UFC - UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE SOBRAL
CURSO DE ENGENHARIA DA COMPUTAÇÃO
PROFESSOR: MARCELO SOUZA

RELATÓRIO 2
VOLTÍMETRO COM PIC18F4550

ALUNO: FRANCILÂNDIO LIMA SERAFIM
MATRÍCULA: 472644

Sobral - CE
2021.1

1 Introdução

O presente trabalho exigiu a implementação de um voltímetro utilizando o microcontrolador PIC18F4550, em que fez-se necessária a integração entre software (MPLABX) e Hardware (Proteus).

Em relação ao Software, foi desenvolvido um programa em C que fazia o controle das entradas e saídas do Microcontrolador no sentido de tratar a entrada em um dos pinos, que é uma tensão, de modo a mostrar a leitura através dos pinos de saída em mili Volts.

Com relação ao Hardware, foram usados um gerador de tensão, um potenciômetro, o PIC18F4550, e um dispositivo contendo quatro displays de sete segmentos. O fluxo de trabalho consistiu na geração de uma tensão sendo a entrada do PIC18F4550 que através do software integrado manipulava a entrada de modo a entregar nos pinos de saída a representação da tensão em mili Volts e daí com os pinos de saída conectados aos displays de sete segmentos, era exibido esse valor nos LEDs.

2 Desenvolvimento

Para a resolução do problema, foi editado um código escrito pelo professor de forma a adaptá-lo. A adaptação consistiu em fazer com que os números nos LEDs fossem dados em mili Volts ao invés da codificação que antes era mostrada. Baseando-se em rotinas de atraso que permitiam a multiplexação das saídas do microcontrolador,foi possível obter a impressão de que todos os LEDs acendem de modo simultâneo, quando na realidade um LED é aceso por vez e em 2 ms um próximo LED é aceso. A rotina responsável por isso está representada na Figura 1.

Onde a parte principal dessa rotina está na função Switch(aux) que é responsável por multiplexar as saídas do PIC18, fazendo com que aux seja atualizada a cada interrupção e atribuindo-a ao índice do vetor digitos[] que guarda os valores de cada algarismo a ser representado nos LEDs.

Cada algarismo do número a ser mostrado é atribuído por uma função na qual é passado por parâmetro o valor inteiro em mV e daí são feitas divisões por 10 e operações com módulo para povoar as posições do vetor digitos[]. A parte do código responsável pelo que foi descrito está representada na Figura 2.

É importante citar a estrutura de dados responsável por fazer a conversão dos números BCD para sete segmentos. Essa estrutura é um vetor que pode ser visto na Figura 3, em que, por exemplo, o índice 0 representa o número 0 que transposto para sete segmentos é representado por $0 \times C0$ em hexa, que em binário é 1100 0000, daí em determinado momento do código tem a instrução PORTD = $\sim$ bcd2seg[dig]; que significa que PORTD recebe o valor de bcd2seg[dig] com os bits invertidos (operação not bit a bit), ou seja, se dig = 0 $\rightarrow$ PORTD = 0011 1111 então os pinos de entrada do componente com LEDs recebe as atribuições A = B = C = D = E = F = 0 e G = DP = 1, portanto são acesos os segmentos de A até F para representar o 0. Esse raciocínio é generalizado para os outros números decimais de 1 até 9.

Nas Figuras 4 e 5 estão representadas a tabela contendo a codificação decimal para hexadecimal para sete segmentos de ânodo comum e as representações dos números nos displays, respectivamente.

Figura 1: Rotina de atraso.

```
void __interrupt(low_priority) rotina_interrup(void){
  static unsigned char aux = 0; // aux indica o display a acionar
  unsigned char dig; // valor a apresentar no display
  static unsigned char conta = 0;

  if (TMR2IF){  // Verifica se é timer 2
   TMR2IF = 0;  // limpa flag.


   PORTC = 0xFF; // desliga todos displays.
   dig = digitos[aux]; // coloca o valor a apresentar em d
   PORTD = ~bcd2seg[dig]; // converte para segmentos e coloca em PORTD

   // Dependendo do valor de aux aciona o display correspondente
   switch (aux) {
    case 0: // display 0
        PORTCbits.RC0 = 0;
        aux = 1; // prepara para acionar display 1 na próxima interrupção
        break;
    case 1: // display 1
        PORTCbits.RC1 = 0;
        aux = 2; // prepara para acionar display 2 na próxima interrupção
        break;
    case 2: // display 2
        PORTCbits.RC2 = 0;
        aux = 3; // prepara para acionar display 3 na próxima interrupção
        break;
    case 3: // display 3
        PORTCbits.RC6 = 0;
        aux = 0; // prepara para acionar display 0 na próxima interrupção
        break;
     }
   }
}
```

Fonte: Próprio autor.

Na Figura 6 é possível observar a parte principal do código onde são feitas algumas configurações com respeito às portas de entrada e saída do microcontrolador, além disso é nessa parte em que trabalha-se o resultado da conversão da entrada analógica para um valor digital por parte do conversor AD. O valor da conversão é armazenado na variável ADRES e atribuído a resultado_AD, daí como os valores são decodificados numa escala de 0 a 1023 que representam tensões de 0 a 5 V, faz-se uma manipulação algébrica para que o valor de resultado_AD esteja em mV, precisamente entre 0 e 5000 mv, daí faz-se a seguinte regra de três:

$$\frac{\text{result}}{\text{resultado\_AD}} = \frac{5000}{1023} \tag{1}$$

$$\text{result} = 4.89 \cdot \text{resultado\_AD} \tag{2}$$

Daí, para evitar trabalhar com números floats, é feita a conversão da variável result

Figura 2: Função dec2digit.

```
void dec2digit(unsigned int n) {
    unsigned char d; //variável auxiliar q recebe os algarismos de n.
    unsigned char *ptr = &digitos[3]; //ponteiro q aponta pra quarta posição do vetor dígitos[].

    // Se a tensão a ser mostrada for 0 os valores de digitos[] são todos nulos.
    if (n == 0) {
        digitos[3] = 0;
        digitos[2] = 0;
        digitos[1] = 0;
        digitos[0] = 0;
        return;
    }

    for (unsigned char i =0;i<4;i++) {
     d = n % 10; //O módulo de n dividido por 10 faz com q d receba o valor atual do algarismo das unidades de n.
     n = n / 10; //O piso de n/10 faz com q n passe a ter um algarismo a menos, o algarismo das unids atual some.
     *ptr = d; //Conteúdo apontado por ptr em digitos[] passa a ser d
     ptr--; //ptr passa a apontar para uma posição de digitos[] cujo índice é uma unid a menos q o anterior.
    }
}
```

Fonte: Próprio autor.

Figura 3: Estrutura BCD para 7 segmentos.

```
// bcd2seg[] tabela de decodificação BCD para sete segmentos
unsigned char bcd2seg[] = { 0xC0,  // 0
                            0xF9,  // 1
                            0xA4,  // 2
                            0xB0,  // 3
                            0x99,  // 4
                            0x92,  // 5
                            0x82,  // 6
                            0xF8,  // 7
                            0x80,  // 8
                            0x90 };// 9
```

Fonte: Próprio autor.

Figura 4: Tabela de conversão.

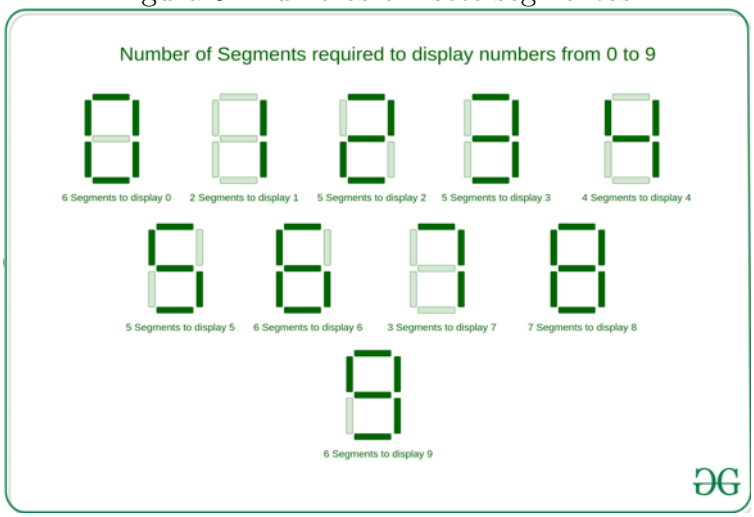
BCD	A	B	C	D	E	F	G	DP	Hexa
	RD0	RD1	RD2	RD3	RD4	RD5	RD6	RD7	
0	0	0	0	0	0	0	1	1	0xC0
1	1	0	0	1	1	1	1	1	0xF9
2	0	0	1	0	0	1	0	1	0xA4
3	0	0	0	0	1	1	0	1	0xB0
4	1	0	0	1	1	0	0	1	0x99
5	0	1	0	0	1	0	0	1	0x92
6	0	1	0	0	0	0	0	1	0x82
7	0	0	0	1	1	1	1	1	0xF8
8	0	0	0	0	0	0	0	1	0x80
9	0	0	0	0	1	0	0	1	0x90

Fonte: Aula de Micros.

através de (int) result e atribuímos seu valor à resultado_AD, atualizando-a.

Após tratar o valor a ser mostrado nos Leds, usamos a função dec2dit() para fazê-lo, como descrito anteriormente.

Figura 5: Números em sete segmentos.



Fonte: Aula de Micros.

Figura 6: Função principal.

```
// Programa principal
void main(void) {
    ADCON1 = 0x0E; // Configura RA0 como sendo
                   //AN0
    ADCON2 = 0x80; // Resultado da conversão
                   //justificado para direita
    ADCON0bits.ADON = 1; // liga conversor A/D

    // inicializações do Timer 2
    T2CONbits.T2CKPS = 0b11;    // PRESCALER 1:16
                                //(16 us)
    T2CONbits.TOUTPS = 0b0000; // POSTSCALER 1:1
    PR2 = 125;       // Valor do estouro do timer
                     //(2 ms)
    TMR2IE = 1; // Ativa interrupção do timer 2
    T2CONbits.TMR2ON = 1; // Ativa contagem
    // Inicializações das portas utilizadas no
    //acionamento dos displays
    PORTC = 0xFF;
    TRISC = 0x00;  // PORTC como saida
    PORTD = 0x00;
    TRISD = 0x00;  // PORTD como saida
    GIE = 1;      // Habilita todas as interrupções
    PEIE = 1;    // Habilita interrupção dos periféricos

    int resultado_AD = 0; // número a ser apresentado.
                          // tem de ser menor que 9999
    while (1) {
     ADCON0bits.GO = 1; // Inicia conversão
     while (ADCON0bits.GO == 1); // Espera terminar
                                 //conversão
     resultado_AD = ADRES; //ADRES guarda o valor da
                           //conversão e esse valor é
                           //atribuído a resultado_AD.

     float result = resultado_AD * (5000/1023); //result
     //vem da regra de 3 onde result está para resultado_AD,
     //assim como 5000 mV está para 1023
     resultado_AD = (int) result; //para trabalhar com inteiros
     dec2digit(resultado_AD);
     __delay_ms(50);
     };
```

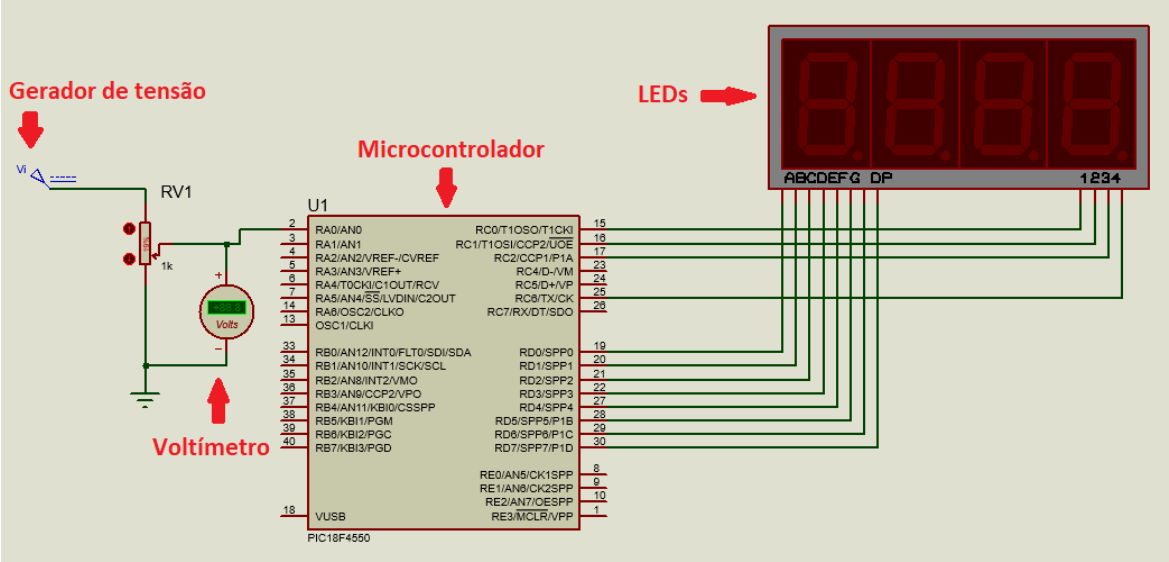
Fonte: Próprio autor.

3 Testes

Tratando-se da parte do hardware, pode-se ver o esquema montado a partir da Figura 7. Nela os componentes gerador de tensão, voltímetro, microcontrolador, e Leds unem suas funções para mostrar a conexão entre software e hardware, pois ao aplicar-se uma tensão de 5 V através do gerador de tensão, esse valor é ajustado pelo potenciômetro que por sua vez é lido no microcontrolador e tratado pelo conversor AD juntamente com as funções programadas via software, e daí a saída do PIC é lida e mostrada nos Leds.

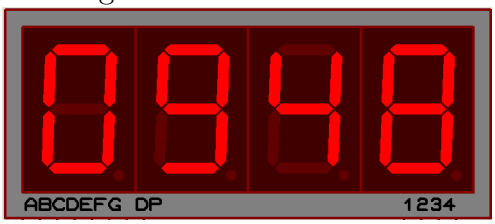
A seguir, nas Figuras 8, 9, 10, 11, 12 e 13 são mostrados os testes feitos, sendo que cada imagem mostra o valor da tensão (em Volts) aplicada na entrada do microcontrolador através do voltímetro e em sua saída (em mili Volts) através dos Leds. A partir da análise entre cada valor de entrada e respectiva saída, é possível concluir que a pequena diferença deve-se ao arredondamento feito da leitura do conversor AD, ao transformar um valor float em inteiro.

Figura 7: Esquema montado.



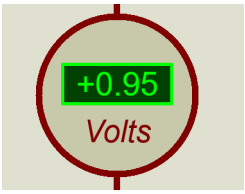
Fonte: Próprio autor.

Figura 8: Leds no teste 1.



Fonte: Próprio autor.

Figura 9: Valor 1 no voltímetro.

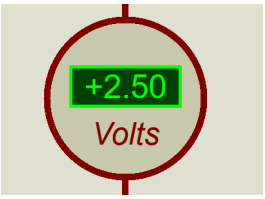


Fonte: Próprio autor.

Figura 10: Leds no teste 2.

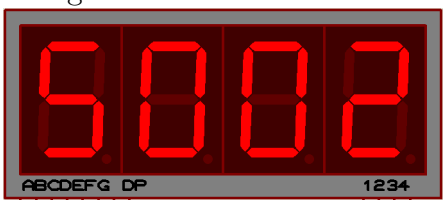
Fonte: Próprio autor.

Figura 11: Valor 2 no voltímetro.



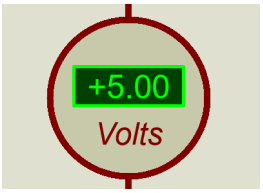
Fonte: Próprio autor.

Figura 12: Leds no teste 3.



Fonte: Próprio autor.

Figura 13: Valor 3 no voltímetro.



Fonte: Próprio autor.

4 Solução alternativa

Para solucionar o problema proposto, há uma outra forma distinta da que foi apresentada com relação a uma parte do programa, mais precisamente na parte de tratamento da saída da conversão AD. Tal solução propõe fazer rotação de bits ao invés de multiplicar ou dividir um número, dado que multiplicar um número inteiro por 2 equivale a deslocar os bits do mesmo para a esquerda e dividí-lo por 2 equivale a deslocar seus bits para a direita. Então aplicou-se o seguinte raciocínio para substituir a equação 1:

$$\frac{\text{result}}{\text{resultado\_AD}} = \frac{5000}{1023} \approx \frac{5000}{1024} \tag{3}$$

$$\text{result} \approx \frac{5000}{1024} \cdot \text{resultado\_AD} \tag{4}$$

$$\text{result} \approx 4 \cdot \text{resultado\_AD} + \text{resultado\_AD} - \frac{\text{resultado\_AD}}{8} + \frac{\text{resultado\_AD}}{128} \quad (5)$$

Ou seja, a equação 5 indica que em relação aos bits da variável resultado_AD, faz-se o seguinte:

- Rotaciona duas vezes para a esquerda;
- Soma com o valor original;
- Subtrai pelo resultado de 3 rotações para a direita;
- Soma com o resultado de 7 rotações para a direita.

Por fim obtém-se o valor da tensão em mili Volts a ser exibida nos Leds e a parte do código modificada encontra-se representada na Figura 14.

Figura 14: Código com rotação de bits.

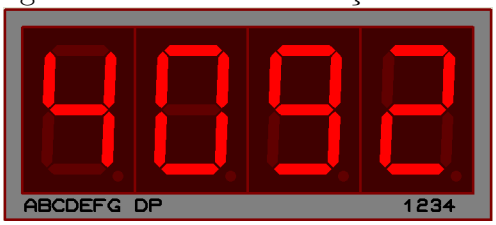
```
while (1) {
 ADCON0bits.GO = 1; // Inicia conversão
 while (ADCON0bits.GO == 1); // Espera terminar conversão
 resultado_AD = ADRES;

 resultado_AD = (resultado_AD << 2)+resultado_AD-(resultado_AD >> 3)+(resultado_AD >> 7);
 //>> rotaciona bits pra direita e << rotaciona pra esquerda
 dec2digit(resultado_AD);
 __delay_ms(50);
 };
}
```

Fonte: Próprio autor

Por fim, na Figura 15 está representado o valor mostrado pelos Leds ao aplicar-se 5 V na entrada do microcontrolador em que o novo código foi anexado.

Figura 15: Teste com rotação de bits.



Fonte: Próprio autor.