Projeto Relógio na FPGA

Eli Jose e Pedro Azambuja

- Introdução

Este projeto consiste na criação de um relógio digital, em VHDL, com as seguintes características:

- Indica horas e minutos e segundos;
- Possui um sistema de passagem acelerada de tempo para mudar o horário;
- Possui um sistema para trocar o modo de visualização do horário de base 12 (AM/PM) para base 24.
- Montador (assembler) para a sequência de instruções criadas;
- Barramento de dados em 8 bits;
- Barramento de memória em 8 bits;
- Horário mostrado através de um display de 7 segmentos;
- Possui seleção da base de tempo:
    - Para mostrar a passagem das 24 horas em tempo reduzido.

A implementação física do projeto foi feita utilizando uma placa FPGA Cyclone IV EP4CE115F29C7.

- Especificações técnicas

- **Pseudocódigo do relógio**
  Abaixo está exposto o pseudocódigo em Python da lógica implementada no relógio. Ao lado de cada linha há, em forma de comentário, o que seria um primeiro rascunho do código em Assembly.

```
while(1):

    time.sleep(base_tempo)
                                #LINHA_Y
    if(passou_segundo == true): #LOAD IO_TEMPO REG_TEMPO
                                #CMP  REG_TEMPO 1
                                #JNE  LINHA_Y

        us += 1          #ADD 0x01 REG_US

        if(us == 10):    #CMP REG_US 0X0A
                         #JNE LINHA_X
            us = 0       #MOV 0x00 REG_US
            ds += 1      #ADD 0x01 REG_DS
        if(ds == 6):     #CMP REG_DS 0X06
                         #JNE LINHA_X
            ds = 0       #MOV 0x00 REG_DS
            um += 1      #ADD 0x01 REG_UM
        if(um == 10):    #CMP REG_UM 0X0A
                         #JNE LINHA_X
            um = 0       #MOV 0x00 REG_UM
            dm += 1      #ADD 0x01 REG_DM
        if(dm == 6):     #CMP REG_DM 0X06
                         #JNE LINHA_X
            dm = 0       #MOV 0x00 REG_DM
            uh += 1      #ADD 0x01 REG_UH
        if(uh == 10):    #CMP REG_UH 0X0A
                         #JNE LINHA_X
            uh = 0       #MOV 0x00 REG_UH
            dh += 1      #ADD 0x01 REG_DH
        if(dh == 2):     #CMP REG_DH 0X02
                         #JNE LINHA_X
            if(uh == 4):#CMP REG_UH 0X04
                         #JNE LINHA42
                uh= 0    #MOV 0x00 REG_UH
                dh= 0    #MOV 0x00 REG_DH
                         #LINHA_X

    print('{0}{1}:{2}{3}:{4}{5}\n' .format(dh, uh, dm, um, ds, us)); #STORE REG_US LCD_US
                                                                     #STORE REG_DS LCD_DS
                                                                     #STORE REG_UM LCD_UM
                                                                     #STORE REG_DM LCD_DM
                                                                     #STORE REG_UH LCD_UH
                                                                     #STORE REG_DH LCD_DH
                                                                     #JUMP LINHA_Y
```

- **Código em Assembly**

  Abaixo estão as 45 primeiras linhas do código em assembly apenas para ilustrar o formato das instruções e do código. As instruções completas usadas para o projeto estão no arquivo *assembly_relogio.asm* no repositório do Github.

```
MOV REG_SIGLA $15
MOV REG_TRACO $14
MOV REG_UM $0
MOV REG_DM $5
MOV REG_UH $1
MOV REG_DH $2
MOV REG_AM_UH $9
MOV REG_AM_DH $0
MOV REG_ZERA $13
STORE REG_SIGLA LCD_US
STORE REG_TRACO LCD_DS
STORE REG_UM LCD_UM
STORE REG_DM LCD_DM
STORE REG_UH LCD_UH
STORE REG_DH LCD_DH
LINHA_Y:
LOAD REG_TEMPO IO_TEMPO
CMP REG_TEMPO $1
JNE LINHA_Y
LOAD REG_CLEAR CLEAR_TEMPO
ADD REG_US $1
CMP REG_US $10
JNE LINHA_Z2
MOV REG_US $0
ADD REG_DS $1
CMP REG_DS $6
JNE LINHA_Z2
MOV REG_DS $0
ADD REG_UM $1
CMP REG_UM $10
JNE LINHA_Z2
MOV REG_UM $0
ADD REG_DM $1
CMP REG_DM $6
JNE LINHA_Z2
MOV REG_DM $0
ADD REG_UH $1
ADD REG_AM_UH $1
CMP REG_UH $10
JNE LINHA_Z1
MOV REG_UH $0
ADD REG_DH $1
LINHA_Z1:
CMP REG_AM_UH $10
JNE LINHA_Z2
```

- **Arquitetura do processador:**

No processador criado foi utilizada a arquitetura Registrador – Registrador com o uso de um Banco de Registradores com 8 registradores. Neste foram armazenados os dados dos dígitos de horas, minutos e segundos do relógio, tanto para o formato em base 24 quanto base 12. Além disso, nesses registradores estão o valor da passagem de tempo (determinar se passou ou não 1 segundo) e valor do switch de transição de base 12 para 24. Esse banco de registradores está descrito e detalhado no item abaixo. Além disso, há também uma Unidade de controle para ativar os pontos de controle de acordo com a instrução a ser realizada pelo processador, uma ROM, onde estão armazenadas a sequência de instruções do processador e uma ULA, responsável por realizar as operações necessárias com os dados.

- **Fluxo de dados**

- **Endereçamento do Banco de Registradores**

Para cada dado a ser usado na lógica do relógio, utilizou-se um endereço do banco registradores para armazená-lo, como descrito abaixo:

Nome do registrador	**Endereço <5 bits>**	**Descrição**
REG_TEMPO	00011	Guarda dado da Base de Tempo
REG_US	00001	Unidade de Segundo
REG_DS	00010	Dezena de segundo
REG_UM	00100	Unidade de Minuto
REG_DM	01000	Dezena de Minuto
REG_UH	10000	Unidade de Hora em base 24
REG_DH	10001	Dezena de Hora em base 24
REG_CLEAR	01111	Clear da Base Tempo
REG_SWITCH	01010	Guarda dado do switch da base 24/12
REG_AM_DH	10011	Unidade de Hora em base 12
REG_AM_UH	10101	Dezena de Hora em base 12
REG_SIGLA	11110	Guarda a sigla do horário (AM ou PM)
REG_TRACO	11010	Usado para mostrar traço no display
REG_ZERA	11011	Usado para limpar o display

Os registradores REG_TEMPO e REG_CLEAR foram utilizados, respectivamente, para armazenar e resetar os dados da passagem de 1 segundo provenientes do periférico da Base de Tempo. E o registrador REG_SWITCH foi usado para armazenar o estado atual do switch de seleção do modo de visualização das horas (base 12 ou 24).

- **Mapa de Memória (Periféricos)**

Periférico	**Endereço <8 bits>**
IO_TEMPO (base de tempo)	0000 0000
CLEAR_TEMPO	0000 0001
IO_SWITCH (base 24 ou 12)	0000 1001

LCD_US	0000 0011
LCD_DS	0000 0100
LCD_UM	0000 0101
LCD_DM	0000 0110
LCD_UH	0000 0111
LCD_DH	0000 1000

- **Formato das instruções:**

OpCode	**addrA**	**reservado**
3 bits	5 bits	8 bits

O opcode é o binário da instrução a ser realizada, o addrA é o endereço do registrados a ser acessado no banco de registradores e o Reservado é um campo disponível para guardar um número inteiro positivo (instruções de ADD, MOV e CMP), para guardar o endereçamento de um periférico (instruções de LOAD e STORE), ou para guardar o endereço de uma label (instruções de JUMP e JNE).
As instruções de JNE e JUMP não levam um addrA, sendo esse campo preenchido apenas por 0s.

- **Instruções usadas:**

Instrução	**Binário**	**Descrição**
LOAD	000	Carrega o dado de um endereço I/O em um Registrador.
CMP	001	Compara o dado de um Registrador com um imediato.
MOV	010	Move o valor de um imediato para um Registrador.
ADD	100	Adiciona o valor de um imediato para um Registrador.
STORE	101	Envia para um endereço I/O o dado de um Registrador.
JNE	110	Realiza um JMP para a label do imediato se Flag CMP for 0
JUMP	111	Realiza um JMP para a label do imediato.

- **Modos de endereçamento:**

Linha de Instrução	**Binário <16 bits>**
LOAD REG_TEMPO IO_TEMPO	000 00000 0000 0000
CMP REG_TEMPO $1	001 00000 0000 0001
JNE LINHA_Y	110 00000 0000 0000
STORE REG_US LCD_US	101 00001 0000 0011
ADD REG_US $1	100 00001 0000 0001
MOV REG_US $0	010 00001 0000 0000
JUMP LINHA_Y	111 00000 0000 0000

- **Listagem dos pontos de controle e sua utilização:**

MUX JMP (1 bit): Mux utilizado para determinar se o processador deve realizar um JUMP ou continuar com a leitura normal das instruções.

WR Reg (1 bit): Enable para a escrita em um endereço no Banco de Registradores.

MUX MAIN (2 bits): Mux utilizado para selecionar a entrada de dados do Banco de Registradores, sendo as opções: Imediato, ULA e I/O.

OP ULA (3 bits): Indica a ULA qual operação deve ser efetuada entre suas entradas.

WR I/O (1 bit): Enable para a escrita nos periféricos.

RD I/O (1 bit): Enable para a leitura dos periféricos.

Observação: Para determinar esses pontos de controle, a Unidade de Controle recebe o opcode da instrução a ser realizada e, para a instrução de JNE (JUMP IF NOT EQUAL), ela também recebe a **flag de comparação** que sai da ULA, resultante da instrução anterior de CMP.

- **Operações da ULA**

Operação	**Binário <3>**	**Descrição**
ADD	000	Passa A + B para a saída.
EQUAL	001	Se A = B, Flag COMPARACAO = 1.
Passa A	011	Passa o valor da entrada A para a saída.
Passa B	010	Passa o valor da entrada B para a saída.
Nada	111	Não faz nada

- **Pontos de Controle para cada Instrução**

Instrução	**MUX JMP**	**WR Reg**	**MUX MAIN**	**OP ULA**	**WR I/O**	**RD I/O**
LOAD	0	1	10	111	0	1
CMP	0	0	11	001	0	0
JNE	1	0	11	111	0	0
ADD	0	1	01	000	0	0
MOV	0	1	00	111	0	0
STORE	0	0	11	111	1	0
JUMP	1	0	11	111	0	0

- **Fluxo de Dados:**

Utilizamos um Banco de registradores para armazenar os valores manipulados. Há um Mux para selecionar se a entrada B da ULA vem do I/O (para realizar comparações) ou do Imediato (para realizar somas e comparações). Há um Mux para verificar se a

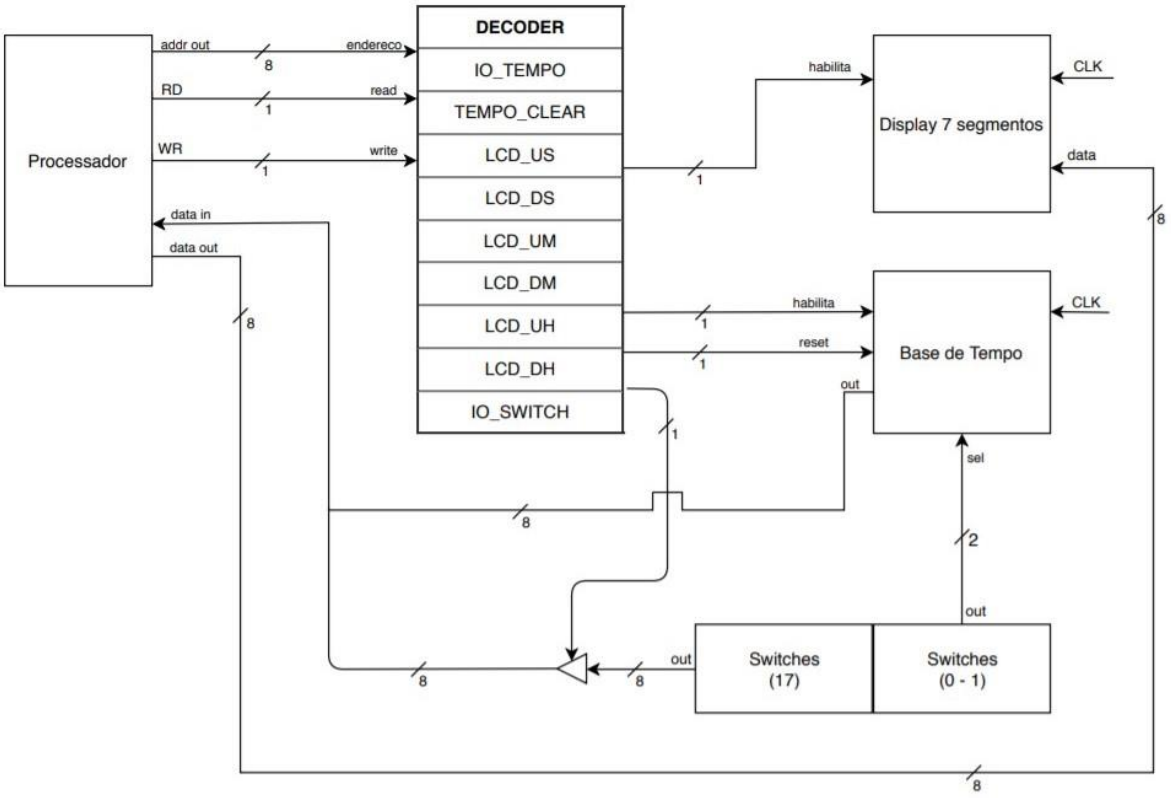
instrução é de um JUMP/JNE, sendo ele selecionado pela Unidade de Controle. Caso seja de JMP/JNE o valor passado para o PC é o de imediato do JUMP/JNE que vem nos bits de Reservado da Instrução. Há um Mux para selecionar se o dado a ser escrito em um registrador do Banco de registradores vem da ULA, resultado de alguma operação, ou se ele vem de um imediato, por exemplo, quando zeramos o valor das horas na passagem de 23:00 para 00:00. Há uma ULA que possui duas entradas, a entrada A recebe a saída do Banco de Registradores do dado lido de algum registrador, a entrada B recebe algum imediato da ROM ou um dado do I/O, dependendo do Mux. A saída da ULA é direcionada a saída da Unidade de Processamento e ao Banco de Registradores. A Unidade de Controle ativa os pontos descritos no tópico abaixo.

- **Assembler**

  Foi criado um assembler para converter o código Assembly no formato .***asm*** para um código em binário utilizável pela ROM da FPGA, no formato .***mif***. Este assembler, escrito em Python está no repositório do projeto, no arquivo *assembler.py* e permite a montagem automática e genérica para as instruções do processador criado. Seu funcionamento se inicia a partir da leitura de todas as labels de JUMP presentes no código e seu armazenamento em uma lista, além de sua retirada do código. Depois é adicionado o cabeçalho do arquivo no formato .mif, necessário para ser lido pela ROM. Por fim, as instruções são substituídas pelos seus opcodes em binário assim como os nomes dos registradores e das posições de I/O são substituídos pelos seus endereços em binário e as labels de JUMP e JNE são substituídas pelas suas respectivas linhas em binário. O código convertido é salvo no arquivo *binário.mif.*
  Caso seja necessário adicionar novos registradores ao processador, endereços de I/O ou novas instruções, basta modificar os dicionários presentes no arquivo do próprio assembler, incluindo neles os novos endereços ou instruções.

- **Diagrama dos periféricos**

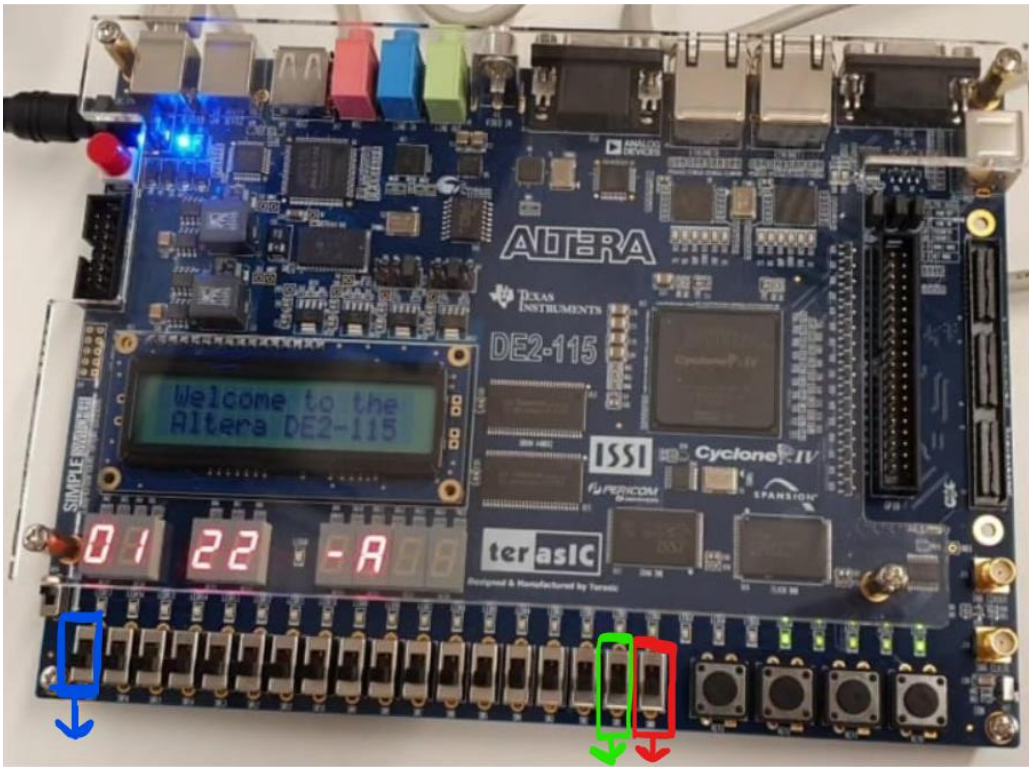
  Abaixo está ilustrado o diagrama dos periféricos utilizados no projeto.



O funcionamento dessa parte se baseia no Mapa de endereços (Decoder), nele estão codificados o endereçamento a cada um dos periféricos que são utilizados pelo processador. Assim, para que a Base de tempo seja lida, por exemplo, o processador envia o endereço deste I/O pelo addr out e “1” para o RD para o Decoder. Então, o Decoder decodifica o endereço enviado e envia um sinal de habilita para a Base de Tempo. Está por sua vez, após receber esse sinal escreverá na saída out seu valor concatenado com 7 bits mais significativos e esse valor entrará no Processador pela entrada data in e será utilizada na lógica interna deste.
Em outro exemplo, para que seja escrito um valor em um Display de 7 segmentos, o processador envia o endereço deste I/O pelo addr out e “1” para o WR para o Decoder, além do dado a ser escrito no display pelo data out. Então, o Decoder decodifica o endereço enviado e envia um sinal de habilita para o Display de 7 segmentos endereçado. Este por sua vez, após receber esse sinal decodifica a sequência de bits recebida na entrada data e mostra no display o caractere correspondente.

- Instruções de Uso

Ao ligar o relógio na placa FPGA ele já inicia mostrando as horas e minutos e funciona normalmente.



- **Ajustar o horário:**

Para ajustar o horário basta acionar os dois primeiros switches da placa, marcados na imagem em verde e vermelho. O Switch em vermelho aumenta a velocidade de passagem de tempo em 2 vezes e o verde aumenta em 4 vezes, caso os dois sejam acionados a velocidade de passagem de tempo será máxima. Assim, a partir da manipulação dos dois switches é possível acelerar a passagem de tempo até chegar no horário desejado, acionando ambos para avançar grandes períodos e apenas um para menores períodos.

- **Modificar base de tempo**

Para modificar a base de tempo o principio é o mesmo do ajuste de horário, os switches vermelho e verde da imagem acima mudam individualmente a velocidade de passagem de tempo, sendo o verde o mais rápido, e acionados juntos mudam a velocidade de passagem de tempo para o valor máximo, passando 24 horas em 2 minutos.

- **Modificar modo de visualização das horas (Base 12 ou 24 hrs)**

Para modificar a base de visualização das horas basta utilizar o último swicth da FPGA, indicado em azul na imagem acima.

Quando ele não está acionado, ele indica as horas na base 12, mostrando no visor a sigla do período do dia (A ou P), como exposto abaixo:

Quando é acionado, ele mostra as horas na base 24, como mostrado abaixo: