN

        i_h2=GND;

     ELSE

        i_h2=VCC;

     END IF;
```

```
   ELSE

      aux=GND;

      i_m1=VCC;
      i_m2=VCC;
      i_h1=VCC;
      i_h2=VCC;

      a_m1=VCC;
      a_m2=VCC;
      a_h1=VCC;
      a_h2=VCC;

   END IF;
```

Figura 8: Bloco de ajuste em AHDL mostrando o sinal de incremento dos dígitos.

```
165     TABLE
166         aux_vetor[].q=>ajuste_vetor[];
167         --indica o digito que irá piscar
168
169         B"000" => B"1000";
170         B"001" => B"0100";
171         B"010" => B"0010";
172         B"011" => B"0001";
173
174         B"100" => B"1000";
175         B"101" => B"0100";
176         B"110" => B"0010";
177         B"111" => B"0001";
178
179     END TABLE;
180
181 END;
```

Figura 9: Bloco de ajuste em *AHDL* mostrando a tabela que traduz o contador para a indicação do dígito que pisca.

2.3 Alarme

Primeiramente, os blocos usados para o relógio são reaproveitados como registradores do horário do alarme, removendo apenas o sinal de entrada de *clk*. O esquemático externo pode ser observado na Figura 10, enquanto a lógica interna de incremento em *AHDL* é a mesma do relógio e pode ser revista no exemplo da Figura 3.

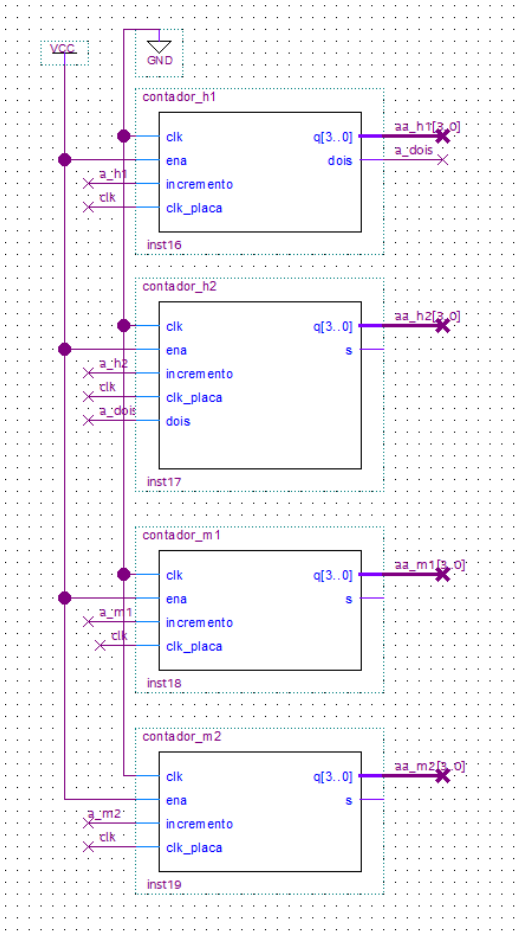

Figura 10: Blocos de registro do horário do alarme.

O módulo alarme é responsável por receber o horário do alarme guardados nos blocos de registro, comparar esse horário com o valor atual do relógio e então ativar o despertar do relógio (*buzzer* e piscagem dos números*)*.

O bloco externo desse módulo pode ser observado na Figura 11.

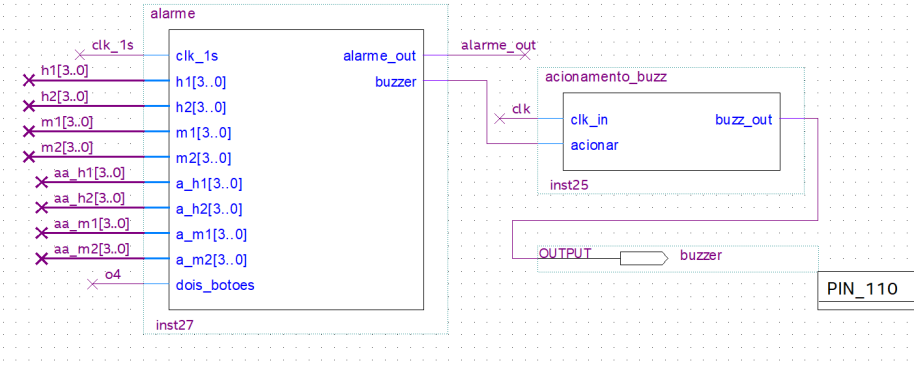

Figura 11: Bloco externo do modelo alarme.

O código interno do módulo, pode ser observado na Figura abaixo 12.

```
SUBDESIGN alarme
(

    clk_1s : INPUT;

    h1[3..0], h2[3..0], m1[3..0], m2[3..0], a_h1[3..0], a_h2[3..0], a_m1[3..0], a_m2[3..0] : INPUT;

    dois_botoes : INPUT;

    alarme_out : OUTPUT;
    buzzer : OUTPUT;
)
BEGIN
    IF h1[]==a_h1[] & h2[]==a_h2[] & m1[]==a_m1[] & m2[]==a_m2[] THEN

        alarme_out=clk_1s;
        buzzer=VCC;

    END IF;

    IF dois_botoes==GND THEN

        alarme_out=GND;
        buzzer=GND;

    END IF;
END;
```

Figura 12: Construção interna do módulo de alarme.

Para o acionamento do buzzer é necessário que exista uma vibração a partir de um clock inicial, essa vibração é responsável por determinar a frequência sonora do buzzer que deve ser modulada de forma que seja um sinal audível.

O código responsável por essa aplicação é ilustrado na Figura 13.

```
SUBDESIGN acionamento_buzz
(
    clk_in : INPUT;
    acionar : INPUT;

    buzz_out : OUTPUT;
)
VARIABLE

    frequencia[14..0] : DFF; -- /25K(11bits) clock resultante de 2KHz
    q[23..0] : DFF;
    aux : TFF;
    aux2 : TFF;

BEGIN

    q[].clk=clk_in;
    frequencia[].clk=clk_in;
    aux.clk=clk_in;
    aux2.clk=clk_in;

    CASE q[].q IS
        WHEN 12500000 => --mudança de estado a cada 25M (1/2 clock)

            aux2.t=VCC;

            q[].d=0; --zera o contador de 25M

        WHEN OTHERS =>

            aux2.t=GND;

            q[].d=q[].q+1; -- incremento

    END CASE;

    CASE frequencia[].q IS
        WHEN 25000 => --mudança de estado a cada 2K (1/2 clock)

            aux.t=VCC;

            frequencia[].d=0; --zera o contador de 2K

        WHEN OTHERS =>

            aux.t=GND;

            frequencia[].d=frequencia[].q+1; -- incremento

    END CASE;

    IF acionar == VCC THEN

        IF aux2.q == VCC THEN

            buzz_out=aux.q;

        ELSE

            buzz_out = GND;

        END IF;

    ELSE

        buzz_out = GND;

    END IF;

END;
```

Figura 13: Código de ativação do *buzzer*.

2.4 Exibição

2.4.1 Display de 7 segmentos & Decodificador (conceitos)

Conforme o relógio avança, temos 4 números a serem exibidos. Cada um desses números representa um algarismo decimal que são exibidos em quatro displays de 7 segmentos. Para isto, faz-se necessário a implementação de um decodificador que recebe um número decimal de entrada e uma saída de 7 bits, especificando qual led do display deve ser aceso.

A Figura 14 ilustra um esquemático genérico de um decodificador com um display.

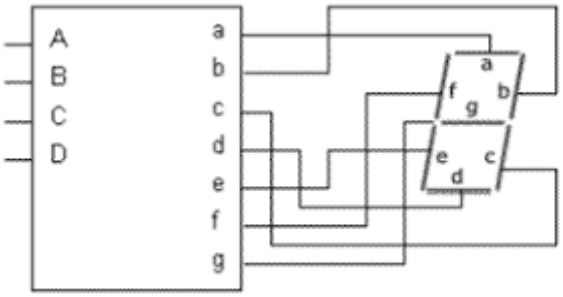

Figura 14: Esquemático de um decodificador e um display de sete segmentos.

2.4.2 Seletor

Como o relógio possui dois modos de exibição (relógio normal e modo ajuste) então é aplicado um seletor interno para escolher quais dígitos serão exibidos, esse processo de seleção é percebido na Figura 15.

```
SUBDESIGN mux_visor
(
    h1[3..0], h2[3..0], m1[3..0], m2[3..0] : INPUT;
    a_h1[3..0],a_h2[3..0],a_m1[3..0],a_m2[3..0] : INPUT;

    seletor : INPUT;

    mux_h1[3..0], mux_h2[3..0], mux_m1[3..0], mux_m2[3..0] : OUTPUT;

)

VARIABLE
aux_h1[3..0],aux_h2[3..0], aux_m1[3..0], aux_m2[3..0] : NODE;

BEGIN
    mux_h1[]=aux_h1[];
    mux_h2[]=aux_h2[];
    mux_m1[]=aux_m1[];
    mux_m2[]=aux_m2[];

    IF seletor==0 THEN
        aux_h1[]=h1[];
        aux_h2[]=h2[];
        aux_m1[]=m1[];
        aux_m2[]=m2[];

    ELSE

        aux_h1[]=a_h1[];
        aux_h2[]=a_h2[];
        aux_m1[]=a_m1[];
        aux_m2[]=a_m2[];


    END IF;
END;
```

Figura 15: código seletor do display em *AHDL*.

2.4.3 Multiplexador e decodificação

Os displays disponíveis na placa possuem os 7 pinos de acionamento em paralelo e mais 4 pinos para selecionar qual display acionar.

Para acionar os quatro displays ao mesmo tempo foi desenvolvido uma máquina de estado que tem como entrada um *clock* e varia a saída do multiplexador na forma 00, 01, 10 e 11. No caso dos ajustes de horário e alarme é utilizado o mesmo tipo de multiplexador para seleção de display, parte do processo é ilustrado nas Figura 16.

```
IF ajuste_in==GND THEN

   IF alarme_in==GND THEN

      CASE aux[] IS  %alterna os digitos de display a partir de aux %
         WHEN B"00" =>  seletor[] = B"0111";
                        disp[] = h1[];
         WHEN B"01" =>  seletor[] = B"1011";
                        disp[] = h2[];
         WHEN B"10" =>  seletor[] = B"1101";
                        disp[] = m1[];
         WHEN B"11" =>  seletor[] = B"1110";
                        disp[] = m2[];

      END CASE;
```

Figura 16: Código multiplexador em *AHDL*.

Para a implementação do decodificador de fato basta implementar uma tabela que segmenta cada número para a sua representação em bits do display como é apresentado na Figura 17, lembrando que o display da placa é ânodo comum.

```
TABLE
   disp[]=>seg[];

   B"0000" => B"0000001";
   B"0001" => B"1001111";
   B"0010" => B"0010010";
   B"0011" => B"0000110";
   B"0100" => B"1001100";
   B"0101" => B"0100100";
   B"0110" => B"0100000";
   B"0111" => B"0001111";
   B"1000" => B"0000000";
   B"1001" => B"0000100";
END TABLE;
```

Figura 17: Tabela de decodificação

Dessa forma o display consegue apresentar os quatro dígitos ao mesmo tempo e alternar exibição do modo ajuste (horário e alarme) e do relógio usual.

2.4.4 Piscar os leds do *display*

Para a função piscar dos *display* é usada uma *flag* do modo ajuste e outra do alarme que retornam qual ou quais dígitos do display devem piscar e em que circunstâncias.

Para o alarme e o ajuste as condicionais podem ser observadas nas Figuras 18 e 19.

```
ELSE --apaga o digito selecionado no ajuste_vetor

   IF ajuste_vetor[]==B"1000" THEN

      CASE aux[] IS
         WHEN B"00" =>  seletor[] = B"1011";
                        disp[] = h2[];

         WHEN B"01" =>  seletor[] = B"1011";
                        disp[] = h2[];
         WHEN B"10" =>  seletor[] = B"1101";
                        disp[] = m1[];
         WHEN B"11" =>  seletor[] = B"1110";
                        disp[] = m2[];

      END CASE;
   END IF;
```

Figura 18: Faz piscar o dígito 1 ao entrar no modo ajuste ao apagar o número

```
IF alarme_in==GND THEN

   CASE aux[] IS  %alterna os digitos de display a partir de aux %
      WHEN B"00" =>  seletor[] = B"0111";
                     disp[] = h1[];
      WHEN B"01" =>  seletor[] = B"1011";
                     disp[] = h2[];
      WHEN B"10" =>  seletor[] = B"1101";
                     disp[] = m1[];
      WHEN B"11" =>  seletor[] = B"1110";
                     disp[] = m2[];

   END CASE;

ELSE

   seletor[]=B"1111"; --apaga o display para alarme_in=1

END IF;
```

Figura 19: Apaga todos os números, fazendo assim que pisquem.

3 Manual de operação

Nesta seção consta o manual de operação do relógio digital e do seu despertador, a Figura 20 abaixo, apresenta uma legenda que determina o local de cada botão e chave citados nos parágrafos abaixo.

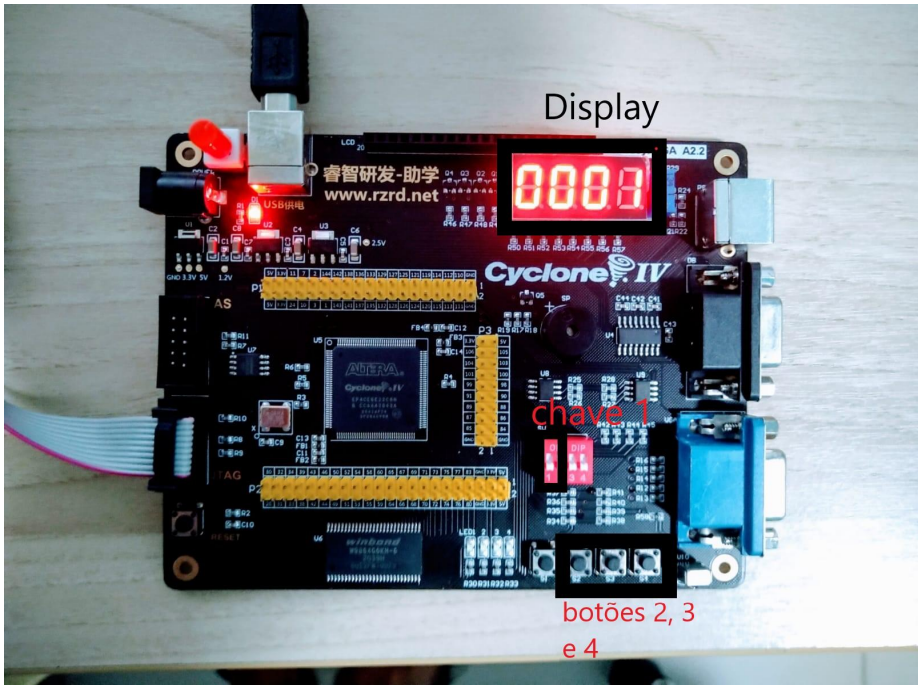

Figura 20: Esquemático Cyclone IV

Em modo de operação usual o relógio funciona normalmente contando de 00:01 até 00:00 (24:00), nesse modo o alarme é ajustado naturalmente para às 00:00 horas. Dito isto, o usuário tem a capacidade de modificar o horário do relógio e o horário do alarme, para ajustar de acordo com o horário que quiser.

Os processos de ajuste serão descritos abaixo:

Ajuste de horário e alarme:

- Ao ativar a chave 1 o relógio entra no modo ajuste de horário, o desativamento sai desse mesmo modo.
- Após entrar no modo ajuste o dígito que estiver piscando será aquele que será selecionado para alteração.
- Ao pressionar o botão 2 o usuário confirma o ajuste do dígito e troca o número selecionado para a alteração.
- Ao pressionar o botão 3 o dígito será incrementado.
- Ao pressionar o botão 2 durante o modo ajuste de horário, e já estiver no quarto dígito do display, o sistema entra no modo ajuste de alarme.
- O ajuste do alarme funciona do mesmo modo que o ajuste do horário.
- Para sair do modo ajuste do alarme também, basta desativar a chave 1.

Alarme: Quando o horário do relógio chega ao horário do alarme, o equipamento produz um sinal sonoro ao mesmo tempo que os leds do *display* piscam. Para o relógio voltar para o funcionamento normal basta que o usuário pressione ao mesmo tempo dois dos 3 botões de uso do relógio (2,3 e 4).

4 Resultados & Discussão

De modo geral os requisitos para o funcionamento do projeto foram completamente correspondidos.

Para melhor compreensão do sistema a visualização RLT de todo o projeto pode ser observada na Figura 21.

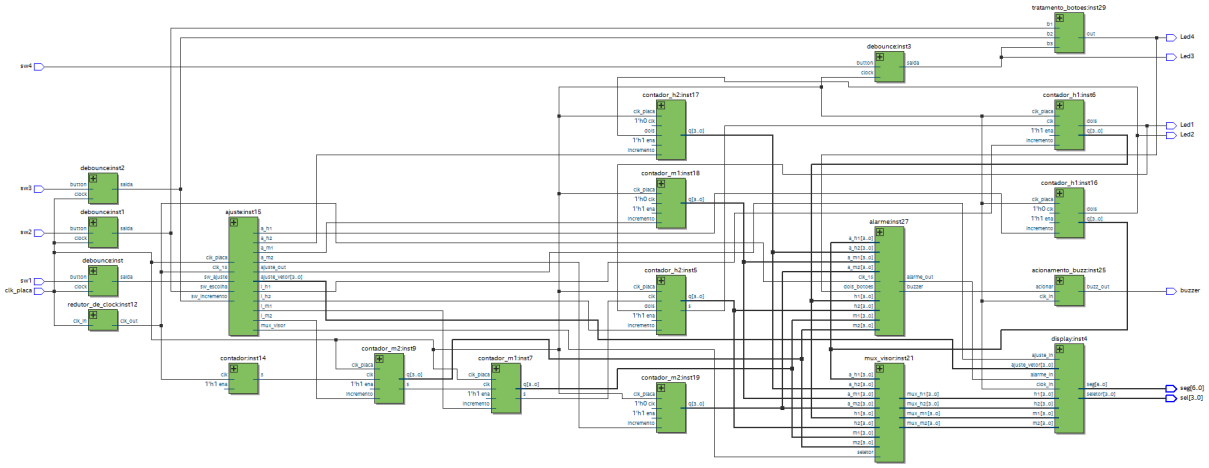

Figura 21: Visualização RLT do projeto

4.1 Principais problemas encontrados

1. Dois do contador: o contador contava até 29 ao invés de 24, isso porque o sistema não percebia que o primeiro dígito estava em 2, então foi criada uma flag de identificação e o problema foi resolvido.

2. clock do contador: O mesmo clock de atualização do flip flop era o da contagem, portanto ele não atualizava na medida certa. Para corrigir isso foi implementado um clock de maior frequência no contador.

3. Alarme dentro do ajuste: O módulo de alarme estava dentro do ajuste o que trazia complexidade para o código e mal funcionamento no incrementador, portanto foram separados.

4. O despertador não desliga: O despertador não desativa ao apertar dois botões ao mesmo tempo.

5. Buzzer pwm: O buzzer não estava funcionando pois não recebia uma frequência do som e sim apenas um sinal de VCC, esse problema foi resolvido após implementar um módulo de ativação do buzzer responsável por definir essa frequência sonora.

6. Não começa às 00:01: De fato não foi possível implementar uma entrada de início para o contador, portanto começa às 00:00.

7. O incremento do ajuste: No modo ajuste ocorria somas ao apertar botões de saídas ou outros botões que não tinha essa intenção, esse problema foi resolvido implementando condicionais para a soma.

4.2 Debugs

Como ferramentas de *debug* foram utilizados leds para certificação de funcionamento do relógio antes do display estar em pleno funcionamento.

Para debug de toda a parte de ajuste o relógio utilizado não foi em horas pois é inviável para a realização de testes, ao invés disso foi utilizado um relógio com valores em segundos e minutos.

5 Conclusão

O resultado final apresentado foi satisfatório, visto o que é pedido pelo experimento. Por meio dessa prática foi possível desenvolver e otimizar o conceito de gerenciamento de projetos, assim como uma experiência em equipe.

Além disso, vale destacar que, devido à crescente complexidade dos projetos dos circuitos de eletrônica digital, fica evidente a importância de desenvolver habilidades em *HDLs*, sendo *AHDL* uma boa linguagem introdutória. A prática do presente projeto permitiu aos alunos a exploração e desenvolvimento de tais habilidades.

Por fim, o advento da invenção do relógio digital com alarme trouxe um impacto incalculável no gerenciamento do tempo para a humanidade. Desde a regulação do horário para acordar até a maior precisão no controle de tempo em processos industriais diversos. Um maior controle sobre o tempo deu ao humano comum a possibilidade de uma performance mais elevada, em contrapartida, o mesmo controle dominou o humano comum com prazos e expectativas cada vez mais exigentes.

"A vida é sofrimento." - Engenharia, Estudante de.