UNIVERSIDADE FEDERAL DO PARANÁ

CURSO DE ENGENHARIA ELÉTRICA

MINI CURSO

MICROCONTROLADOR PIC 18F452 / PROTEUS

Orientando:
João Luiz Glovacki Graneman de Melo

Orientador:
Prof. Dr. Gideon Villar Leandro

CURITIBA
2011

SUMÁRIO

1 INTRODUÇÃO

1.1 Linguagem C

Linguagem C foi desenvolvida por Brian Kernighan e Dennis M. Ritchie na década de 70 no AT&T Bell Labs.

Em pouco tempo, esta linguagem teve grande utilização em Universidades espalhadas pelo Mundo, tornando-se praticamente oficial em cursos de engenharias. É uma linguagem estruturada, eficiente, rápida e tão poderosa quanto a Linguagem Assembly. A cada ano vem aumentando o número de programadores de Microcontroladores que acabam migrando da Linguagem Assembly para o C.

Os programas em C se tornam muito mais eficientes e mais rápidos de serem escritos pois os Compiladores de Programas em Linguagem C para Microcontroladores possuem várias bibliotecas de funções prontas como Comunicação Serial, ADC, EEPRON, I2C, PWM, LCD, etc.

Contudo, uma das grandes vantagens de se programar em C é que o programador não precisa se preocupar com o acesso a bancos, localização de memória e periféricos dos Microcontroladores pois o Compilador é responsável por gerenciar esses controles.

Toda essa eficiência da Linguagem C proporciona ao programador preocupar-se apenas com o programa em si e o compilador traduz da Linguagem C para a Linguagem de máquina (.HEX) que é a linguagem que os Microcontroladores conseguem entender.

1.2 Microcontrolador

Um Microcontrolador é um sistema computacional completo inserido em um único circuito integrado. Possui CPU, memória de dados RAM (*Random Access Memory*) e programa ROM (*Read Only Memory*) para manipulação de dados e armazenamento de instruções, sistema de *clock* para dar seqüência às atividades da CPU, portas de I/O além de

outros possíveis periféricos como, módulos de temporização, conversores analógico digital e até mesmo nos mais avançados conversores USB (*Universal Serial Bus*) ou ETHERNET.

Apesar de seu funcionamento exigir uma freqüência de *clock* de alguns MHz, o que é pouco comparado aos microprocessadores modernos, sua utilização é perfeitamente adequada para utilizações típicas. Consomem pouca energia, algo em torno de *miliwatt*s, possuem a capacidade de "hibernar" enquanto aguardam o acontecimento de um evento que o colocará em funcionamento novamente, ideal para circuitos alimentados a baterias químicas pois seu consumo reduz para algo em torno de *nanowatts*. São componentes de baixo custo e compactos.

Para fazer uso do Microcontrolador é necessário desenvolver, além do programa que controla determinado processo, um hardware responsável pela interface entre o mundo externo e o Microcontrolador, adaptando os níveis de tensão e corrente. Porém, para aplicações mais simples e de valores de tensão e corrente próximos aos valores nominais do Microcontrolador pode-se utilizar seus pinos de I/O diretamente interligados ao sistema.

Existe uma grande quantidade de Microcontroladores utilizados em projetos de equipamentos eletroeletrônicos. O Microcontrolador que iremos utilizar será o 18F452 da Microchip. A grande vantagem é que possui memória Flash, possibilitando assim escrever/apagar com grande rapidez.

Abaixo temos principais características do 18F452 utilizado em nosso projeto:

- 40 pinos podendo ter até 34 I/O (*Input/Output digital*);
- 8 canais A/D de 10 Bits;
- 02 Módulos CCP – capture, compare e PWM;
- Memória de Programa *Flash* – 32K
- Memória RAM – 1536 bytes;
- Memória EEPROM – 256 bytes;
- velocidade de processamento – até 10MIPS (milhões de instruções por segundo);
- Módulo MSSP (*Master Synchronous Serial Port*);
- Módulo USART;
- Possibilita até 100.000 ciclos de escrita/leitura na memória de programa;

- Possibilita 1.000.000 de ciclos de escrita/leitura na EEPROM;
- Retenção dos dados na memória por até 40 anos;
- Possibilita habilitação do *Watchdog Timer*;
- Possibilita interrupção externa através de pinos do Microcontrolador;
- 4 Temporizadores/Contadores.

1.3 Compilador MikroC

Existe no mercado vários compiladores para desenvolvimento de programas na Linguagem C como Hi-Tech, CCS, PICmicro C, etc. Adotamos para nosso treinamento o compilador MikroC da Microelektronika por ser bastante poderosa e fácil de trabalhar (permite Editar, Simular e Compilar programas das famílias 12, 16 e 18 da Microchip) além de também possuir uma vasta biblioteca de controle de periféricos dos Microcontroladores.

Além disso, a versão estudante é gratuito para desenvolvimento de programas de até 2Kwords, o que torna bastante atraente também para uso educacional.

Pode-se fazer download em www.mikroe.com.

Neste compilador encontramos uma vasta biblioteca de Funções:
- ADC Library
- CAN Library
- CANSPI Library
- EEPRON Library
- Ethernet Library
- SPI Ethernet Library
- I2C Library
- LCD Library
- One Wire Library
- PWM Library
- PS2 Library
- RS485 Library
- SPI Library

- UART Library
- USB Library

1.4 Simulador - Proteus

O Proteus é um conjunto de ferramentas EDA (*Electronic Design Automation*) reunidas em um único pacote, combinando captura esquemática, ISIS, (*Intelligent Schematic Input System*), simulação, *ProSpice*, (SPICE *Simulation Program with Integrated Emphasis*), layout de PCB, ARES, (*Advanced Routing and Editing Software*) e simulação de projetos eletrônicos baseados em microcontroladores, VSM, (*Virtual System Modelling*).

O ISIS é certamente o destaque do Proteus, pois trabalha em conjunto com a ferramenta de simulação, ProSpice, isto é, construído o esquemático, a partir da mesma tela pode ser feita a simulação.

2 PRINCÍPIO DE PROGRAMAÇÃO

A Álgebra de Boole e seus operadores são muito utilizados em sistemas digitais e também na programação em Linguagens como o *Assembly* e também a Linguagem C.

Temos então as operações E ou AND, OU ou OR e Não ou NOT e também NAND, NOR ou XOR (OR EXCLUSIVO).

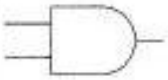
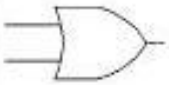
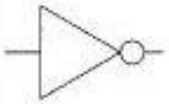
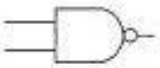
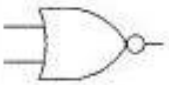
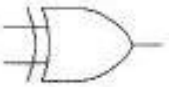
PORTA	Simbologia	Tabela da Verdade	Função Lógica	Expressão
E **AND**		A B S 0 0 0 0 1 0 1 0 0 1 1 1	**Função E:** Assume 1 quando todas as variáveis forem 1 e 0 nos outros casos.	**S=A.B**
OU **OR**		A B S 0 0 0 0 1 1 1 0 1 1 1 1	**Função OU:** Assume 0 quando todas as variáveis forem 0 e 1 nos outros casos.	**S=A+B**
NÃO **NOT**		A S 0 1 1 0	**Função NÃO:** Inverte a variável aplicada à sua entrada.	**S=$\overline{A}$**
NE **NAND**		A B S 0 0 1 0 1 1 1 0 1 1 1 0	**Função NE:** Inverso da função E.	**S=$\overline{(A.B)}$**
NOU **NOR**		A B S 0 0 1 0 1 0 1 0 0 1 1 0	**Função NOU:** Inverso da função OU.	**S=$\overline{(A+B)}$**
OU EXCLUSIVO		A B S 0 0 0 0 1 1 1 0 1 1 1 0	**Função OU Exclusivo:** Assume 1 quando as variáveis assumirem valorem diferentes entre si.	**S=A⊕B** **S=$\overline{A}$.B + A.$\overline{B}$**
COINCIDÊNCIA		A B S 0 0 1 0 1 0 1 0 0 1 1 1	**Função Coincidência:** Assume 1 quando houver coincidência entre os valores das variáveis.	**S= A⊙B** **S= $\overline{A}$.$\overline{B}$ + A.B**

Figura 1 - Blocos Lógicos Básicos

3 INTRODUÇÃO À LINGUAGEM C

3.1 Estrutura Básica de um Programa

Programas em C são baseados em uma ou mais funções que serão executadas, no entanto, a função Main( ) é a primeira a ser executada.

```
/* ------------------------------------------------

abaixo temos um Exemplo de estrutura

Básica de um programa em Linguagem C

------------------------------------------------- */

Main() // esta é a primeira função que será executada

{ // inicializa a função

Trisb=0x00; // aqui entram os comandos que serão executados

Portb=0xFF;

} // finaliza a função
```

Observações:

- ✓ Toda função deve iniciar abrindo chave e finalizar fechando-se a chave.
- ✓ Toda instrução deve ser finalizada com ponto e vírgula (obrigatoriamente)

Logo após /* são inseridos os comentários para múltiplas linhas e deve-se

Colocar */ para fechar o bloco de comentários.

Utilizamos // para comentários em apenas uma linha.

3.2 Representação Numérica

Decimal:

```
Contador=125;
```

Binário:

```
Portb=0b11010011;
```

Hexadecimal:

Variável1=0xA4;

Octal:

Teste=075;

3.3 Tipos de Dados

Tipo	Tamanho – bits	Valor Mínimo	Valor Máximo
Void	Zero	Sem valor	Sem valor
Char	8	-128	127
Int	16	-32768	32767
Short	16	-32768	32767
Long	32	-2^{31}	$2^{31}-1$
Float	32	$-3{,}4 \times 10^{38}$	$3{,}4 \times 10^{38}$
Double	64	$-1{,}8 \times 10^{308}$	$1{,}8 \times 10^{308}$

Modificadores:

Tipo	Tamanho - bits	Valor Mínimo	Valor Máximo
Char	8	-128	127
signed char	8	-128	127
unsigned char	8	0	255
Short int	8	-128	127
signed short int	8	-128	127
unsigned short int	8	0	255
Int	16	-32768	32767
signed	16	-32768	32767
unsigned int	16	0	65535
Short	16	-32768	32767
signed short	16	-32768	32767
unsigned short	16	0	65535
long int	32	-2^{31}	$2^{31}-1$
signed long int	32	-2^{31}	$2^{31}-1$
unsigned long int	32	0	$2^{32}-1$

3.4 Operadores Matemáticos

3.4.1 Aritméticos

Operador	Descrição	Exemplo
+	Soma dos argumentos	a + b
-	Subtração dos argumentos	a – b
*	Multiplicação dos argumentos	a * b
/	Divisão dos argumentos	a / b
%	Resto da divisão	a % b
++	Soma 1 ao argumento (a=a+1)	a++
--	Subtrai 1 ao argumento (a=a-1)	a--

3.4.2 Relacionais

Operador	Descrição
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que
==	Igual
!=	Diferente

3.4.3 Lógicos

Operador	Descrição
&&	Lógica E (AND)
||	Lógica OU (OR)
!	Complemento (NOT)

3.4.4 Operadores Bit a Bit

Operador	Descrição
&	Lógica E (AND)
|	Lógica ou (OR)
^	Lógica OU-Exclusivo
~	Complemento (NOT)
>>	Deslocamento à direita
<<	Deslocamento à esquerda

3.5 Controle de Fluxo

3.5.1 Decisão IF

Sintaxe: if (expressão) comando;

A expressão é avaliada e se for verdadeiro executa o comando.

Podemos ter também mais que um comando:

```
Sintaxe if (expressão)
{
    comando1;
    comando2;
    comandoN;
}
```

Exemplo:

```
If (conta == 5)
{
    a=a++;
    portc=0xFF;
}
```

3.5.2 Decisão IF-ELSE

Sintaxe: if (expressão) comando1;

else comando2;

Neste caso, temos duas possibilidades. Se comando for verdadeiro, comando1 é executado, caso seja falso, comando2 será executado.

Podemos ter também vários comandos:

```
if (expressão)
{
        comando1;
        comando2;
        comando3;
}
else
{
        comando4;
        comando5;
}
```

Exemplo:

```
if (a>22)
{
        Valor1=x;
        y=contador+10;
}
else
{
        Valor2=x;
        Y=contador-5;
}
```

Podemos ainda ter superposição de comandos if-else podendo assim ter uma escolha entre várias possibilidades.

```
if (expressão1)
{
        comando1;
}
else if (expressão2)
{
        comando2;
        comando3;
}
else
{
        comando4;
        comando5;
}
```

Exemplo:

```
if (contador==12)
{
        Dúzia++;
        X++;
}
else if (contador<12)
{
        Dúzia=0;
        y--;
}
else if (contador>12)
{
```

```
        Dúzia = 0;
        Z++;
}
```

3.5.3 Decisão SWITCH - CASE

```
switch (variável)
{
        case valor1:
                comando1;
                comando2;
        break;
        case valor2:
                coman do3;
                comando4;
                comando5;
        break;
        default: //opcional
        comando6;
}
```

Exemplo:

```
void main( )
{
         int contagem=4;
        int valor=5;
        switch (contagem)
        {
                case 2:
```

```
                valor++;
            break;
            case 5:
                valor--;
            break;
                case 10:
                valor=0;
                break;
            default:
                valor=5;
        }
}
```

3.5.4 Loop FOR

Este é um comando de laço (loop ou repetição).

Sintaxe:

for (inicialização; condição (término); incremento) comando;

Podemos também ter um bloco de comandos:

```
For (inicialização; condição (término); incremento)
{
    comando1;
    comando2;
    comandoN;
}
```

Exemplo:

```
void main ()
{
    int contador;
    int a = 0;
    for (contador=0; contador<=10; contador++) a=a+contador;
}
```

3.5.5 Loop While

Neste caso, o loop é repetido enquanto a expressão for verdadeira.

Sintaxe:

```
while (expressão)
{
    comando1;
    comando2;
}
```

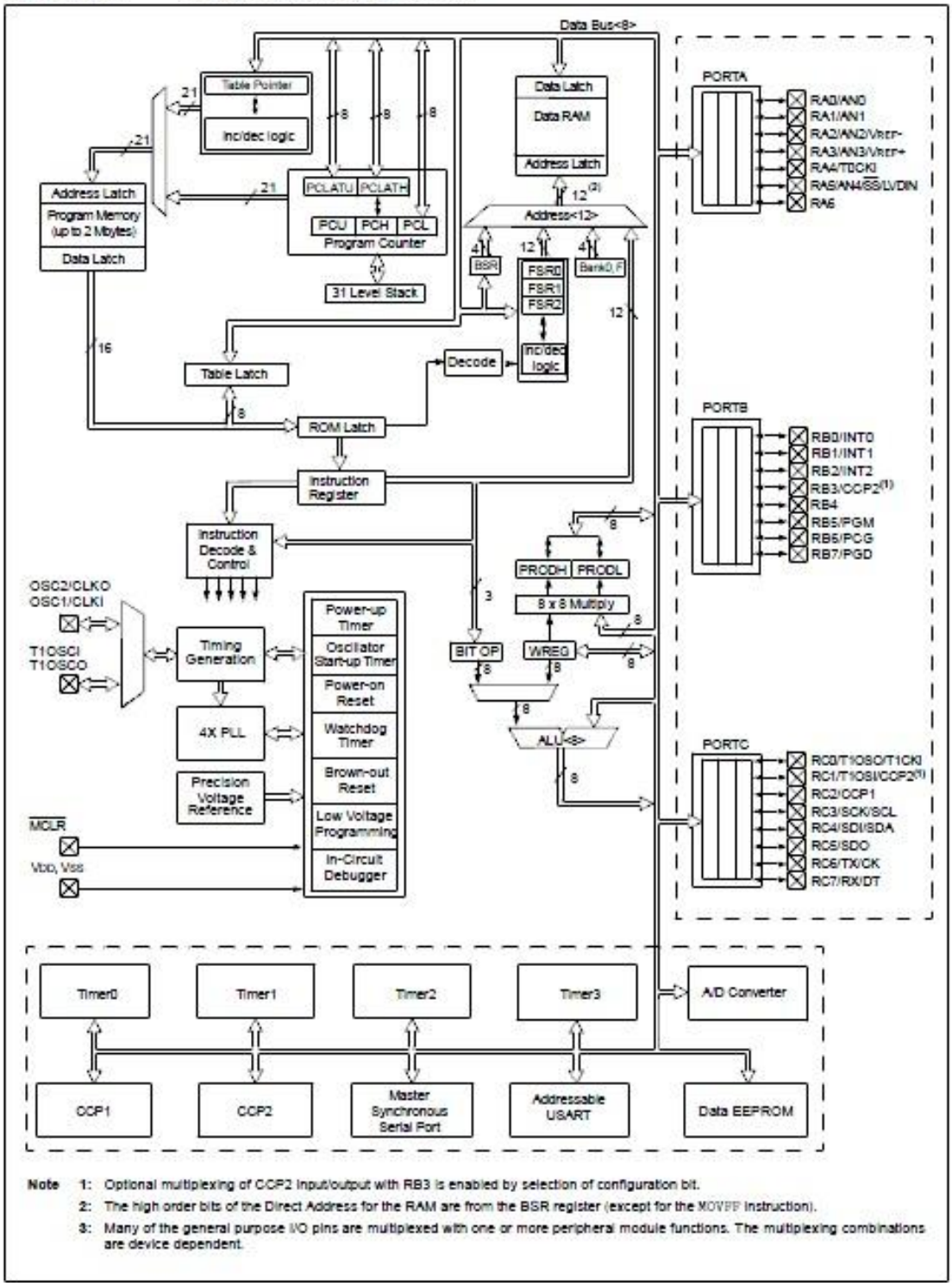
É feita a avaliação no início do loop e, enquanto verdadeira, os comandos serão executados. Ao término do último comando, volta a ser testada a expressão e caso seja falsa, o loop é finalizado.

Exemplo:

```
void main( )
{
    int a=15;
    while (a>10)
    {
        a--;
        delay_ms(100);
    }
}
```

4 ESTRUTURA INTERNA DO PIC 18F452

FIGURE 1-1: PIC18F2X2 BLOCK DIAGRAM



Note 1: Optional multiplexing of CCP2 input/output with RB3 is enabled by selection of configuration bit.
2: The high order bits of the Direct Address for the RAM are from the BSR register (except for the MOVFF instruction).
3: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

Figura 2 - Diagrama de Blocos do PIC 18F452

5 FUNÇÃO DE CADA PINO DO 18F452

O PIC 18F452 possui cinco PORT's: PORTA, PORTB, PORTC, PORTD e PORTE. Cada PORT possui pinos com acesso aos periféricos como Conversor Analógico/Digital, Interrupções, I2C, UART, Módulo CCP, SPI, ou podem ser utilizados como I/O de uso geral.

Tabela 1 - Descrição da Pinagem do Microcontrolador

PINO	FUNÇÃO	TIPO	FUNCIONALIDADE
1	/MCLR / VPP	In-In	Reset externo e programação ICSP
2	RA0 / AN0	I/O e input A/D	I/O digital e entrada do AD0
3	RA1 / AN1	I/O e input A/D	I/O digital e entrada do AD1
4	RA2 / AN2/ Vref-	I/O e input A/D	I/O digital e entrada do AD1
5	RA3/AN3/ Vref+	I/O e input A/D	I/O digital, entrada do AD3 e entrada de referência alta do A/D
6	RA4 / **T0CKI**	I/O e Input TMR0	I/O digital e entrada do TMR0
7	RA5/AN4/SS/LVDIN	I/O e Inputs	I/O digital, entrada do AD4, entrada do SPI e Detector de LV
8	RE0 / RD / AN5	Fonte	I/O digital, Leitura da Porta Paralela e entrada do AD5
9	RE1 / WR / AN6	Fonte	I/O digital, Escrita da Porta Paralela e entrada do AD6
10	RE2 / CS / AN7	Fonte	I/O digital, Seleção da Porta Paralela e entrada do AD7
11,32	VCC	Fonte	Positivo da Fonte de Alimentação
12,31	GND	Fonte	Negativo da Fonte de Alimentação
13	OSC1 / CLK1	Input	Entrada do Cristal e entrada do Clock externo
14	Osc2 / CLK1 / **RA6**	I/O e Inputs	I/O digital, Saída do Cristal e saída do Clock externo
15	RC0/T10S0/T1Ck1	I/O Out e In	I/O digital, saída do 2° oscilador e entrada do contador externo Timer1/Timer3
16	RC1/T10S1/**CCP2**	I/O In e Out	I/O digital, entrada do 2° oscilador e saída do Módulo CCP2
17	RC2 / **CCP1**	I/O e Out	I/O digital e saída do Módulo CCP1
18	Rc3 / SCK / SCL	I/O, I/O e I/O	I/O digital, in e out do Clock serial para modo SPI e in/out do Clock serial para modo I^2C
19	RD0 / PSP0	I/O e I/O	I/O digital e Porta de Comunicação Paralela
20	RD1 / PSP1	I/O e I/O	I/O digital e Porta de Comunicação Paralela
21	RD2 / PSP2	I/O e I/O	I/O digital e Porta de Comunicação Paralela
22	RD2 / PSP3	I/O e I/O	I/O digital e Porta de Comunicação Paralela
23	RC4 / SDI / SDA	I/O e I/O	I/O digital e Porta de Comunicação Paralela
24	RC5 / SD0	I/O e I/O	I/O digital e saída de dados SPI
25	RC6 / TX / CK	I/O e I/O	I/O digital, Transmissão UART e Clock de sincronismo UART
26	RC7 / RX / DT	I/O e I/O	I/O digital, Recepção UART e Dados do UART
27	RD4 / PSP4	I/O e I/O	I/O digital e Porta de Comunicação Paralela
28	RD5 / PSP5	I/O e I/O	I/O digital e Porta de Comunicação Paralela
29	RD6 / PSP6	I/O e I/O	I/O digital e Porta de Comunicação Paralela
30	RD7 / PSP7	I/O e I/O	I/O digital e Porta de Comunicação Paralela
33	RD0 / **INT0**	I/O e In	I/O digital e entrada de Interrupção Externa 0
34	RD0 / **INT1**	I/O e In	I/O digital e entrada de Interrupção Externa 1
35	RD0 / **INT2**	I/O e In	I/O digital e entrada de Interrupção Externa 2
36	RB3 / **CCP2**	I/O e I/O	I/O digital Módulo CCP2
37	RB4	I/O e In	I/O digital e entrada de Interrupção por Mudança de Estado
38	RB5 / PGM	I/O e In	I/O digital, Interrupção por Mudança de Estado e Habilita ICSP baixa tensão
39	RB6 / PGC	I/O e In	I/O digital, Interrupção por Mudança de Estado e ICSP in-circuit Debuger
40	RB7 / PGD	I/O e In	I/O digital, Interrupção por Mudança de Estado e ICSP in-circuit Debuger

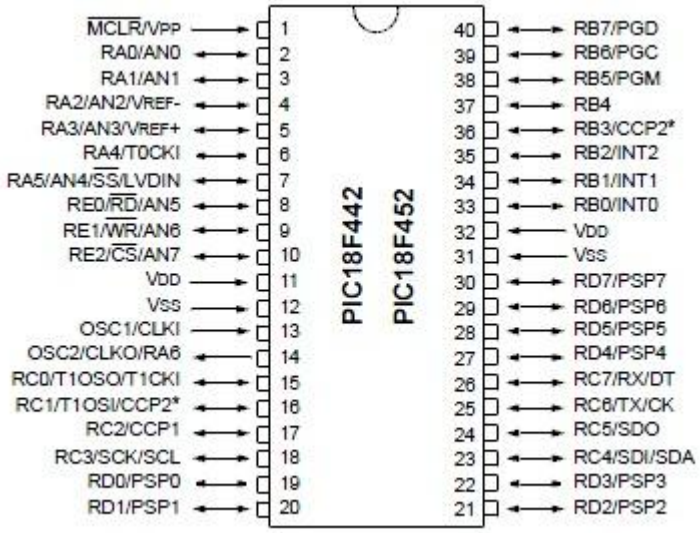


Figura 3 - Pinagem do PIC 18f452

Nos Microcontroladores PIC temos os GPR (General Porpose Registers) que são os registradores de usos gerais que nada mais são do que variáveis criadas pelo programador – dados voláteis.

Temos também os SFR(Special Funcion Registers) que são os Registradores de uso específico que na verdade contém o SETUP do Microcontrolador – configura como irão trabalhar determinados periféricos como PWM, Conversores A/D, USART, etc.

O Microcontrolador 18F452 possui memória RAM de 1536 bytes sendo dividido em 16 bancos de 256 bytes de memória cada um

6 MAPA DE REGISTRADORES ESPECIAIS – SFRS

TABLE 4-1: SPECIAL FUNCTION REGISTER MAP

Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2(3)	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2(3)	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2(3)	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2(3)	FBCh	CCPR2H	F9Ch	—
FFBh	PCLATU	FDBh	PLUSW2(3)	FBBh	CCPR2L	F9Bh	—
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—
FF9h	PCL	FD9h	FSR2L	FB9h	—	F99h	—
FF8h	TBLPTRU	FD8h	STATUS	FB8h	—	F98h	—
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	—	F97h	—
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	—	F96h	TRISE(2)
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD(2)
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	LVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—
FF0h	INTCON3	FD0h	RCON	FB0h	—	F90h	—
FEFh	INDF0(3)	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—
FEEh	POSTINC0(3)	FCEh	TMR1L	FAEh	RCREG	F8Eh	—
FEDh	POSTDEC0(3)	FCDh	T1CON	FADh	TXREG	F8Dh	LATE(2)
FECh	PREINC0(3)	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD(2)
FEBh	PLUSW0(3)	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	—	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—
FE7h	INDF1(3)	FC7h	SSPSTAT	FA7h	EECON2	F87h	—
FE6h	POSTINC1(3)	FC6h	SSPCON1	FA6h	EECON1	F86h	—
FE5h	POSTDEC1(3)	FC5h	SSPCON2	FA5h	—	F85h	—
FE4h	PREINC1(3)	FC4h	ADRESH	FA4h	—	F84h	PORTE(2)
FE3h	PLUSW1(3)	FC3h	ADRESL	FA3h	—	F83h	PORTD(2)
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	—	FA0h	PIE2	F80h	PORTA

Note 1: Unimplemented registers are read as '0'.
2: This register is not available on PIC18F2X2 devices.
3: This is not a physical register.

Figura 4 - Mapa dos Registradores Especiais

Temos então: PORTA à PORTE e cada pino de cada porta pode ser configurado como entrada ou saída. No entanto, para podermos configurar como irá trabalhar cada pino de cada porta, temos que fazer uso do SFR – TRIS.

O Registrador TRISA é responsável pela configuração do PORTA, o TRISB pelo PORTB, o TRISC pelo PORTC, o TRISD pelo PORTD e o TRISE pelo PORTE.

É muito simples, se colocarmos **0** no bit do TRIS estamos definindo determinado pino como **SAÍDA** e colocando-se **1** no bit do TRIS estamos definindo o pino do PORT como **ENTRADA**.

Vejamos um **Exemplo**:

TRISB=0b00001100;

Neste caso temos:

RB0=saída

RB1=saída

RB2=entrada

RB3=entrada

RB4 à RB7 = saída

Outro **Exemplo**:

TRISD=0b11111111;

Neste caso temos todo o PORTD configurado como entrada

Outro **Exemplo**:

TRISC=0b00000000;

Neste caso temos todo o PORTC configurado como saída

7 DEVICE FLAGS

É o local onde podemos configurar várias funções através de bits de configuração.

Funções estas que podem ser tipo de oscilador, habilitar *Watchdog*, Código de proteção, etc.

5.1 Clock

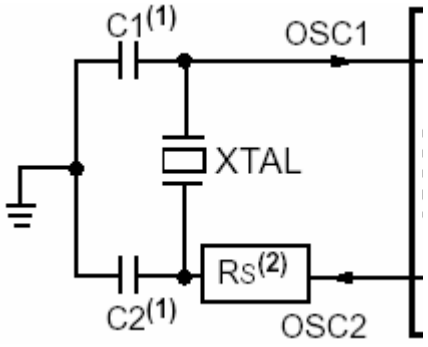


Figura 5 - Configuração do Cristal de Clock

Resistor RS pode ser necessário para melhor funcionamento do cristal.

O PIC 18F452 pode funcionar com os seguintes tipos de clock:

LP - cristal de baixa freqüência

XT - cristal ou ressonador

HS - cristal ou ressonado de alta freqüência

HS + PLL - Cristal ou ressonador com PLL habilitado

RC - Resistor e capacitor externo

RCIO - Resistor e capacitor externo e liberação de 1 pino de I/O

EC - Clock externo

ECIO - Clock externo e liberação de 1 pino de I/O

Valores de capacitores que devem ser acoplado ao cristal:

Mode	Freq	C1	C2
LP	32.0 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	22-68 pF	22-68 pF
	1.0 MHz	15 pF	15 pF
	4.0 MHz	15 pF	15 pF
HS	4.0 MHz	15 pF	15 pF
	8.0 MHz	15-33 pF	15-33 pF
	20.0 MHz	15-33 pF	15-33 pF
	25.0 MHz	15-33 pF	15-33 pF

Figura 6 - Valores dos Capacitores

5.2 PWRTEN (system clock switch bit)

É o temporizador de Power-Up que faz o microcontrolador aguardar certo tempo assim que o chip é energizado. Este tempo é de 72ms deixando assim o microcontrolador inoperante, tempo ideal para que o circuito oscilador estabilize sua freqüência.

5.3 BROW-OUT detector

É um interessante circuito para *resetar* o Microcontrolador caso ocorra uma queda de tensão no mesmo. Restabelecido a tensão, o programa é reiniciado. Podemos escolher os seguintes limites de tensão: 2,0V, 2,7V, 4,2V ou 4,5V.

5.4 BOREN

Este bit é responsável por habilitar/desabilitar o *Brown-out*.

5.5 WDTEN – Watchdog Timer Enable

Aqui temos um recurso bastante interessante. Um temporizador de 8 bits que não pode estourar pois caso isso ocorra o programa será *resetado*. Imagina que por algum motivo o programa trava ou entra em algum loop infinito. A função do *watchdog* é não deixar o programa travado.

5.5.1 Watchdog postscale select bit

O tempo padrão de estouro do *watchdog* é 18ms, porém, podemos modificar este tempo modificando o fator de divisão: 1:1, 1:2, 1:4, 1:8, 1:16, 1:32, 1:64 e 1:128.

5.6 LVP (low voltage programming)

Temos aqui uma opção de poder gravar o Microcontrolador com 5V. O usual é colocar tensão de 13,5V no pino MCLR. Portanto, é possível deixar o MCLR em 5V e fazer a gravação mas, o pino RA5 não poderá mais ser utilizado como I/O.

5.7 CCP2MX

Este bit tem a função de Multiplexar o módulo CCP2.

Para CCP2MX=0 usaremos pino RC1 para CCP2

Para CCP2MX=1 usaremos pino RB3 para CCP2

5.8 STVREN (Stack Full / Underflow Reset Enable Bit)

O Microcontrolador 18F452 possui 31 endereços de pilha. Este bit serve para habilitar o chip para ser resetado caso a limite da pilha seja ultrapassado.

5.9 BKBUG (Debug ou Depuração)

Este bit habilita o modo *Debugger* do PIC. Com isso pode-se emular o programa desenvolvido na placa de testes. No entanto, esta função ativa faz com que os pinos RB6 e RB7 deixam de funcionar como I/O.

8 PROJETOS

6.1 Pisca-Pisca

```
/*****************************************************************
Linguagem C – Compilador MikroC
Programa 1
Objetivo: Configuração das Portas e acesso das mesmas usando funções de tempo
*****************************************************************/
void main ( )
{
    trisd = 0; //configura todo o portd como saída
    portd = 0; // todo o portd é colocado em nível zero
    while(1) //loop infinito
    {
      portd.f0=1; //ativa saída RD0
      delay_ms(5000); //aguarda 5 segundos
      portd.f0=0; //desativa saída RD0
      delay_ms(1000 ); //aguarda 1 segundo
      }}
```

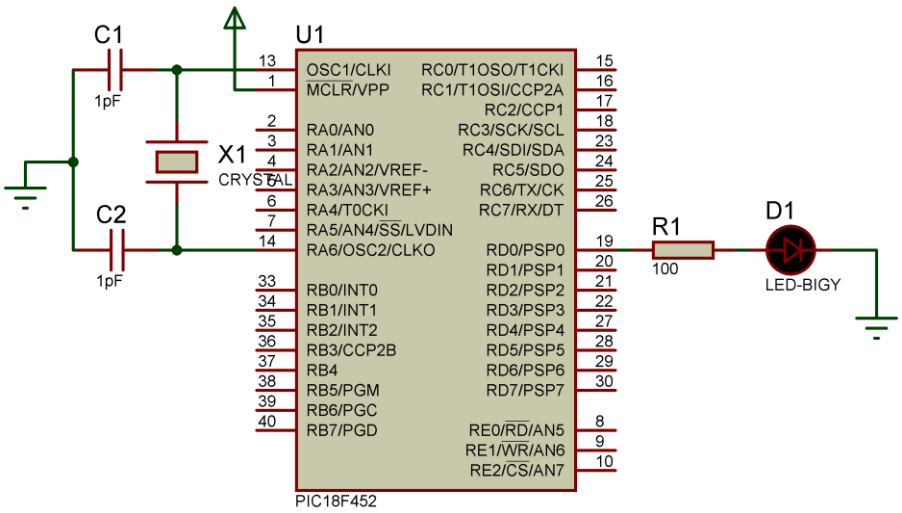


Figura 7 - Esquemático Programa 1

6.2 Acionamento de Botões

```
/*********************************************************************
Linguagem C – Compilador MikroC
Programa 2
Objetivo: Fazer leitura de um botão e utilização do IF-ELSE
*********************************************************************/
void main ( )
{
   trisb = 0b00000001; // configura RA0 como entrada e o restante como saída
   trisd = 0b00000000; // configura PORTD como saida
   portb = 0; // todos os pinos de saída do porta = nível baixo
   portd = 0; // todos os pinos de saída do portd = nível baixo
   while(1) //loop infinito
   {
     if (portb.f0==1) //testa se RB0 está em nível alto
     portd = 0b10101010; //
     else
     portd = 0b01010101; //
   }
}
```

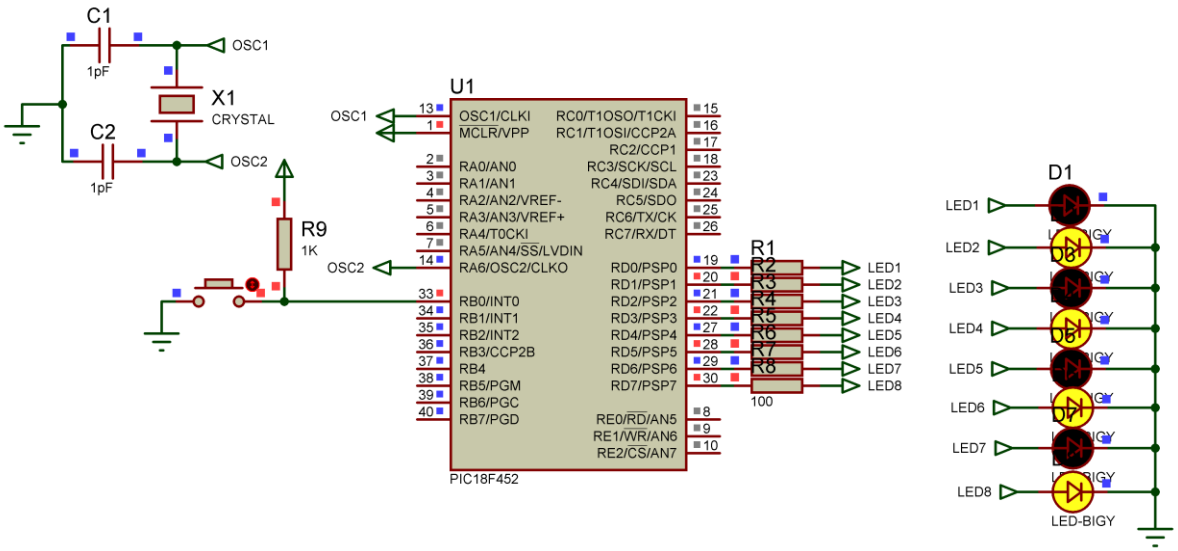


Figura 8 - Esquemático Programa 2

6.3 Acionamento de Botões Utilizando IF-ELSE-IF

```
/*****************************************************************************
Linguagem C – Compilador MikroC
Programa 3
Objetivo: Fazer leitura de vários botões e verificar a utilização do IF-ELSE-IF
*****************************************************************************/
void main()
{
    trisd=0; // define todo o portd como saída
    trisb=0b11111111; // define todos os pinos do portd como entrada
    portd=0; // coloca portd em 0
    while(1)
    {
        if (portb.f0==0) // RB0 esta presionado? se nao, pula para testar RB1
          portd.f0=1; // se pressionado, entao aciona RD0
        else
        if (portb.f1==0)
          portd.f1=1;
        else
        if (portb.f2==0)
          portd.f2=1;
        else
        if (portb.f3==0)
          portd.f3=1;
        else
```

```
            if (portb.f4==0)
              portd.f4=1;
            else
            if (portb.f5==0)
              portd.f5=1;
            else
            if (portb.f6==0)
              portd.f6=1;
            else
            if (portb.f7==0)
              portd.f7=1;
        }
    }
```

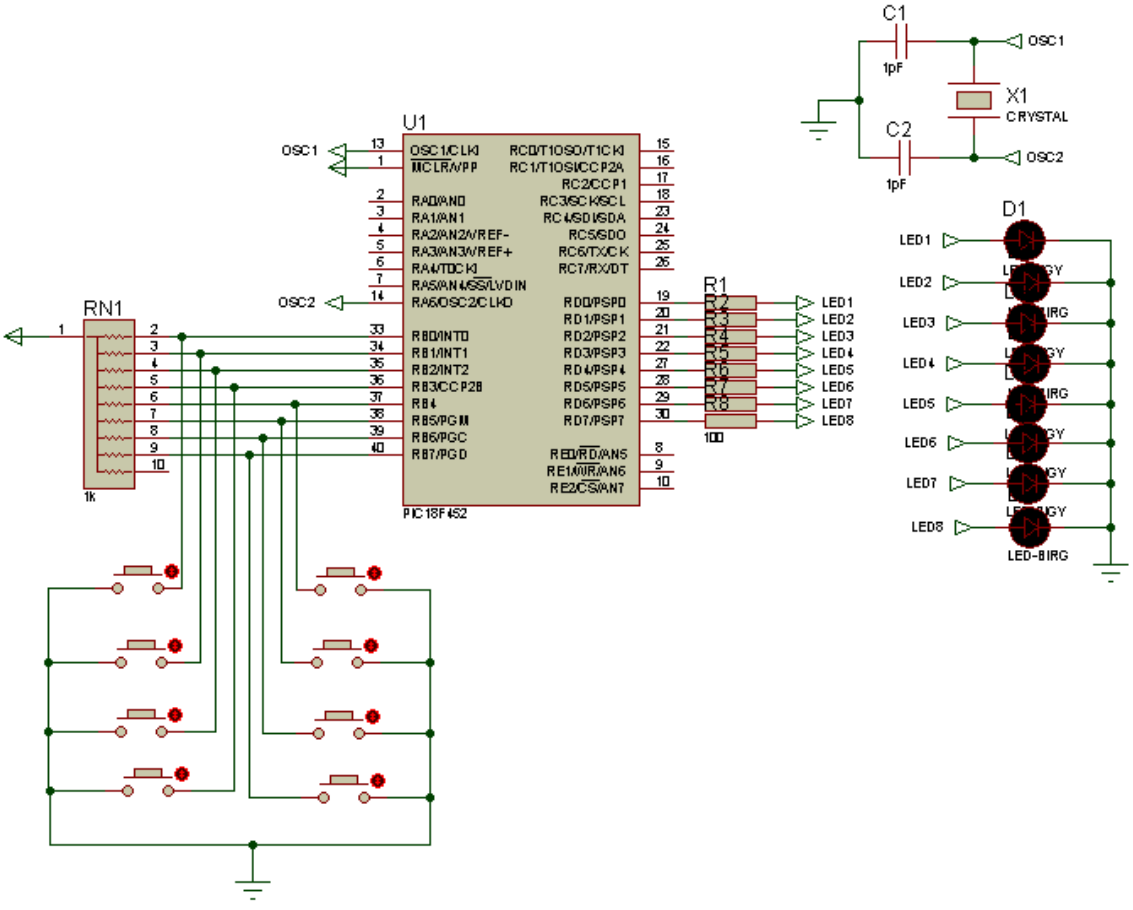


Figura 9 - Esquemático Programa 3

6.4 Função Button

Button(&portX, pinoX, tempoX, Estado_Tecla)

Onde:

&portX = port da tecla (porta, portb, portc, portd ou porte);

pinoX = é o pino onde está conectado a tecla (varia de 0 à 7);

tempoX = tempo de Debounce em milisegundos;

Estado_Tecla = Valor do nível lógico quando a tecla é pressionada.

```
/*****************************************************************
Linguagem C – Compilador MikroC
Programa 4
Objetivo: Solucionar o problema de ruído (Debounce) e também aguardar soltar o botão.
*****************************************************************/
void main()
{
    trisd=0; // define todo o portd como saída
    trisb=0b11111111; // define portb como entrada
    portd=0; // coloca portd em 0
    while(1)
    {
          if (button(&portb, 0, 20, 0)) // &port identifica o port, 0 identifica o pino,
          // 20ms é o tempo e 1 é nível para tecla acionada
          {
            portd.f0=~portd.f0; // se pressionado, então aciona RD0
            while(portb.f0==0); // aguarda soltar o botão
            {
            }
          }
    }
}
```

Esquemático igual aos do item 6.4.

6.5 Display de Cristal Líquido – LCD

Esta é um dos recursos mais interessantes quando utilizamos Microcontroladores. Isto porque podemos implementar projetos muito interessantes onde pode-se desenvolver uma IHM (Interface Homem Máquina) agregando valor muito interessante ao projeto.

Utilizaremos um Display alfanumérico 16x2 (dezesseis caracteres e duas linhas). O interessante é que o display já possui internamente outro microcontrolador que gerencia as funções do mesmo. Isto facilita bastante o desenvolvimento do projeto pois, o programador se preocupa apenas em enviar os dados e a localização onde quer que os mesmos apareçam no display.

Descrição da pinagem do display:

Pino	Nome	Função
1	Vss	Terra
2	Vdd	Positivo (normalmente 5V)
3	Vo	Contraste do LCD. Às vezes também é chamado de Vee
4	RS	Register Select
5	R/W	Read/Write
6	E	Enable
7	D0	Bit 0 do dado a ser escrito no LCD (ou lido dele).
8	D1	Bit 1 do dado a ser escrito no LCD (ou lido dele).
9	D2	Bit 2 do dado a ser escrito no LCD (ou lido dele).
10	D3	Bit 3 do dado a ser escrito no LCD (ou lido dele).
11	D4	Bit 4 do dado a ser escrito no LCD (ou lido dele).
12	D5	Bit 5 do dado a ser escrito no LCD (ou lido dele).
13	D6	Bit 6 do dado a ser escrito no LCD (ou lido dele).
14	D7	Bit 7 do dado a ser escrito no LCD (ou lido dele).
15	A	Anodo do back-light (se existir back-light).
16	K	Catodo do back-light (se existir back-light).

Figura 10 - Pinagem do Display LCD 16x2

Comandos disponíveis:

_LCD_FIRST_ROW - Move o cursor para a linha 1
_LCD_SECOND_ROW - Move o cursor para a linha 2
_LCD_CLEAR – Limpa o visor
_LCD_RETURN_HOME - Cursor Retorna à posição inicial
_LCD_UNDERLINE_ON - Underline sobre cursor
_LCD_MOVE_CURSOR_LEFT - Cursor Mover para a esquerda sem alterar exibir dados RAM

_LCD_MOVE_CURSOR_RIGHT - Mover para a direita do cursor sem alterar exibir dados RAM
_LCD_TURN_ON - Liga tela LCD
_LCD_TURN_OFF - Desliga telar LCD

No MikroC podemos utilizar um LCD em dois modos:

- Modo 4 bits

- Modo 8 bits

Quando utilizamos o Modo 8 bits, utilizamos um port para envio de dados e, para os sinais de controle utilizamos alguns pinos de outro port.

Por exemplo: RD0 a RD7 para dados e RB7, RB6 e RB5 para sinais E, R/W e R/S.

Quando utilizamos o Modo 4 bits, podemos utilizar somente um port para dados e controle ou utilizar dois ports, um para dados e outro para controle.

```
/**************************************************************************
Linguagem C – Compilador MikroC
Programa 5
Objetivo: inicializar o módulo e apresentar uma mensagem no mesmo
**************************************************************************/

// Conexões LCD
sbit LCD_EN at RD0_bit;
sbit LCD_RS at RD1_bit;
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;

sbit LCD_RS_Direction at TRISD1_bit;
sbit LCD_EN_Direction at TRISD0_bit;
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;
// End LCD module connections

char txt1[] = "Microcontrolador";
char txt2[] = "18F452";
char txt3[] = "LCD 4 bits";
char txt4[] = "Exemplo";

char i;                         // Variável de Loop
```

```
void Move_Delay() {                // Função usada para movimento do Texto
  Delay_ms(500);                   // Velocidade do Movimento
}

void main(){

  trisd=0;

  Lcd_Init();                      // Initializa LCD

  Lcd_Cmd(_LCD_CLEAR);             // Clear display
  Lcd_Cmd(_LCD_CURSOR_OFF);        // Cursor off
  Lcd_Out(1,5,txt3);               // Escreve texto na 1° linha

  Lcd_Out(2,6,txt4);               // Escreve texto na 2° linha
  Delay_ms(2000);
  Lcd_Cmd(_LCD_CLEAR);             // Clear display

  Lcd_Out(1,1,txt1);               // Escreve texto na 1° linha
  Lcd_Out(2,6,txt2);               // Escreve texto na 2° linha

  Delay_ms(2000);

  // Movimento do Texto
  for(i=0; i<4; i++) {             // Move o texto para a direita 4x
    Lcd_Cmd(_LCD_SHIFT_RIGHT);
    Move_Delay();
  }

  while(1) {                       //  loop
    for(i=0; i<8; i++) {           // Move o texto para a esquerda 7x
      Lcd_Cmd(_LCD_SHIFT_LEFT);
      Move_Delay();
    }

    for(i=0; i<8; i++) {           // Move o texto para a direita 7x
      Lcd_Cmd(_LCD_SHIFT_RIGHT);
      Move_Delay();
    }
  }
}
```

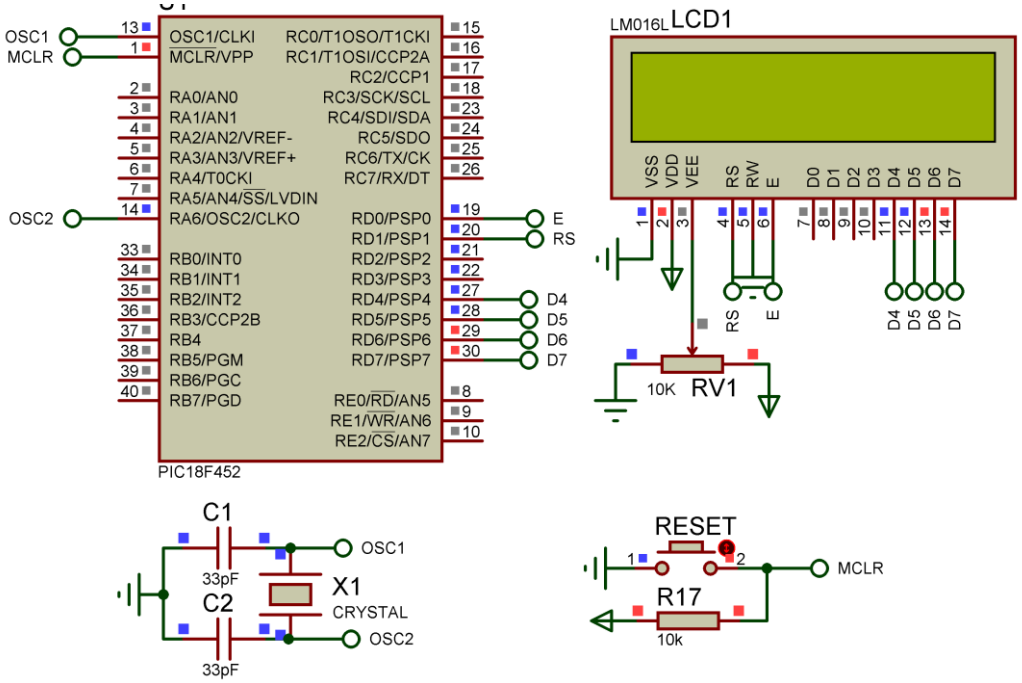


Figura 11 - Esquemático LCD

6.6 *Módulo PWM (Pulse Width Modulation)*

Encontramos uso do mesmo em Inversores de freqüência, controle de servo motores, Fontes chaveadas, controle de potência e velocidade de motores, etc.

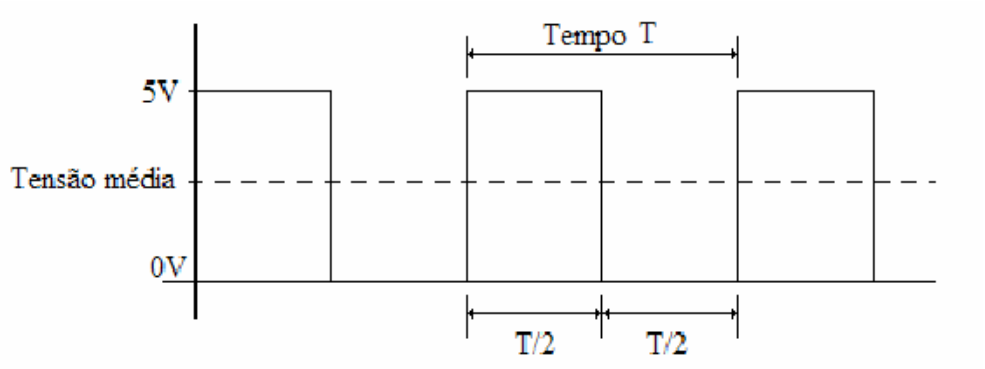


Figura 12 – Duty-Cycle

Conforme figura acima, uma onda pulsante, com amplitude 5V é acionada/desacionada no tempo. A tensão média varia em função do tempo em que o sinal fica no nível alto (5V) e do tempo em que a onda fica no nível baixo (0V).

Para a figura acima, como 50% do tempo a onda fica em 5V e 50% em 0V; obtemos neste caso uma tensão média de 2,5V.

A relação entre o tempo em que a onda fica em nível alto e o período total é chamado de *Duty-cycle* ou ciclo de trabalho. Portanto, para um *duty-cycle* de 50% temos metade do tempo com a onda em VCC e a outra metade em 0V. Isso implica em uma tensão média de 2,5V, o que significa que se colocarmos uma fonte contínua com 2,5V produz o mesmo efeito que a fonte pulsante de 5V com *duty-cycle* de 50%.

O PIC 18F452 possui 2 módulos CCP – CCP1 e CCP2 (Capture/Compare/PWM)

Os Comandos abaixo são para o Módulo CCP1:

PWM1_Init ( ) - X é o valor da freqüência em Hz;

PWM1_Set_Duty (Y) - Y é valor tipo char entre 0 e 255

Valor 0 e 255 determina a porcentagem em que está operando o Módulo.

Exemplo: para *duty-cycle* de 50% basta colocar valor de Y=127.

PWM1_Start ( )- Inicia o Módulo;

PWM1_Stop ( ) - Finaliza o módulo PWM

```
/************************************************************************
Linguagem C – Compilador MikroC
Programa 6
Objetivo: através de botões, modificar o duty-cycle dos Módulos CCP1 e CCP2
************************************************************************/
unsigned short int duty_cycle1=127;
unsigned short int duty_cycle2=30;
void seta()
{
  PWM1_Init(1000);
  PWM1_Set_Duty(duty_cycle1);       // Set current duty for PWM1
  PWM1_Start();

  PWM2_Init(1000);
  PWM2_Set_Duty(duty_cycle2);
  PWM2_Start();
}
void main()
{
    seta();
    while(1)
    {
    if (button(&portd, 0,20,1))
```

```
    {
        duty_cycle1++ ;
        delay_ms(50);
        seta();
    }
    if (button(&portd, 1,20,1))
    {
        duty_cycle1--;
        delay_ms(50);
        seta();
     }
    if (button(&portd, 2,20,1))
    {
        duty_cycle2++ ;
        delay_ms(50);
        seta();
    }
    if (button(&portd, 3,20,1))
    {
        duty_cycle2--;
        delay_ms(50);
        seta();
    }
    }
}
```

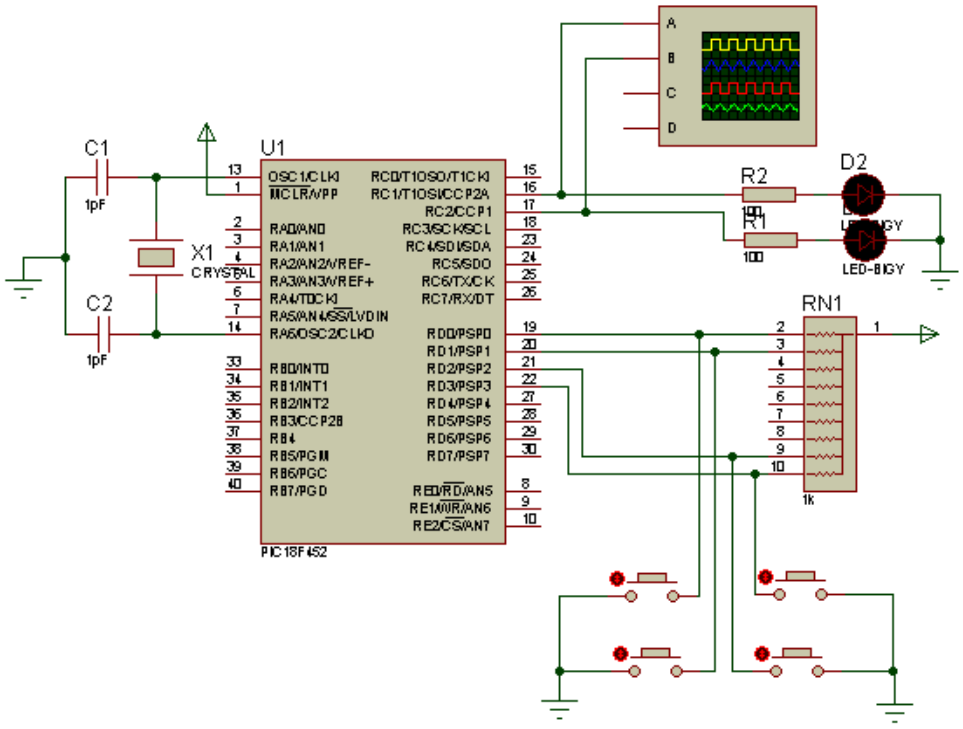


Figura 13 - Esquemático Programa 6

6.7 Conversor A/D – Analógico/Digital

Este é um dos periféricos mais importantes do Microcontrolador, pois através dele podemos fazer leitura de grandezas físicas como temperatura, pressão, humidade, etc.

O PIC 18F452 possui 8 canais com resolução de 10 bits ou seja, para um sinal de 5Vpoderemos fazer leitura de 4,8876mV.

As 8 entradas são multiplexadas ou seja, na verdade temos apenas 1 conversor de 10 bits e 8 entradas disponíveis para serem selecionadas pelo MUX e obter a conversão

Na figura abaixo temos o circuito interno do conversor A/D no PIC:

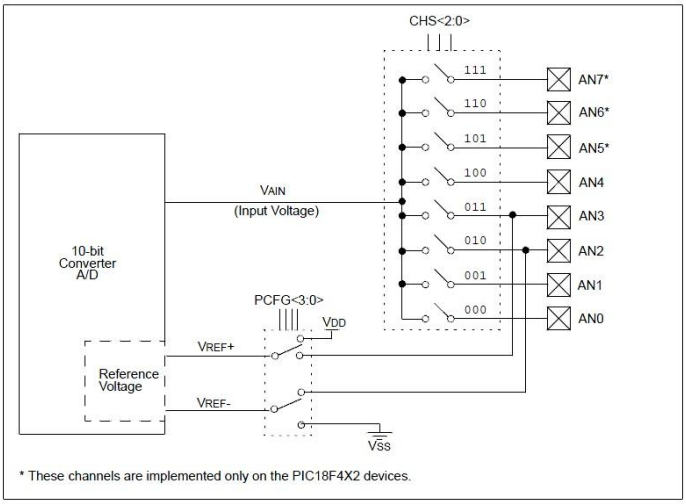


Figura 14 - Diagrama de Blocos Conversor AD

Estas entradas estão disponíveis nos seguintes pinos:

RA0 – Entrada Analógica 0 = AN0

RA1 – Entrada Analógica 1 = AN1

RA2 – Entrada Analógica 2 = AN2

RA3 – Entrada Analógica 3 = AN3

RA5 – Entrada Analógica 4 = AN4

RE0 – Entrada Analógica 5 = AN5

RE1 – Entrada Analógica 6 = AN6

RE2 – Entrada Analógica 7 = AN7

Os Registradores que configuram como irá trabalhar o Conversor A/D são:

- ADCON0

- ADCON1

Como os registradores são de 8 bits e a resolução do A/D é de 10 bits, teremos então 2 registradores para receberem os dados da conversão:

- ADRESH
- ADRESL

Com isso, poderemos fazer é a justificativa à direita ou justificativa à esquerda dos bits mais/menos significativos. Isso funciona da seguinte maneira:

Caso seja configurado para justificativa à direita, os 8 bits menos significativos ficarão no registrador ADRESL e os 2 bits mais significativos ficarão em ADRESH.

Caso seja configurado para justificativa à esquerda, os 8 bits mais significativos ficarão no Registrador ADRESH e os 2 bits menos significativos estarão em ADRESL.

OBS: O conversor A/D necessita de 1,6μs para poder realizar a conversão e, com isso, é necessário certa atenção na hora de configurar os bits.

```
/*************************************************************************
Linguagem C – Compilador MikroC
Programa 7
Objetivo: apresentar a leitura da tensão medida através do potenciômetro acoplado em
AN0 e mostrar no LCD a tensão correspondente - valor 0 a 5V
*************************************************************************/

// Conexões LCD 2x16
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
```

```
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Final Conexões LCD 2x16

float leitura_AD1;
char texto[16];
void main()
{
 char temp []= "Tensao:";
 trisc=0;
 trisb=0; // configura portb como saída
 trisa.f1=1; // configura RA1 como entrada
 adcon1= 0b00000100; // configura RA0, RA1 e RA3 como entrada analógica e demais pinos como I/O digital.

 Lcd_Init();                    // Inicializa LCD
 Lcd_Cmd(_LCD_CLEAR);           // Clear display
 Lcd_Cmd(_LCD_CURSOR_OFF);      // Cursor off
 Delay_ms(20);
 Lcd_Cmd(_LCD_TURN_ON);
 Lcd_Out(1,1,"Tensao:");

 do
 {
  Delay_ms(1000);
  leitura_AD1=ADC_Read(1);
  delay_us(22);
  leitura_AD1 = ((leitura_AD1*5)/1023);

  FloatToStr (leitura_AD1,texto);  //Converte uma variável "float" para uma "string"
```

```
  Lcd_Cmd(_LCD_CLEAR);
  Lcd_Out(1,1,"Tensao:");
  Lcd_Out(2,6,texto);
  Lcd_Out(2,15,"V");
}
while(1);
}
```

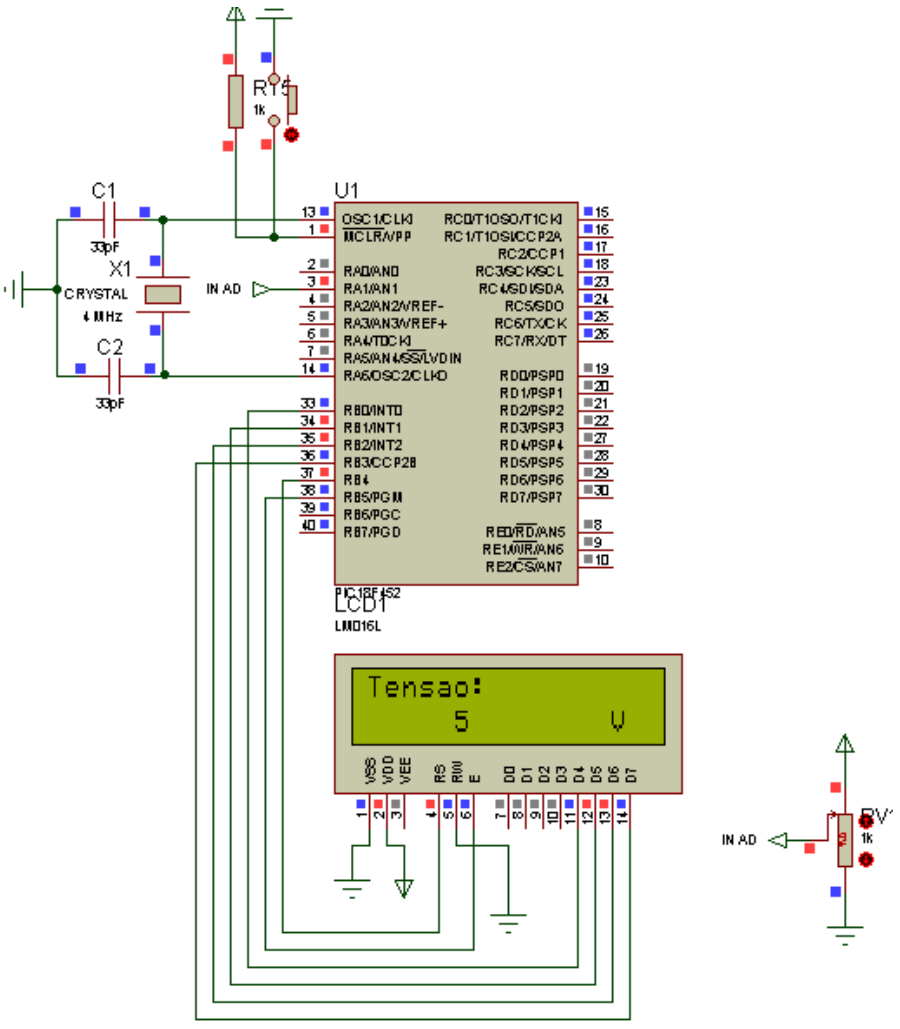


Figura 15 - Esquemático Programa 7

6.8 Comunicação Serial – RS232

O PIC 18F452 possui um USART (*Universal Synchronous Asynchronous Receiver* Transmitter) que têm a função de transmitir/receber dados entre o Microcontrolador e outro Microcontrolador ou equipamentos em geral como PLCs, PCs, etc.

Podemos configurá-lo para as seguintes taxas (bps): 110, 220, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, etc.

Os pinos RC6/TX e RC7/RX são responsáveis pela comunicação, mas também existe a necessidade de se alocar os sinais no padrão RS232 e para isso utilizamos o circuito integrado MAX232 pois o PIC trabalha apenas em 5V.

As Funções abaixo são responsáveis pelo tratamento da Comunicação Serial no MikroC:

Soft_UART_Init(taxa) - inializa o módulo de comunicação do PIC e configura a taxa de transmissão.

Soft_UART_Write('x') – comando para enviar o dado.

Soft_UART_Read( ) – comando para fazer leitura do dado no Buffer Serial.

```
/**********************************************************************
Linguagem C - Compilador MikroC
Programa 8
Objetivo: programa que mostra no display a temperatura ambiente e
transmite a cada 10 segundos para o PC via canal serial
**********************************************************************/
// Conexões LCD 2x16
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Final Conexões LCD 2x16

int leitura_AD1;
char texto[8];
```

```
void main()
{
 char temp []= "Temperatura:\r";
 trisc=0;
 trisd=0b11111011; // configura pino d1 saida
 trisb=0; // configura portb como saída
 trisa.f1=1; // configura RA1 como entrada
 adcon1= 0b00000100; // configura RA0, RA1 e RA3 como entrada analógica e demais pinos como I/O digital.

  Lcd_Init();                    // Inicializa LCD
  Lcd_Cmd(_LCD_CLEAR);           // Clear display
  Lcd_Cmd(_LCD_CURSOR_OFF);      // Cursor off
  Delay_ms(20);
  Lcd_Cmd(_LCD_TURN_ON);
  Lcd_Out(1,1,"Leitura da ");
  Lcd_Out(2,4,"Temperatura... ");
  Delay_ms(2000);

  // Comunicação com o PC
  UART1_Init(9600); //Inicializa o modulo UART com 9600 bps
  Delay_ms(100);
  UART1_Write_Text( temp );
  do
  {
   Delay_ms(1000);
   leitura_AD1=ADC_Read(1);
   delay_us(22);
   leitura_AD1=(leitura_AD1/2.048) ;
   IntToStr(leitura_AD1,texto);  //Converte uma variável "Int" para uma "string"
   UART1_Write_Text(texto);
   UART1_Write('\r');

   portd.f2=1;
   delay_ms(25);
   portd.f2=0;

   Lcd_Cmd(_LCD_CLEAR);
   Lcd_Out(1,1,"Temperatura:");
   Lcd_Out(2,8,texto);
   Lcd_Out(2,15,"oC");
 }
 while(1);
 }
```

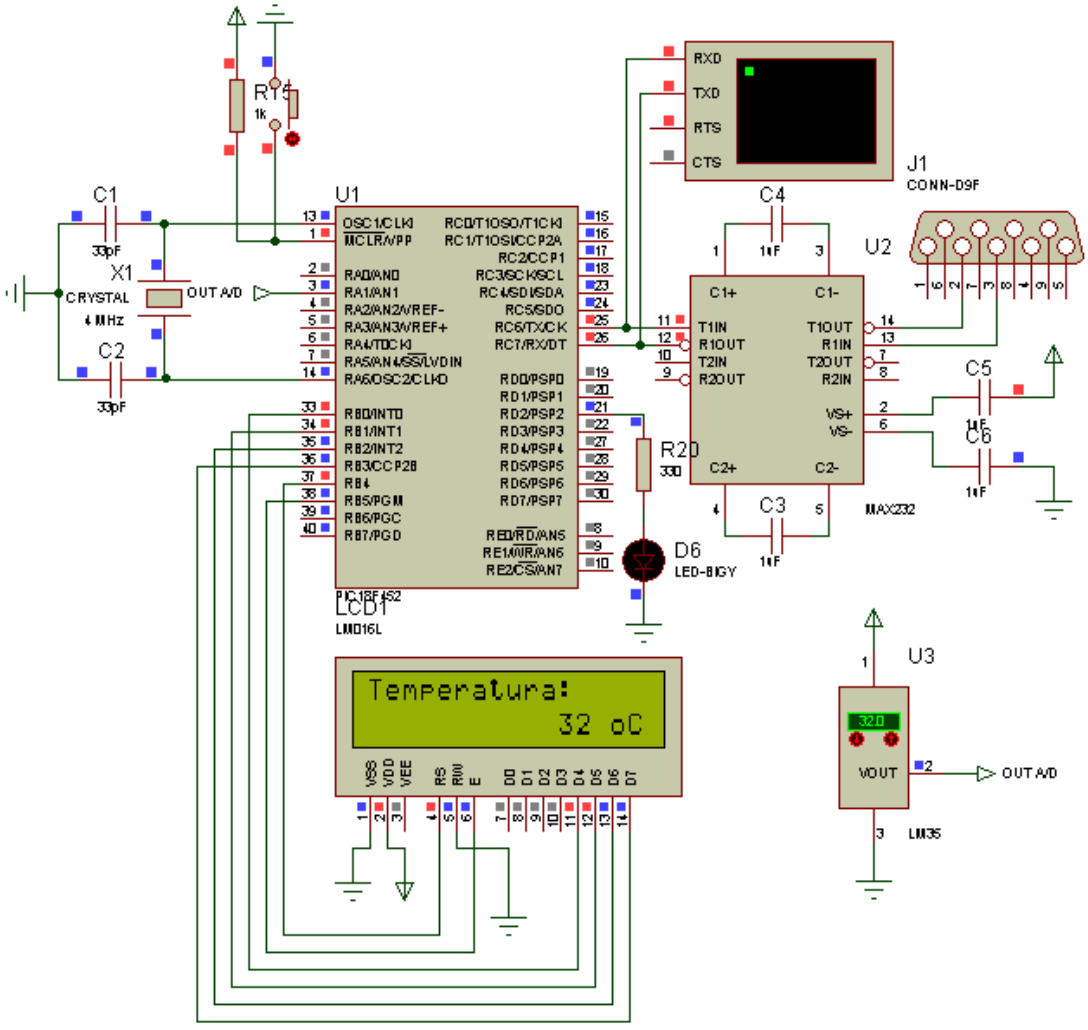


Figura 16 - Esquemático Programa 8

7 REFERÊNCIAS

[1] Dogan Ibrahim, Advanced PIC Microcontroller Projects in C, NewNes, 2008.

[2] http://www.microchip.com/ - Acessada em 18/05/2011

[3] http://www.mikroe.com/pdf/mikroc/mikroc_manual.pdf - Acessada em 18/05/2011

[4] http://www.labcenter.com/index.cfm - Acessada em 10/08/2011