MikroC

MikroC

A biblioteca de linguagem de programação MikroC é a programação mais “rica”, porque é composto por muitas rotinas já implementadas. Além disso, possui recursos excecionais, como:

- Baseado em C.
- A estrutura de comando é fácil, conveniente e flexível.
- Contém o software necessário para programar o microcontrolador.
- Apresenta exemplos de diagramas de circuito e códigos de programa em quase todos os contextos.
- Possibilita o acesso fácil e flexível ao hardware do microcontrolador.

Mesmo que um utilizador que não tenha conhecimento de programação PIC pode tornar-se um bom programador PIC, analisando os exemplos contidos na biblioteca MikroC. A maioria das aplicações apresentadas neste manual foram feitas utilizando a biblioteca MikroC. Os programas MikroC Pro e Proteus são suficientes para poder aplicar todos estes exemplos em ambiente digital.

Podemos descarregar o MikroC PRO para microprocessadores PIC a partir do site www.mikroe.com/mikroc-pic. Podemos descarregar o programa Proteus do site www.labcenter.com/simulation para fazer simulações de circuitos com o microcontrolador relevante.

Regras de sintaxe

A linguagem MikroC tem suas próprias regras de sintaxe. O compilador MikroC irá ocorrer um erro quando essas regras forem violadas. Essas regras são:

- Existem palavras-chave reservadas pelo MikroC não podem ser utilizadas pelo utilizador. Tais como:
  asm, code, switch, static, operator.

- As explicações ou comentários devem ser escritas após o sinal “//”, por exemplo:
  //Erasmus+ KA202 Project

- Para dar nome a variáveis, a constantes e a funções, podem ser utilizados todas as letras do alfabeto (maiúsculas ou minúsculas), números, “_”. No entanto, o primeiro caractere do nome especificado deve ser uma letra ou "_". MikroC diferencia maiúsculas de minúsculas.
  Total3, Fuse, temp_t1, _user

- Os inteiros podem ser escritos em sistemas numéricos decimais, hexadecimais, binários ou octais. O sistema de numeração hexadecimal deve começar com “0x”, o sistema de numeração binário “0b” e o sistema de numeração octal “0”. Não precisamos escrever uma expressão para o sistema de numeração decimal. A parte decimal em números decimais continua com “.”.
  0xFE, 0b1101, 0131, -1.35

- As expressões de caracteres são escritas entre aspas simples. Expressões de string são colocadas entre aspas duplas.
  'b', 'M', "Hello World"

• Os caracteres que não são digitados diretamente no ecrã, mas têm tarefas relacionadas à exibição, são chamados de caracteres de espaço. Começam com "\". Por exemplo:
  \n - vai para a última linha

Os sinais separadores são usados:

[ ] é usado para especificar os comprimentos e índices dos arrays.

```
char str[9] = "IQinDIGIT"
```

( ) é usado em grupos, instruções de comparação, chamadas de função e operações aritméticas.

```
top = goods * (qty + price),    if (m == 100) ++y;    fonktop ( );
```

{ } indica o início e o fim de um bloco de código.

```
if (m == 100) {    }
```

, é colocado entre parâmetros, variáveis ou elementos de array.

; é usado para terminação.

= é usado para atribuir valores a variáveis ou constantes.

é usado para indicar que a ação é uma ação do compilador.

```
#include <math.h>
```

Tipos de Dados

Os tipos devem ser especificados ao definir variáveis e constantes na linguagem de programação MikroC.
O qualificador const é usado para definir uma constante. O valor da constante definida permanece o mesmo ao longo de todo o programa.

```
const double HP = 0.735;
```

Expressões que especificam tipos aritméticos são void, char, int, float e double. *Signed* e *unsigned* vêm antes dessas palavras para indicar que a variável representa um número negativo ou positivo. Além disso, as palavras short e long indicam o comprimento da variável.

Tipo	Byte	Limites
(unsigned) char	1	0…255
signed char	1	-128…127
(signed) short (int)	1	-128…127
unsigned short (int)	1	0…255
(signed) int	2	-32768…32767
unsigned (int)	2	0…65535
(signed) long (int)	4	-2147483648…2147483647
unsigned long (int)	4	0…4294967295
float	4	$-1.5*10^{45}$…$+3.4*10^{38}$
double	4	$-1.5*10^{45}$…$+3.4*10^{38}$
long double	4	$-1.5*10^{45}$…$+3.4*10^{38}$

Operadores

Operadores da mesma categoria têm a mesma prioridade. Para operadores com a mesma precedência na mesma linha, a ordem das operações é da esquerda para a direita. = *= /= %= += -= &= ^= <<= >>=, ! ++ -- + - * & (tipo) exceto sizeof. A não ser que haja algum parêntese na linha.

Os operadores que realizam operações como adição, subtração, multiplicação são chamados de operadores aritméticos.

Operador	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
++	Incrementador
--	Decrementador

Os operadores que realizam comparações lógicas entre expressões e retornam TRUE ou FALSE como resultado são chamados de operadores de comparação.

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

Os operadores usados no processo de passar um valor para uma variável são chamados de operadores de atribuição. O mais utilizado é o operador "=". O operador de comparação não deve ser confundido com ==.

Operador	Significado
=	Atribuição básica
+=	Atribuição com adição
-=	Atribuição com subtração
*=	Atribuição com multiplicação
/=	Atribuição com divisão
%=	Atribuição com resto de divisão

Os operadores que retornam TRUE ou FALSE fazendo uma ou mais comparações são chamados de operadores lógicos.

Operador	Significado
&&	AND lógico
||	OR lógico
!	NOT lógico

Além disso, os operadores << deslocamento à esquerda e >> deslocamento à direita também são muito usados.

Comandos de controlo

As instruções de controlo são estruturas que comparam os valores fornecidos durante o fluxo do programa e permitem que o programa seja processado de acordo com esses valores.

If - Else

O bloco if-else faz comparações. Isso permite que o programa ramifique com base no resultado da comparação.

```
if (condição)
        Linha a ser executada quando a condição for verdadeira
```

If pode ser usado sozinho ou em conjunto com Else.

```
if (condição)  {
        Linha a ser executada quando a condição for verdadeira
}
else  {
        Linha a ser executada quando a condição for falsa
}
```

Exemplo: `if ( pos > 0 ) pos --; else pos = 3 ;`

Switch - Case

A instrução switch verifica se uma instrução corresponde a um inteiro constante e permite que o programa flua de acordo com a correspondência. Quando uma correspondência é encontrada, o comando break é usado para encerrar a instrução de controlo. Se nenhuma das condições corresponder, os comandos na *default* padrão serão executados.

```
Switch (variável) {

    case constant0 : expressões ; break;

    case constant1 : expressões ; break;

    case constant2 : expressões ; break;

    default : expressões ;

{
```

Estruturas de iteração (ciclos)

Ao escrever um programa, precisamos que algum código seja executado mais de uma vez. Estruturas de *loop* são estruturas usadas para várias execuções desses códigos. Existem 3 tipos de ciclos.

While

O ciclo while executa o ciclo se a condição dada for verdadeira. Antes de entrar no ciclo, a condição é verificada, se for TRUE, o ciclo é executado, se for FALSE, o ciclo continua da próxima linha.

```
while ( condição ) {

    linhas de comando para executar;

}
```

Do-While

O ciclo do-while é uma variação do ciclo while. A única diferença é que a correção da condição é feita no final do ciclo. Portanto, mesmo que a condição seja falsa, o bloco de ciclo é executado uma vez.

```
do {
    linhas de comando para executar;

    } while ( condição );
```

For

É o tipo de ciclo mais utilizado. Nos tipos de ciclo anteriores, utilizamos uma variável contador para contar o número de ciclos. Verificamos constantemente a condição do ciclo aumentando ou diminuindo esse contador. No entanto, esse processo é bastante simples no ciclo For.

```
for ( valor inicial do contador; expressão de condição; aumentar/diminuir o valor do contador ) {

    linhas de comando para executar;

}
```

Quando uma instrução *break* é encontrada num ciclo, este é terminado. A instrução *continue* é usada para retornar ao início do ciclo.

```
while ( .. ) {
    ....
    ....
}
```

```
do {
    …
    …
while ( .. );
```

```
for ( .. ; .. ; .. ) {
    …
    …
}
```

Funções e Arrays

Funções são subprogramas que executam algumas tarefas tendo em conta os parâmetros de entrada. Se vamos realizar uma operação mais de uma vez no programa, essa operação pode ser transformada numa função, em vez de reescrever os mesmos códigos. Assim, podemos chamar essa função sempre que precisarmos. Depois das operações contidas na função serem executadas, esta pode retorna um valor ao programa principal com o comando return. Em C, o programa principal é escrito dentro do bloco da função main(). Outras funções são utilizada chamando-as de dentro deste programa principal.

```
O tipo que a função irá devolver Nome da função (lista de parâmetros);

void main() {

    int temp_change ( int x,   int y ) {

        …

      return (…)
    }
}
```

Ao definir uma função, Void é utilizado nos casos em que o resultado desta função não pode ser retornado ao local onde é chamada.

```
Void pressure_change (char pressure) {

        Lcd_Out_Cp (“ Pressure : ”);

        Lcd_Out_Cp (pressure);
}
```

Arrays são um dos das estruturas de dados mais utilizados na linguagem C. Com *arrays*, podemos armazenar mais de um dado do mesmo tipo.

```
int array_name [index] = { Os elementos do array são escritos com uma vírgula entre eles. }

unsigned short fuses [9] = { 6, 10, 16, 20, 25, 32, 40, 50, 63 }
```

O valor do índice do primeiro elemento de *arrays* é sempre 0.

Comandos do pré-processador

O pré-processador faz parte do processo de compilação. Os comandos do pré-processador são executados antes de iniciar a compilação. As linhas que começam com o sinal # são comandos do pré-processador. Podemos adicionar um ficheiro externo ao nosso programa utilizando comandos do pré-processador. O comando #include é utilizado para isso. Também podemos definir macros como funções e usá-las em programas usando um pré-processador. Utilizamos comandos de pré-processador #define para criar macros.

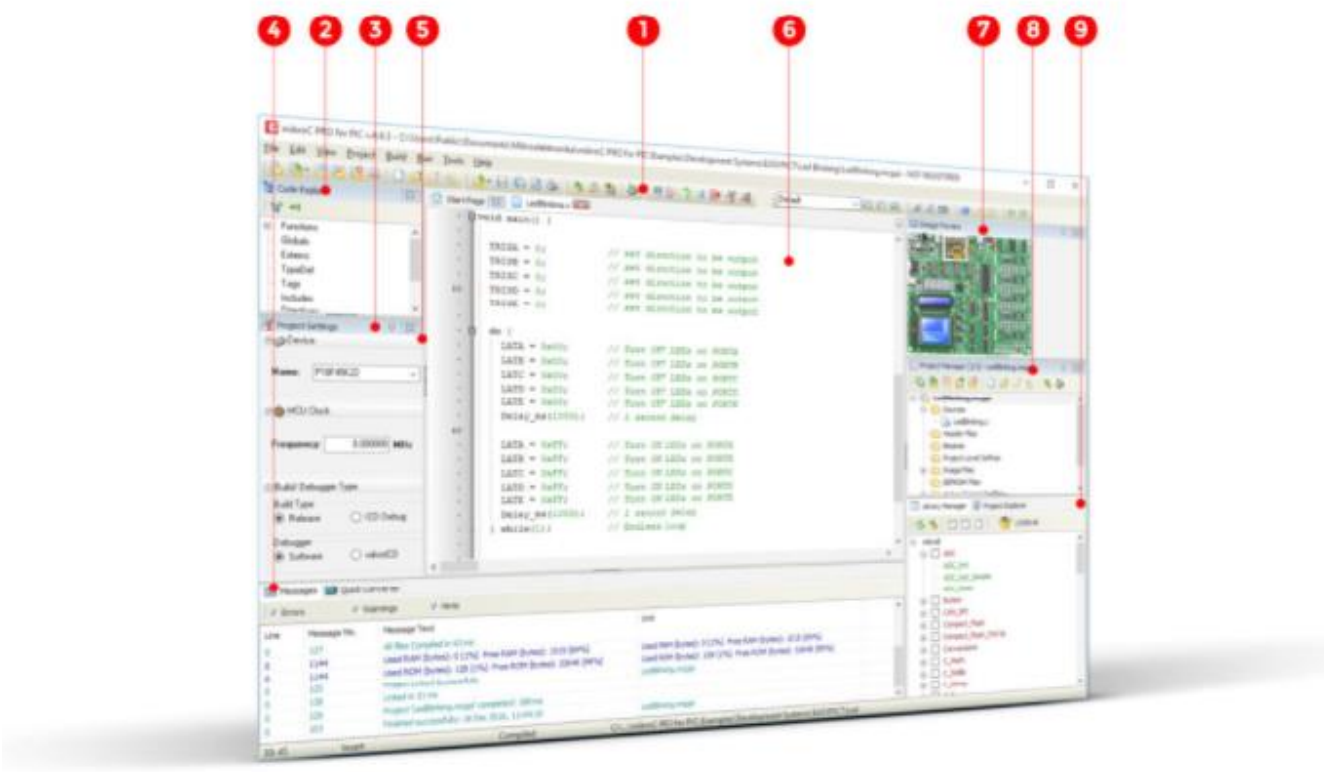
```
#include <file_name>

#include “built_in.h”

#define   servo1    portd.rd0
```

Compilador MikroC e Organização do Projeto

O compilador MikroC Pro é desenvolvido pela empresa MikroElektronika. MikroC Pro não é apenas um compilador; oferece-se também uma área onde códigos de microcontroladores podem ser escritos, janelas onde podemos obter todo tipo de informação sobre esses códigos, utilitários (Terminal USART, Terminal HID, GLCD Bitmap Editor etc.) e uma biblioteca muito vasta. A biblioteca MikroC é instrutiva e fácil de usar.



1. Barra de ferramentas principal

Com *layout* padrão, a primeira seção da barra de ferramentas principal é dedicada à criação, edição e exclusão do projeto. A segunda seção é para adicionar ficheiros. A terceira contém as opções para guardar e de impressão. A quarta é para construir projeto e iniciar o programador. A quinta é para gerir o terminal USART, a ferramenta de edição de EEPROM e outras opções. A sexta é para recursos relacionados com o *layout*. A sétima é para montagem, listagens e estatísticas. A oitava é permite aceder à Ajuda e a pasta de exemplos. A nona e última seção é o Histórico, permite refazer os passos no Editor de Código.

2. Code Explorer

O Code Explorer encontra-se localizado no canto superior esquerdo do ecrã. Podemos ver uma lista de funções, links da web e comentários ativos no projeto que se encontra aberto.

3. Project Settings

Nesta seção, podemos encontrar o nome do dispositivo que está a ser usado, a frequência do relógio do MCU e os tipos de Build/Debugger. A frequência do relógio MCU determina a velocidade do microcontrolador.

4. Messages

Caso sejam encontrados erros durante a compilação, o compilador os colocará na caixa Mensagem. O compilador também cria avisos, mas estes não afetam a saída.

5. Quick Convertor

Facilita a conversão de uma forma para outra. Por exemplo, de um número decimal para um número binário.

6. Code editor

O Editor de código possui realce de sintaxe ajustável, assistente de código, assistente de parâmetros, correção automática para erros de digitação comuns e modelos de código (completar automaticamente).

7. Image Preview

A caixa de visualização da imagem está localizada no canto superior direito do ecrã, por defeitp. Mostra as imagens dos *links* ativos que foram adicionados na seção de comentários do código.

8. Project Manager

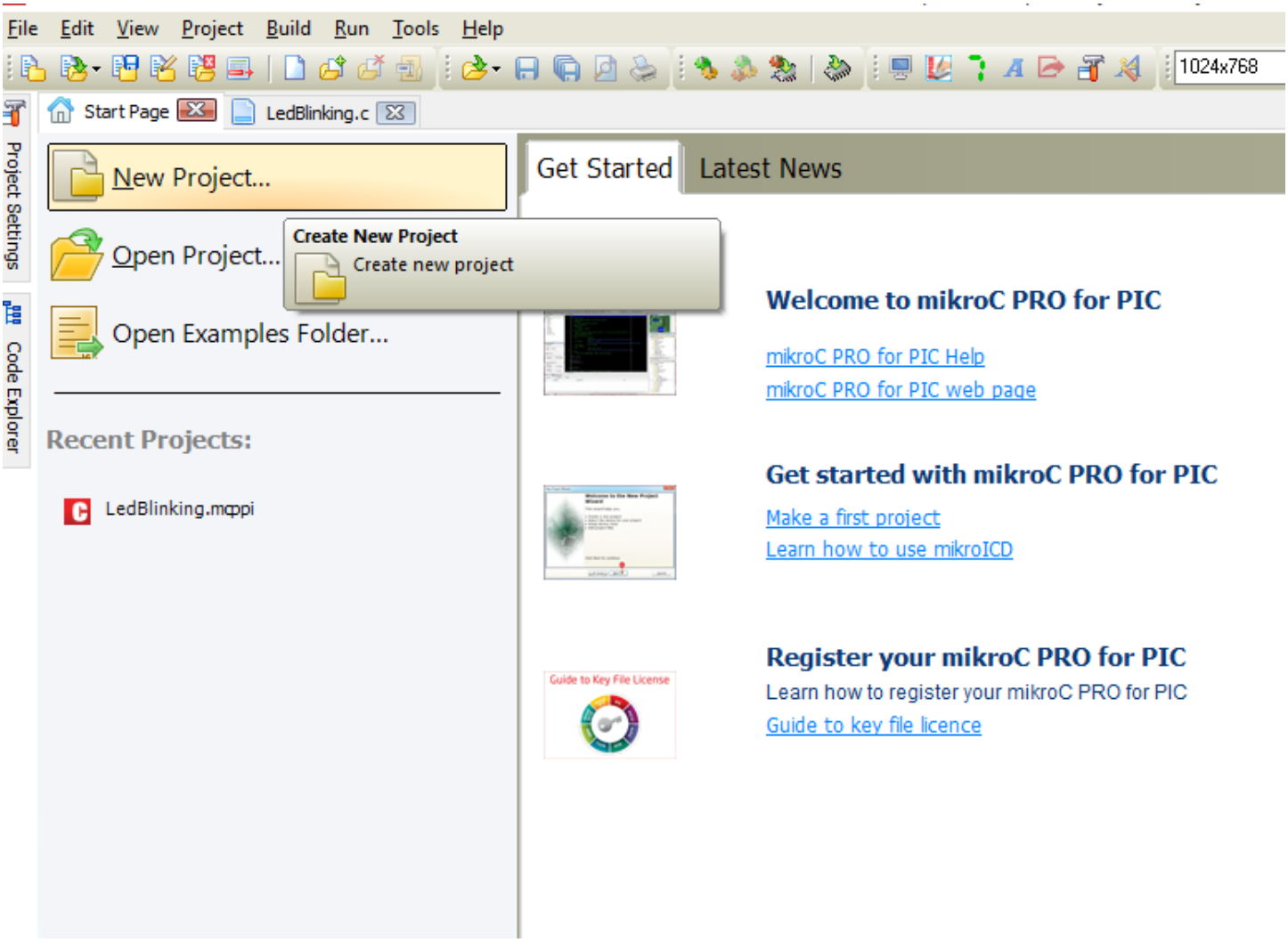
O Project Manager é um recurso do IDE que permite criar vários projetos. Ele mostra os ficheiros de origem e de cabeçalho em cada projeto. Vários projetos podem ser abertos ao mesmo tempo, mas apenas um deles pode estar ativo de cada vez. Para definir um projeto em modo ativo é necessário clicar duas vezes no projeto desejado no *Project Manager*.

9. Library Manager

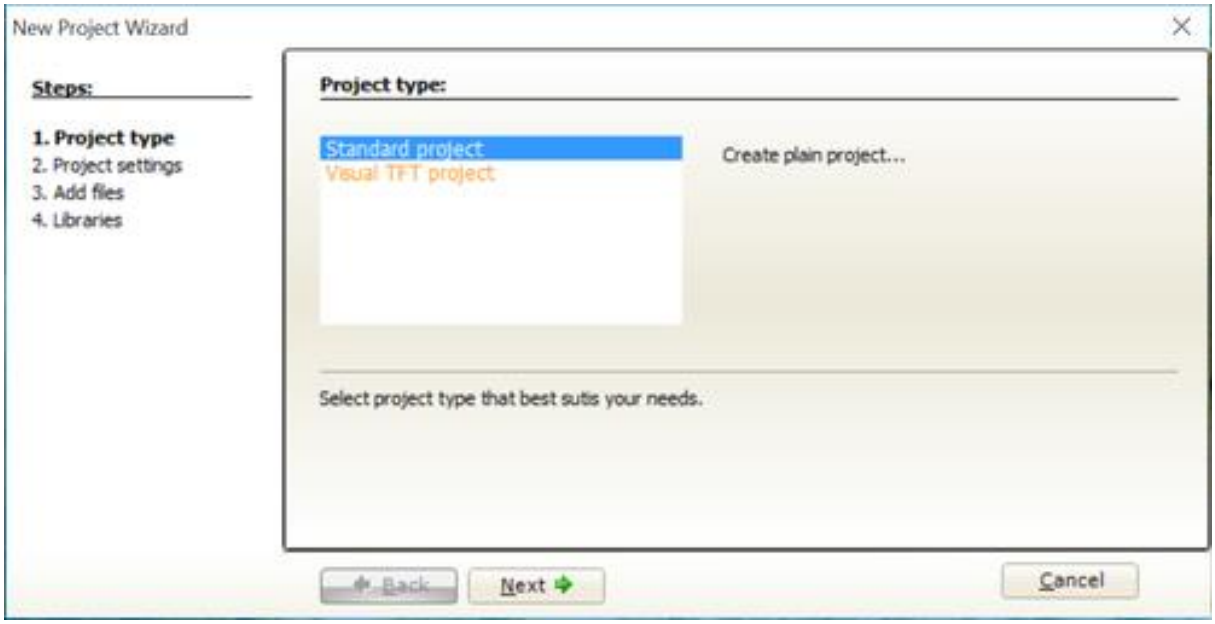
A Library Manager permite trabalhar com as bibliotecas de forma simples e fácil. A janela Library Manager lista todas as bibliotecas. A biblioteca desejada é adicionada ao projeto marcando a caixa de seleção ao lado do nome da biblioteca. Para ter todas as funções da biblioteca disponíveis, basta pressionar o botão Check All.

Criando o primeiro projeto

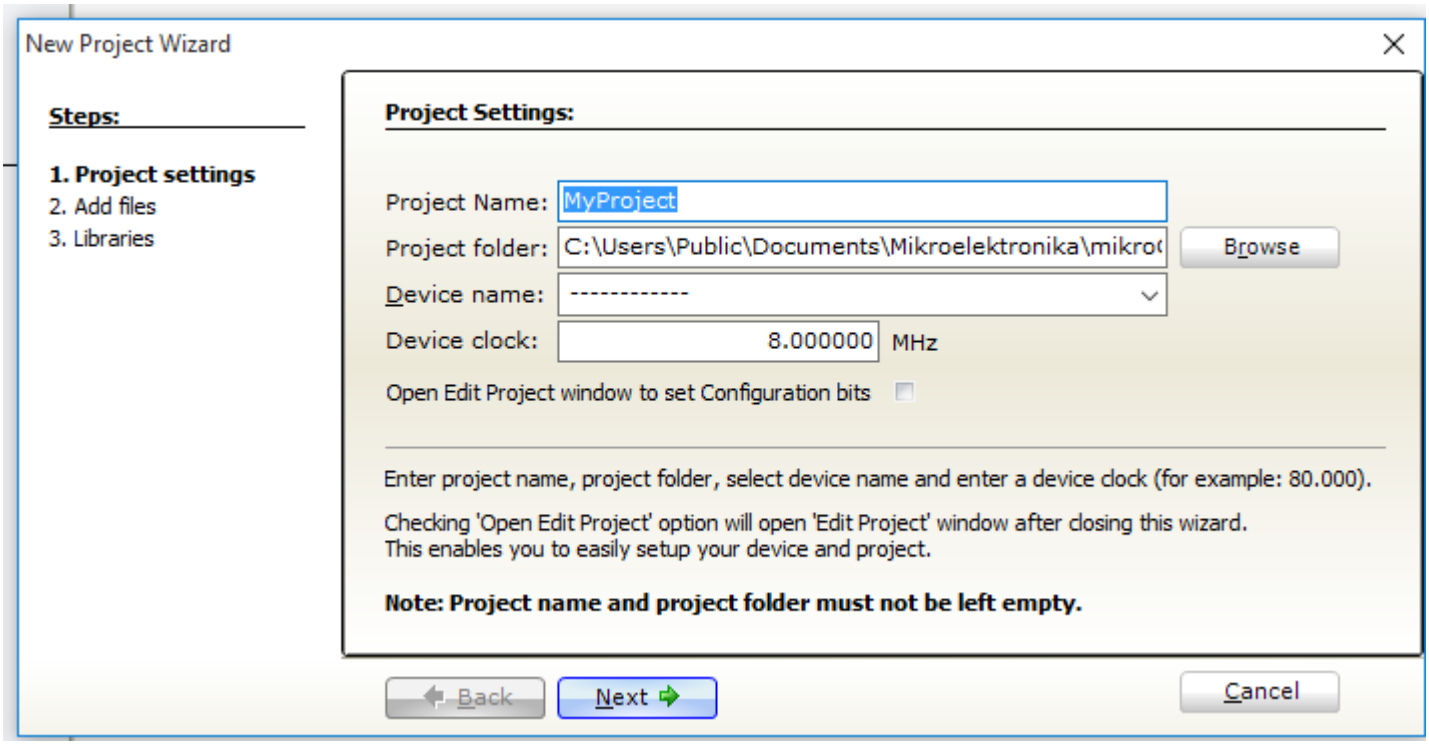
Nesta parte do guia, mostraremos como criar o primeiro projeto no compilador, compilá-lo e testar os resultados.
Antes de tudo, é preciso ter instalado o compilador MikroC Pro no computador.
Ao abrir o compilador, selecione a opção New Project no menu Project.



A janela New Project Wizard aparecerá e o guiará facilmente no processo de criação do projeto.
Uma nova janela aparecerá pedindo para escolher se pretendemos cria um projeto Visual TFT ou um projeto Standard e indicar qual dos seguintes compiladores: PIC, dsPIC ou FT90x.



A primeira coisa que temos a fazer é especificar as informações gerais do projeto. Selecione o microcontrolador de destino, sua frequência de operação e o nome do seu projeto.

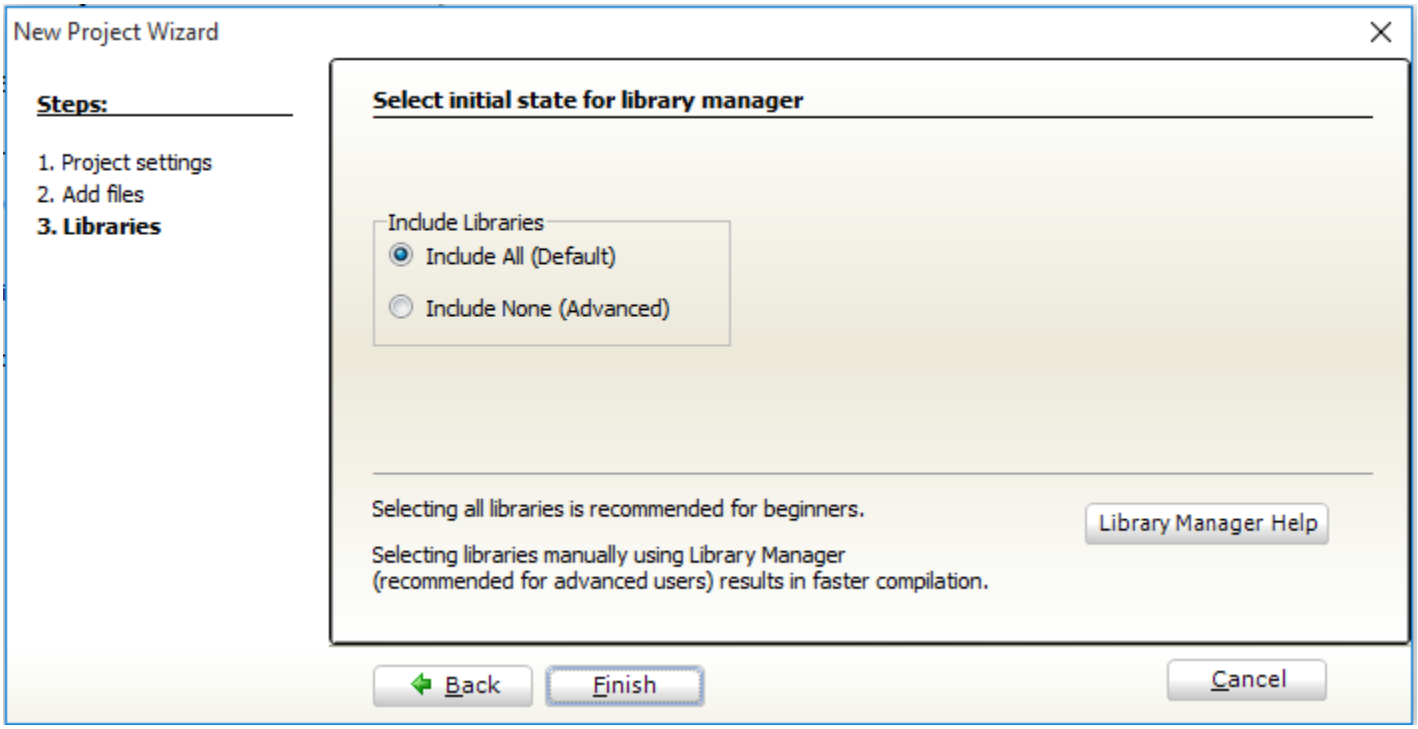


O compilador ajustará as configurações internas com base nas informações inseridas. A configuração padrão é sugerida para no início.
Se não quisermos utilizar o caminho sugerido para guardar o novo projeto, podemos alterar a pasta de destino.
Escolha o nome do projeto, de acordo com o tipo de projeto em que está a trabalhar. Defina o relógio do dispositivo. A velocidade do *clock* depende do hardware a que se destina.

Na outra janela, se os componentes eletrónicos que vamos usar não forem encontradas nas bibliotecas (caso raro), podemos incluir os ficheiros no nosso projeto clicando no botão Add.

Esta etapa permite definir rapidamente se pretendemos incluir todas as bibliotecas no projeto ou não.

Mesmo que todas as bibliotecas estejam incluídas, elas não utilizarão nenhuma memória a menos que elas sejam utilizadas explicitamente no código. A principal vantagem de incluir todas as bibliotecas é que teremos mais de 500 funções disponíveis para utilizar no código. Essas funções serão visíveis no Code Assistant [CTRL+Space]. A configuração padrão é a opção "Include All".
Quando terminar, clique no botão Finish.



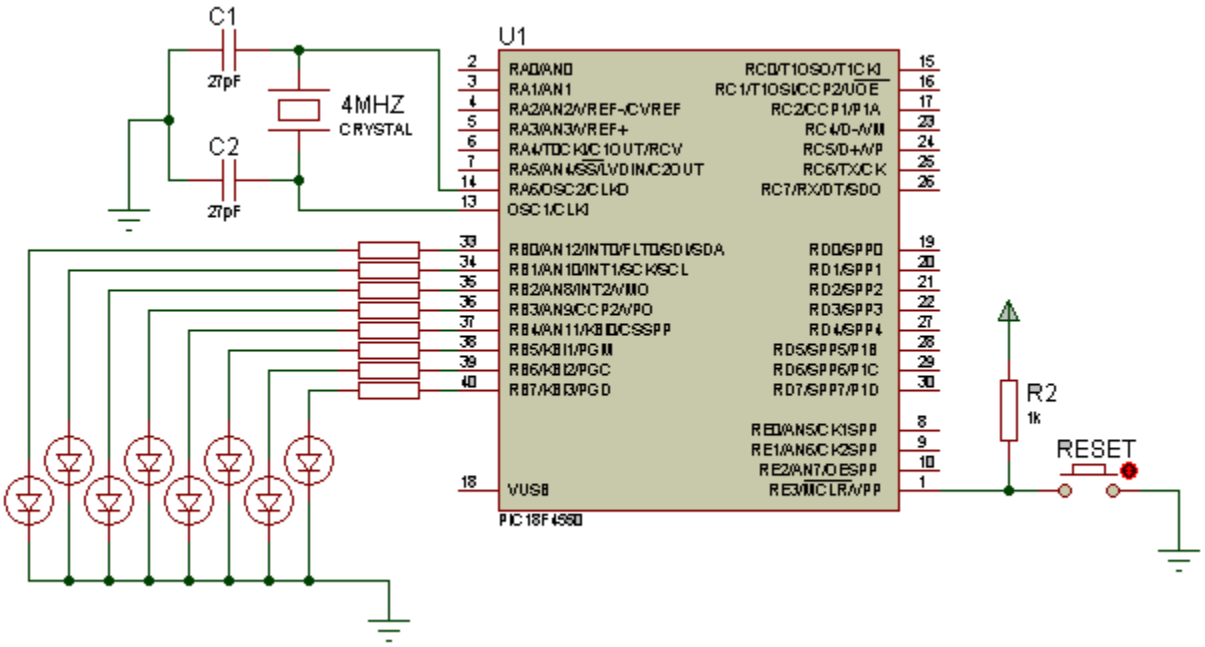
Agora a janela onde vamos escrever nosso código aparece no ecrã. Não podemos carregar diretamente o programa que escrevemos no microcontrolador. Temos que compilar e criar o código .hex. Alguns atalhos são recomendados para isso:

Ctrl+Alt+G - para criar o código
Ctrl + Shift + S – para gravar todos os ficheiros
Ctrl + F9 - para compilar o código criado ou a opção Build no menu Project
Ctrl + F11 - construir código criado e programa para MCU

Após o processo de construção, o compilador cria os ficheiros .asm, .hex, .mcl e .lst na pasta onde o projeto é criado.

Trabalho 1

Nesta aplicação, utilizaremos as portas do microcontrolador PIC18F4550 como saída.



O diagrama de circuito da aplicação é mostrado na figura. Por favor, projete o circuito no programa Proteus. Antes de compilar o programa MikroC, não se esqueça de selecionar o oscilador nas configurações. Como usamos cristal como oscilador no circuito, “XT oscillator” deve ser selecionado na janela que aparece executando a opção de menu Project-Edit Project.

Aplicação 1

Ligar um díodo LED ligado ao pino RB0 da porta B, 5 segundos após o início do programa.

```
void main () {
        trisb.rb0 = 0 ;
        portb.rb0 = 0 ;
        delay_ms (5000) ;
        portb.rb0 = 1 ;
}
```

Aplicação 2

Ligar e desligar um diodo LED ligado ao pino RB0 da porta B em intervalos de 1 segundo.

```
void main () {
        trisb.rb0 = 0 ;
        portb.rb0 = 0 ;
        while (1)
        {
                portb.rb0 = 0 ;
                delay_ms (1000) ;
                portb.rb0 = 1 ;
                delay_ms (1000) ;
        }
}
```

Aplicação 3

Ligar e desligar 8 díodos LED ligados ao PORTB em intervalos de 1 segundo.

```
void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;

        trisb = 0 ;
        portb = 0 ;
        while (1)
        {
                portb = ~portb ;
                delay_ms (1000) ;
        }
}
```

Aplicação 4

Aplicação de contador binário ascendente e descendente até 255 com intervalos de 0,5 segundo com 8 díodos LED ligados ao PORTB.

```
void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;

        trisb = 0 ;
        portb = 0 ;
        while (1)
        {
                do {
                        portb++ ;
                        delay_ms (500) ;
                } while ( portb < 255 ) ;
                delay_ms (2000) ;
                do {
                        portb-- ;
                        delay_ms (500) ;
                } while ( portb > 0 ) ;
                delay_ms (2000) ;
        }
}
```

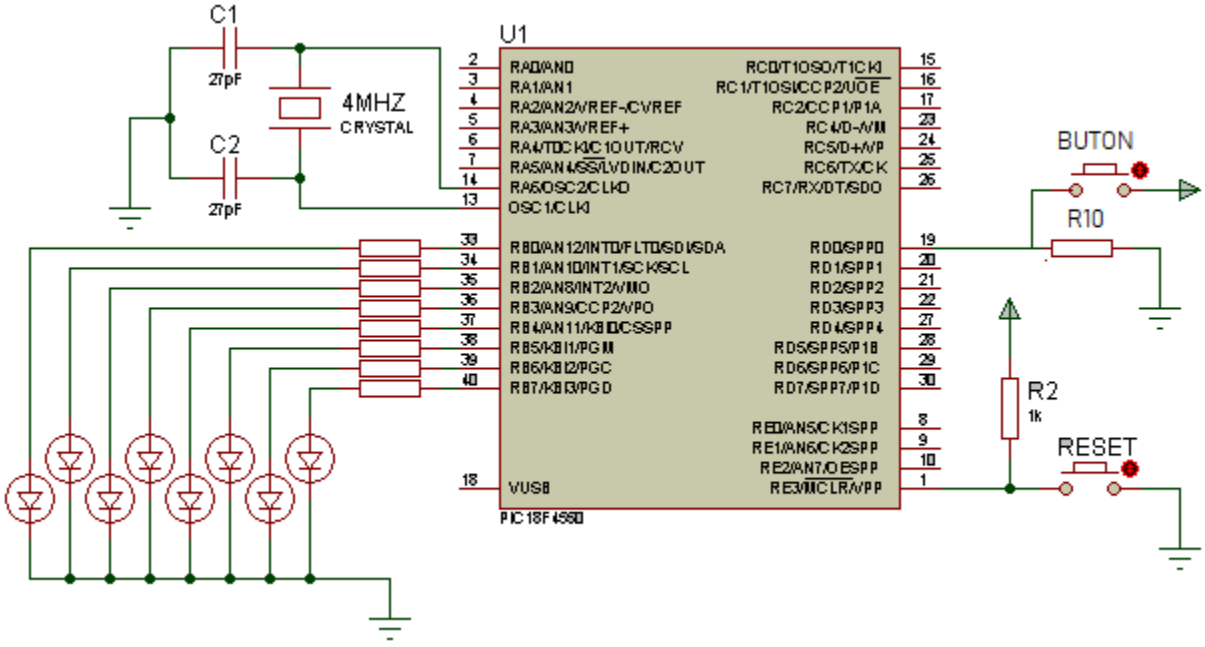
Aplicação 5

Um programa criado para piscar sequencial de 8 díodos LED ligados ao PORTB.

```
void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;
        trisb = 0 ;
        portb = 0 × 01 ;
        while (1)
        {
                while ( portb < 0 × 80 )  {
                        delay_ms (200) ;
                        portb = portb << 1 ;
                }
                while ( portb > 0 × 01 ) {
                        delay_ms (200) ;
                        portb = portb >> 1 ;
                }
        }
}
```

Trabalho 2

Neste trabalho, vamos usar as portas do microcontrolador PIC18F4550 como entradas.



Com o BUTTON ligado ao pino RD0 da porta D, os díodos de LED ligados à porta RB são controlados. O botão ligado ao botão RD0 é configurado para fornecer Logic-1 quando pressionado com um resistor pull-down. No programa, toda a porta B é orientada como saída e o pino RD0 da porta D é orientado como entrada.

Aplicação 1

Um programa que altera o estado de 8 díodos de LED Ligados ao PORTB quando o botão é pressionado.

```
void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;
        trisb = 0 × 00 ;
        portb = 0 × 0F ;
        trisd.rd0 = 1 ;
        portd.rd0 = 0 ;
        while (1)
        {
                if ( portd.rd0 )  {
                        portb = ~portb ;
                        while (portd.rd0 ) ;
                }
        }
}
```

Aplicação 2

Cada vez que o botão é pressionado, o valor de PORTB é aumentado em um valor.

```
void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;
        trisb = 0 ;
        portb = 0 ;
        trisd.rd0 = 1 ;
        portd.rd0 = 0 ;
        while (1)
        {
                if ( portd.rd0 )  {
                        portb++ ;
                        while (portd.rd0 ) ;
                }
        }
}
```

Aplicação 3

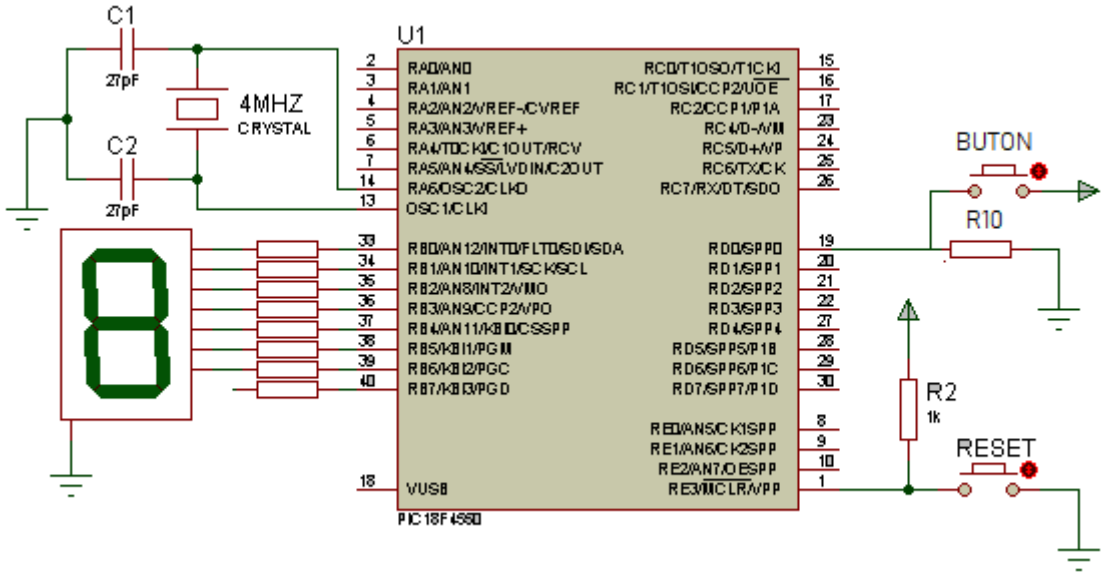
Um programa que tem um valor de 1 carregado no PORTB e desloca o valor em PORTB uma para a esquerda cada vez que um botão é pressionado.

```
void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;
        trisb = 0 ;
        portb = 0 ;
        trisd.rd0 = 1 ;
        portd.rd0 = 0 ;
        while (1)
        {
                if ( portd.rd0 )  {
                        portb <<= 1 ;
                        while (portd.rd0 ) ;
                }
        }
}
```

while (portd.rd0) ; Com esta linha de comando, um loop vazio é criado enquanto o pino RD0 for 1 lógico.

Trabalho 3

Neste trabalho, serão aplicados contadores de 0 a 9 através de displays de 7 segmentos.



Neste circuito, o display de 7 segmentos com cátodo comum é ligado ao PORTB. O link dos pinos está abaixo:

NUMBER	null	g	f	e	d	c	b	a	BIN	HEX
	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0		
0	0	0	1	1	1	1	1	1	0b00111111	0x3F
1	0	0	0	0	0	1	1	0	0b00000110	0x06
2	0	1	0	1	1	0	1	1	0b01011011	0x5B
3	0	1	0	0	1	1	1	1	0b01001111	0x4F
4	0	1	1	0	0	1	1	0	0b01100110	0x66
5	0	1	1	0	1	1	0	1	0b01101101	0x6D
6	0	1	1	1	1	1	0	1	0b01111101	0x7D
7	0	0	0	0	0	1	1	1	0b00000111	0x07
8	0	1	1	1	1	1	1	1	0b01111111	0x7F
9	0	1	1	0	1	1	1	1	0b01101111	0x6F

<u>Aplicação 1</u>

Os dados da tabela estão no programa MikroC. Serão usados como um array que será enviado ao PORTB dentro de um tempo especificado e um após o outro.

Os dados a serem enviados para o display com a linha de comando const char display [10] são definidos como um array de tipo de caractere fixo na memória.

```
const char display [10] = {     0b00111111,
                                0b00000110,
                                0b01011011,
                                0b01001111,
                                0b01100110,
                                0b01101101,
                                0b01111101,
                                0b00000111,
                                0b01111111,
                                0b01101111  } ;
```

```
unsigned short i = 0;

void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;
        trisb = 0 ;
        portb = 1 ;
        while (1)
        {
                portb = display [ i ] ;
                i++ ;
                delay_ms (500 ) ;
                if ( i > 9 ) i = 0 ;
        }
}
```

Aplicação 2

Programa de contador 0-9 para cima e depois 9-0 para baixo.

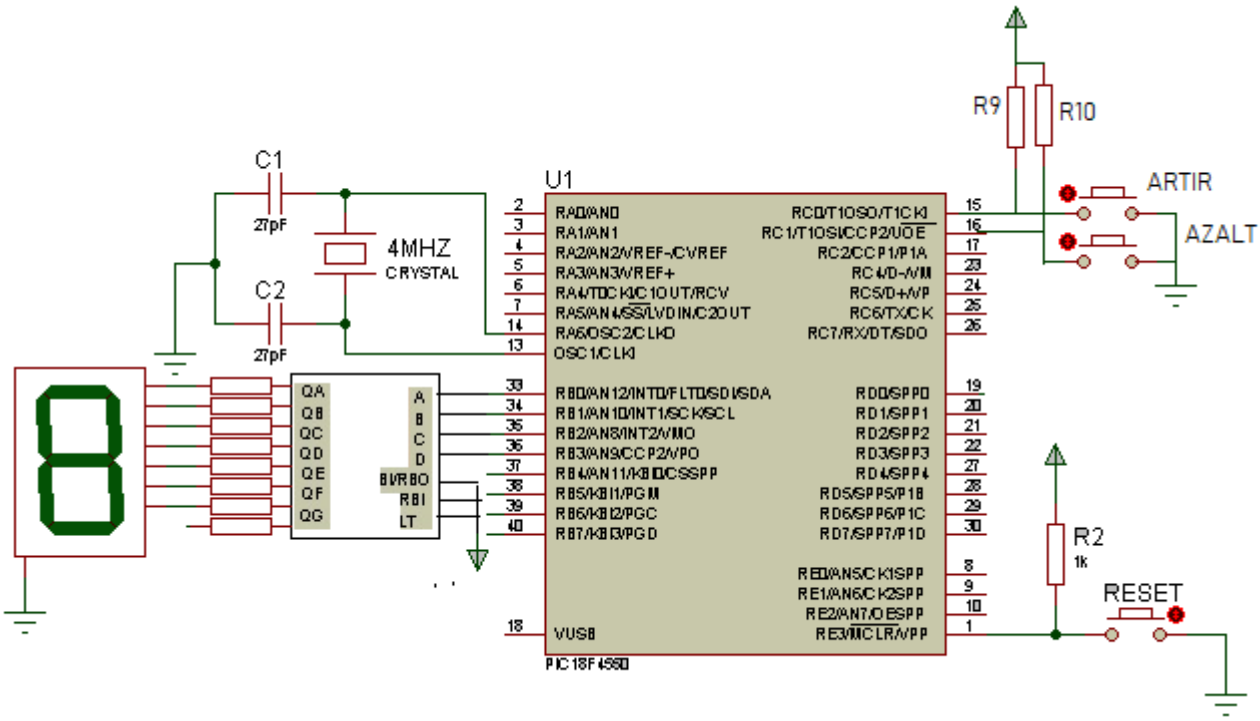
```
const char display [10] = { 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F } ;
unsigned short i = 9;

void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;
        trisb = 0 ;
        portb = 1 ;
        while (1)
        {
                for ( i = 0; i < 10 ; i++ )
                {
                        portb = display [ i ] ;
                        delay_ms (500 ) ;
                }
                for ( i = 10; i > 0 ; i-- )
                {
                        portb = display [ i - 1] ;
                        delay_ms (500 ) ;
                }

        }
}
```

Aplicação 3

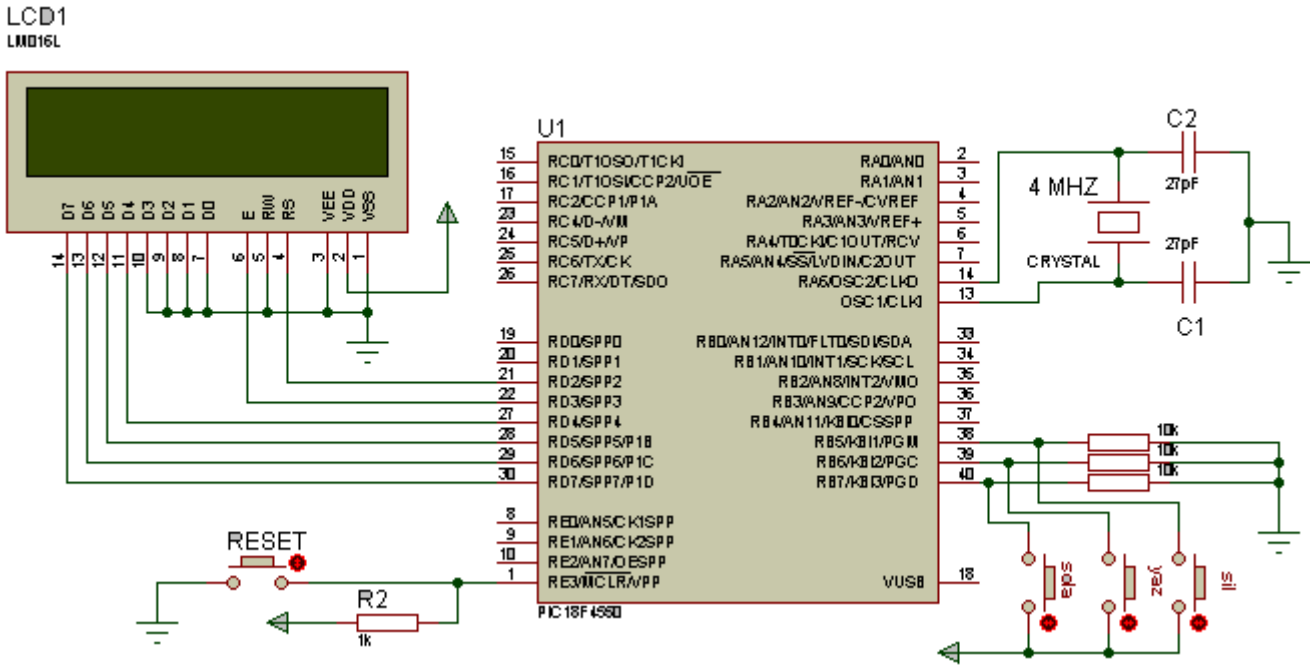
Neste programa, o display de 7 segmentos será utilizado como contador ascendente graças aos botões ligados aos pinos RC0 e RC1 da porta C e ao driver de exibição de cátodo comum 7448 integrado à porta B. Menos pinos do microcontrolador são usados com o IC 7448. Além disso, os códigos MikroC são mais simples.




```
#define artir    PORTC.RC0
#define azalt    PORTC.RC1
short  display = 0 ;
void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;
        trisb = 0 ;
        portb = 0 ;
        trisc = 0 ×03 ;
        portc = 0 ;
        while ( 1)
        {
                if  (!artir)
                {
                        display++ ;
                        while (!artir) ;
                }
                if (!azalt)
                {
                        display-- ;
                        while (!azalt) ;
                }
                if ( (display > 9 ) | (display < 0) ) display  = 0 ;
                portb = display ;
        }
}
```

Trabalho 4

Este trabalho consiste na criação de uma aplicação de LCD com processador HD44780, que é amplamente utilizado no mercado de tecnologia industrial.



As ligações vistas no diagrama devem ser definidas corretamente nos códigos MikroC. Atenção às entradas e às saídas. Além disso, ciclo infinito não é usado neste programa, orque os dados enviados para o LCD permanecem enquanto a RAM do LCD tiver energia.

```
#define sil     portb.rb5
#define yaz     portb.rb6
#define sola    portb.rb7

sbit LCD_RS at RD2_bit ;
sbit LCD_EN at RD3_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 TRISD2_bit ;
sbit LCD_EN_Direction at TRISD3_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 ;
void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 0×07 ;
        Lcd_Init () ;
        Lcd_Cmd (_LCD_CURSOR_OFF) ;
        Lcd_Cmd(_LCD_CLEAR) ;
        Lcd_Out (1, 3,  “Erasmus+”) ;
```

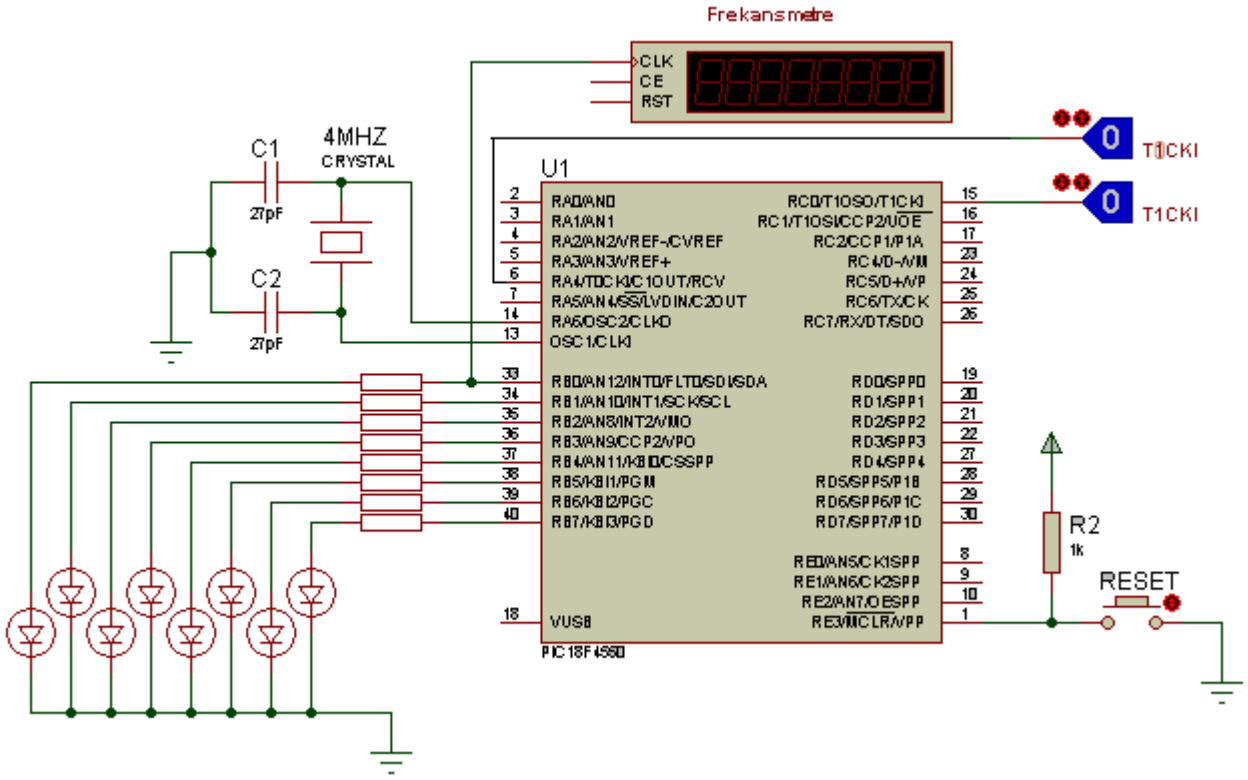
```
        Lcd_Out (2,  1, “IQinDIGIT”= ;
        For ( ; ; )
        {
                if (sil) LCD_Cmd ( _LCD_CLEAR) ;
                if (yaz) {
                        Lcd_Out (1,  2,  “Merhaba” ) ;
                        Lcd_Out (2,  2,  “ Dunya” ) ;
                }
                if (sola) {
                        LCD_Cmd (_LCD_SHIFT_RIGHT) ;
                        while (sola) ;
                }
        }
}
```

Trabalho 5

A caraterística mais importante dos microcontroladores é que eles têm temporizadores de hardware internos. Estes são chamados de TIMERs. Um TIMER pode ser usado como temporizador ou contador. O número de contadores depende do modelo do microcontrolador. Cada TIMER pode criar um INTERRUP. Existem 4 TIMERs no microcontrolador que estamos a utilizar. Utilizaremos o módulo Timer0 aqui. Este módulo pode funcionar como 8/16 bits. Possui recursos como seleção de fonte de clock, seleção de borda de sinal de clock externo, criação de interrupção. A unidade de hardware Timer0 possui o registrador T0CON para configurar suas operações.

Aplicação 1

Consiste em utilizar Timer0 como um timer. Será usado como 8 bits. Timer0 está configurado para ler uma interrupção a cada 5017,6 µs. É utilizado oscilador de cristal de 20 Mhz. O tempo de interrupção depende da proporção do prescaler e do valor padrão carregado no TMR0L.




```
unsigned cnt;
void interrupt () {
        if (TMR0 IF_bit) {
                cnt++;
                TMR0L = 158;
                TMR0 IF_bit = 0 ;
        }
}
void main () {
        ADCON1 l= 0×0F ;
        CMCON l= 7 ;
        cnt = 0 ;
        TRISB = 0 ;
        PORTB = 0×FF ;
```

```
        T0CON = 0×C7 ;
        INTCON = 0×C0 ;
        TMR0IE_bit = 1 ;
        do {
                if (cnt >= 100 ) {
                PORTB = ~PORTB ;
                cnt = 0 ;
                }
        } while (1) ;
}
```

$f_{command}$= Microcontroller Oscillator Frequency / 4 = 20 Mhz / 4 = 5 Mhz

$T_{command} = 1 / f_{komut} = 1 / 5Mhz = 0,2 \times 10^{-6} = 0,2\ \mu s$

Interrupt Time = $T_{command}$ × Prescaler rate ×(256 – TMR0 default value)
= 0,2 µs × 256 × (256 -158)
= 51.2 µs ×98
= 5017,6 µs

PORTB Time to change state = Interrupt Tıme × 100
= 5,0176ms × 100
= 501,76 ms

Referências:

MikroC PRO Complier Quick Start Guide
MikroC and PIC18F4550, Hikmet ŞAHİN, K.Serkan DEDEOĞLU