

UTILIZANDO O COMPILADOR C18 E A IDE MPLABX MULTIPLATAFORMA COM FUNÇÕES EM PORTUGUÊS

Para utilizar esta ferramenta multiplataforma, instale o MPLABX e o compilador C18. É possível baixar gratuitamente o MPLABX e a versão C18 livre e completa (*full*) para o sistema operacional desejado nos links abaixo.

Após instalar o MPLABX e o compilador C18 em <https://drive.google.com/open?id=0B5332OAhnMe2N3czQWxVX0JVSKE&authuser=0> descompacte o arquivo .zip e abra, no MPLABX, o projeto disponível em https://dl.dropboxusercontent.com/u/101922388/ProjSanUSB1_MPLABX/Projeto1C18.X.zip.

Importante enfatizar que estas versões são livres e foram testadas com sucesso. Outras versões podem apresentar erro na compilação e/ou gravação. Detalhes da instalação podem ser vistos em: http://www.youtube.com/watch?v=1EmYiIB_b8I.

Instalador de Gravação:

<https://dl.dropboxusercontent.com/u/101922388/InstaladorGravacao.zip>

O instalador SanUSB contém o Java JRE 6 necessário para a instalação do MPLABX. Caso não seja compatível, baixe e instale o Java JRE 6 (32 ou 64 bits) ou OpenJDK6, se for Linux, antes do MPLABX.

Pasta com todos os arquivos:

<https://drive.google.com/open?id=0B5332OAhnMe2N3czQWxVX0JVSKE&authuser=0>

Compilador Windows:

<https://drive.google.com/open?id=0B5332OAhnMe2STE4MFNBQ2dqYWM&authuser=0>

Compilar Linux:

<https://drive.google.com/open?id=0B5332OAhnMe2cjVRcmF5RU9aMTQ&authuser=0>

Para instalar a IDE MPLABX no Linux após o download, acesse a pasta Downloads pelo terminal e, como super-usuário, digite:

chmod 770 mplabx-ide-v1.41-linux-installer.run

e depois:

./ mplabx-ide-v1.41-linux-installer.run

Realize o mesmo método para instalar o compilador C18.

Compilador Mac OSX:

<https://drive.google.com/open?id=0B5332OAhnMe2b1NISGIYWIBCR0k&authuser=0>

- 1) Para compilar os programas com o C18 e o SanUSB, basta abrir o Projeto1C18.X em https://dl.dropbox.com/u/101922388/ProjSanUSB1_MPLABX/Projeto1C18.X.zip. Todos esses programas estão disponíveis no Grupo SanUSB (www.tinyurl.com/SanUSB). A apostila atualizada pode ser obtida no link:

https://dl.dropboxusercontent.com/u/101922388/Apostila_SanUSB_MPLABX.pdf .

Depois de instalado é possível abrir o projeto MPLAX clicando em *Open project* e escolher o projeto Projeto1C18.X. É possível visualizar o programa com um duplo clique sobre *pisca.c*. Para compilar pressione *Build and Clean* (ícone com martelo e vassoura). Para compilar outros programas.c basta modifica-los ou criá-los com a extensão .c dentro da mesma pasta do projeto Projeto1C18.X e adicioná-los em *Source Files* (somente um firmware por vez).

Em linux, a gravação pode ser realizada também pelo terminal, após a instalação do SanUSB. *Logado* como super-usuário e com o Projeto1C18.X salvo na Pasta Pessoal, basta digitar (ou copiar e colar no terminal):

Endereço absoluto executável sanusb + **-w** + endereço absoluto arquivo .hex + **-r**

`/usr/share/sanusb/./sanusb -w ~/Projeto1C18.X/dist/default/production/Projeto1C18.X.production.hex -r`

Se no Linux Ubuntu, a pasta Projeto1C18.X não aparecer em cor de projeto MPLABX, como um chip, basta criar um novo projeto (File ->New Project-> Standalone Project --> Advanced 8 bits MCU – PIC18F4550 -> PicKit3 -> C18 -> Nome Projeto2 e *Finish*.). Após criar o novo projeto, basta copiar todos os arquivos da pasta Projeto1C18.X e colar dentro do novo Projeto2.X. Pronto, basta abrir o Projeto2.X e compilar o arquivo em *Source files*.

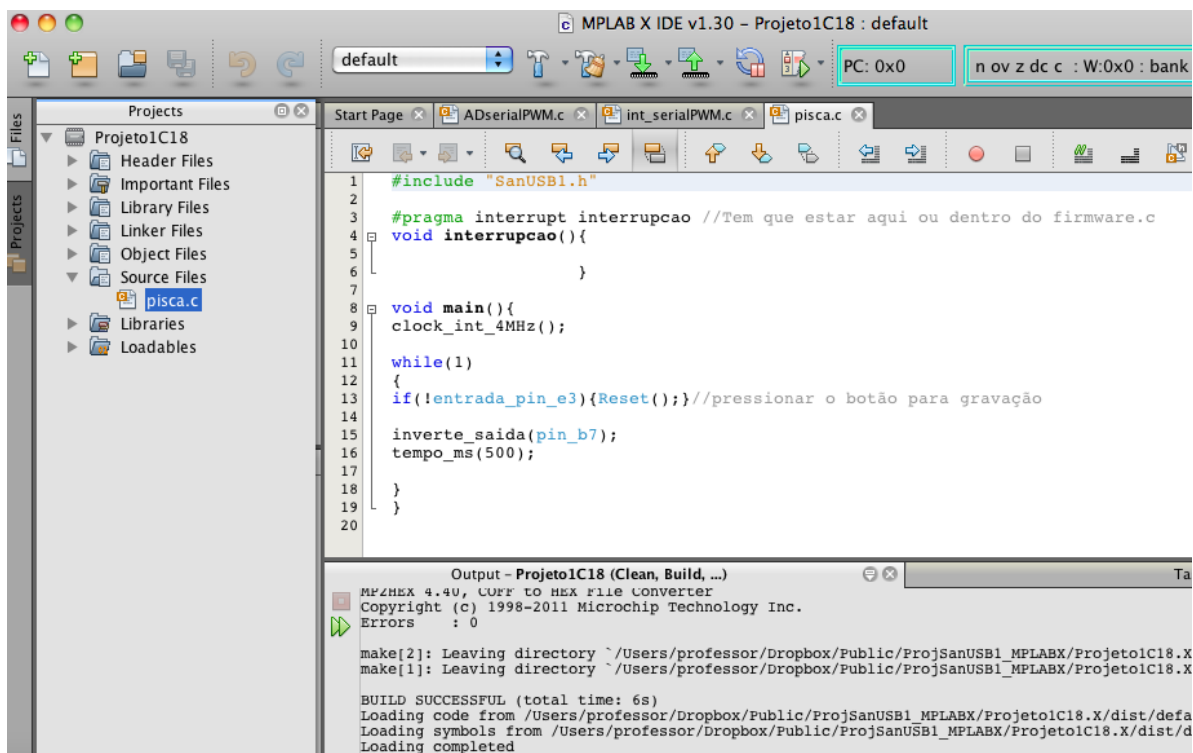


Figura 16. : Projeto pisca LED no compilador C18.

FUNÇÕES EM PORTUGUÊS

Este capítulo descreve todas as funções em português da biblioteca SanUSB no C18. É importante salientar que além dessas funções, são válidas as funções padrões ANSI C e também que as funções do compilador C18 estão descritas em código aberto dentro na biblioteca SanUSB1.h. A fim de facilitar o entendimento, as funções SanUSB foram divididas em grupos, definidos por sua utilização e os periféricos do hardware que estão relacionadas.

FUNÇÕES BÁSICAS DA APLICAÇÃO DO USUÁRIO

Este grupo de funções define a estrutura do programa uma vez que o usuário deve escrever o programa.c de sua aplicação.

O microcontrolador possui um recurso chamado *watchdog timer* (wdt) que nada mais é do que um temporizador cão-de-guarda contra travamento do programa. Caso seja habilitado `habilita_wdt()`; na função principal `main()`, este temporizador está configurado para contar aproximadamente um intervalo de tempo de 16 segundos. Ao final deste intervalo, se a *flag* `limpa_wdt()`; não for zerada, ele provoca um reset do microcontrolador e consequentemente a reinicialização do programa. A aplicação deve permanentemente zerar a *flag* `limpa_wdt()` dentro do laço infinito (`while(1){}`) da função principal `main()` em intervalos de no máximo 16 segundos. Este recurso é uma segurança contra qualquer

possível falha que venha travar o programa e paralisar a aplicação. Para zerar o wdt, o usuário pode também utilizar a função `ClrWdt()`; do compilador C18.

A seguir estão as características detalhadas destas funções.

clock_int_4MHz()

Função: Habilita o clock para a processador do oscilador interno de 4MHz.

Argumentos de entrada: Não há.

Argumentos de saída: Não há.

Observações: O *clock* padrão proveniente do sistema USB interno do PIC é de 48MHz gerado a partir do cristal de 20 MHz. Isto é possível através de um multiplicador interno de *clock* do PIC. A função `_int_4MHz()` habilita, para o processador do microcontrolador, o oscilador RC interno em 4 MHz que adéqua o período de incremento dos temporizadores em 1us. É aconselhável que seja a primeira declaração da função principal *main()*.Exemplo:

```
#include "SanUSB1.h"
```

```
void main (void) {
```

```
    clock_int_4MHz();
```

A seguir estão as características detalhadas destas funções.

clock_int_48MHz()

Função: Habilita o clock para a processador do oscilador interno de 4MHz.

Observações: O *clock* padrão, proveniente do sistema USB interno do PIC é de 48MHz gerado a partir do cristal de 20 MHz, é utilizado também pela CPU:

```
#include "SanUSB48.h"
```

```
void main (void) {
```

```
    clock_int_48MHz();
```

nivel_alto()

Função: Força nível lógico alto (+5V) em uma saída digital.

Argumentos de entrada: Nome da saída digital que irá para nível lógico alto. Este nome é construído pelo início `pin_` seguido da letra da porta e do número do pino. Pode ser colocado também o nome de toda a porta, como por exemplo, `portb`.

Argumentos de saída: Não há.

Observações: Não há.

Exemplo:

```
nivel_alto(pin_b7); //Força nível lógico 1 na saída do pino B7
```

```
nivel_alto(portb); //Força nível lógico 1 em toda porta b
```

nivel_baixo()

Função: Força nível lógico baixo (0V) em uma saída digital.

Argumentos de entrada: Nome da saída digital que irá para nível lógico baixo. Este nome é construído pelo início pin_ seguido da letra da porta e do número do pino. Pode ser colocado também o nome de toda a porta, como por exemplo, portc.

Argumentos de saída: Não há.

Observações: Não há.

Exemplo:

```
nivel_baixo(pin_b7); //Força nível lógico 0 na saída do pino B7
```

```
nivel_baixo(portc); //Força nível lógico 0 em toda porta c
```

saída_pino(pino,booleano)

Função: Acende um dos leds da placa McData.

Argumentos de entrada: Pino que irá receber na saída o valor booleano, valor booleano 0 ou 1.

Argumentos de saída: Não há.

Observações: Não há.

Exemplo:

```
ledpisca=!ledpisca;           // ledpisca é igual ao inverso de ledpisca
```

```
saida_pino(pin_b0,ledpisca);   // b0 recebe o valor de ledpisca
```

inverte_saida()

Função: Força nível lógico inverso em uma saída digital.

Argumentos de entrada: Nome da saída digital que irá ser invertida. Este nome é construído pelo início pin_ seguido da letra da porta e do número do pino.

Argumentos de saída: Não há.

Observações: Não há.

Exemplo:

```
inverte_saida(pin_b7); //Força nível lógico inverso na saída do pino B7
```

saída_pino(pino,booleano)

Função: Acende um dos leds da placa McData.

Argumentos de entrada: Pino que irá receber na saída o valor booleano, valor booleano 0 ou 1.

Argumentos de saída: Não há.

Observações: Não há.

Exemplo:

```
ledpisca=!ledpisca;           // ledpisca é igual ao inverso de ledpisca  
saida_pino(pin_b0,ledpisca);  // b0 recebe o valor de ledpisca
```

tempo_us()

Função: Tempo em múltiplos de 1us.

Argumentos de entrada: Tempo de tempo que multiplica 1 us.

Argumentos de saída: Não há.

Observações: Esta instrução só é finalizada ao final do tempo determinado, ou seja, esta função “paralisa” a leitura do programa durante a execução. Exemplo:

```
tempo_us(200);    //Tempo de 200 us
```

tempo_ms()

Função: Tempo em múltiplos de 1 ms.

Argumentos de entrada: Tempo de tempo que multiplica 1 ms.

Argumentos de saída: Não há.

Observações: Esta instrução só é finalizada ao final do tempo determinado, ou seja, esta função “paralisa” a leitura do programa durante a execução. Exemplo:

```
tempo_ms(500);    //Tempo de 500 ms
```

entrada_pin_xx

Função: Lê nível lógico de entrada digital de um pino.

Argumentos de entrada: Não há.

Observações: Este nome é construído pelo início `entrada_pin_` seguido da letra da porta e do número do pino.

Exemplo:

```
ledXOR = entrada_pin_b1^entrada_pin_b2;           //OU Exclusivo entre as entradas dos
pinos b1 e b2
```

habilita_interrupcao()

Função: Habilita as interrupções mais comuns do microcontrolador na função *main()*.

Argumentos de entrada: Tipo de interrupção: timer0, timer1, timer2, timer3, ext0, ext1, ext2, ad e recep_serial.

Argumentos de saída: Não há.

Observações: As interrupções externas já estão habilitadas com borda de descida. Caso se habilite qualquer interrupção deve-se inserir o desvio `_asm goto interrupcao _endasm` na função `void _high_ISR (void){ }` da biblioteca SanUSB.h

Exemplo:

```
habilita_interrupcao(timer0);
```

```
habilita_interrupcao(ext1);
```

if(xxx_interrompeu)

Função: Flag que verifica, dentro da função de tratamento de interrupções, se uma interrupção específica ocorreu.

Complemento: timer0, timer1, timer2, timer3, ext0, ext1, ext2, ad e serial.

Argumentos de saída: Não há.

Observações: A flag deve ser zerada dentro da função de interrupção.

Exemplo:

```
#programa interrupt interrupcao
```

```
void interrupcao()
```

```
{
```

```
    if (ext1_interrompeu)    {           //espera a interrupção externa 1 (em B1)
```

```
    ext1_interrompeu = 0;           //limpa a flag de interrupção
```

```
    PORTBbits.RB0 =! PORTBbits.RB0;} //inverte o LED em B0
```

```

if (timer0_interrompeu)      {      //espera o estouro do timer0

timer0_interrompeu = 0;      //limpa a flag de interrupção

PORTBbits.RB0 =! PORTBbits.RB0; //inverte o LED em B7

tempo_timer16bits(0,62500); } }

```

liga_timer16bits(timer,multiplicador)

Função: Liga os timers e ajusta o multiplicador de tempo na função *main()*.

Argumentos de entrada: Timer de 16 bits (0,1 ou 3) e multiplica que é o valor do prescaler para multiplicar o tempo.

Argumentos de saída: Não há.

Observações: O timer 0 pode ser multiplicado por 2, 4, 6, 8, 16, 32, 64, 128 ou 256. O Timer 1 e o Timer 3 podem ser multiplicados por 1, 2, 4 ou 8.

Exemplo:

```
liga_timer16bits(0,16); //Liga timer 0 e multiplicador de tempo igual a 16
```

```
liga_timer16bits(3,8); //Liga timer 0 e multiplicador de tempo igual a 8
```

tempo_timer16bits(timer,conta_us)

Função: Define o timer e o tempo que será contado em us até estourar.

Argumentos de entrada: Timer de 16 bits (0,1 ou 3) e tempo que será contado em us (valor máximo 65536).

Argumentos de saída: Não há.

Observações: O Não há.

Exemplo:

```
habilita_interrupcao(timer0);
```

```
liga_timer16bits(0,16);      //liga timer0 - 16 bits com multiplicador (prescaler) 16
```

```
tempo_timer16bits(0,62500);    //Timer 0 estoura a cada 16 x 62500us = 1 seg.
```

habilita_wdt()

Função: Habilita o temporizador cão-de-guarda contra travamento do programa.

Argumentos de entrada: Não há.

Argumentos de saída: Não há.

Observações: O *wdt* inicia como padrão sempre desabilitado. Caso seja habilitado na função principal *main()*, este temporizador está configurado para contar aproximadamente um intervalo de tempo de 16 segundos. Ao final deste intervalo, se a *flag* *limpa_wdt()* não

for zerada, ele provoca um reset do microcontrolador e conseqüentemente a reinicialização do programa. Exemplo:

```
#include "SanUSB1.h"
```

```
void main (void) {  
    clock_int_4MHz();  
    habilita_wdt();    //Habilita o wdt
```

limpaflag_wdt()

Função: limpa a flag do wdt

Argumentos de entrada: Não há.

Argumentos de saída: Não há.

Observações: Caso o *wdt* seja habilitado, a *flag* deve ser limpa em no máximo 16 segundos para que não haja reinicialização do programa. Geralmente esta função é colocada dentro do laço infinito *while(1)* da função principal *main()*. É possível ver detalhes no programa *exemplowdt.c* e utilizar também a função *ClrWdt()* do compilador C18 .

Exemplo:

```
#include "SanUSB1.h"
```

```
#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
```

```
void interrupcao(){}  
  
void main (void) {  
    clock_int_4MHz();  
    habilita_wdt();  
    while(1) {        limpaflag_wdt();  
        .....  
        tempo_ms(500);  
    }  
}
```

escreve_eeprom(posição,valor)

Função: Escrita de um byte da memória EEPROM interna de 256 bytes do microcontrolador.

Argumentos de entrada: Endereço da memória entre 0 a 255 e o valor entra 0 a 255.

Argumentos de saída: Não há.

Observações: O resultado da leitura é armazenado no byte EEDATA.

Exemplo:

```
escreve_eeprom(85,09); //Escreve 09 na posição 85
```

le_eeprom()

Função: Leitura de um byte da memória EEPROM interna de 256 bytes do microcontrolador.

Argumentos de entrada: Endereço da memória entre 0 a 255.

Argumentos de saída: Não há.

Observações: O resultado da leitura é armazenado no byte EEDATA.

Exemplo:

```
dado=le_eeprom(85);
```

FUNÇÕES DO CONVERSOR ANALÓGICO DIGITAL (A/D)

As funções a seguir são utilizadas para a aquisição de dados utilizando as entradas analógicas.

habilita_canal_AD()

Função: Habilita entradas analógicas para conversão AD.

Argumentos de entrada: Número do canal analógico que irá ser lido. Este dado habilita um ou vários canais AD e pode ser AN0, AN0_a_AN1 , AN0_a_AN2 , AN0_a_AN3, AN0_a_AN4, AN0_a_AN8, AN0_a_AN9, AN0_a_AN10, AN0_a_AN11, ou AN0_a_AN12.

Argumentos de saída: Não há.

Observações: Não há.

Exemplo:

```
habilita_canal_AD(AN0); //Habilita canal 0
```

le_AD8bits()

Função: Leitura de uma entrada analógica com 8 bits de resolução.

Prototipagem: unsigned char analog_in_8bits(unsigned char).

Argumentos de entrada: Número do canal analógico que irá ser lido. Este número pode

ser 0, 1, 2, 3, 4, 8, 9, 10, 11 ou 12.

Argumentos de saída: Retorna o valor da conversão A/D da entrada analógica com resolução de 8 bits.

Observações: Não há.

Exemplo:

```
PORTB = le_AD8bits(0); //Lê canal 0 da entrada analógica com resolução de 8 bits e coloca na porta B
```

le_AD10bits()

Função: Leitura de uma entrada analógica com 10 bits de resolução.

Prototipagem: unsigned char analog_in_10bits(unsigned char).

Argumentos de entrada: Número do canal analógico que irá ser lido. Este número pode ser 0, 1, 2, 3, 4, 8, 9, 10, 11 ou 12.

Argumentos de saída: Retorna o valor da conversão A/D da entrada analógica com resolução de 10 bits.

Observações: Não há.

Exemplo:

```
resultado = le_AD10bits(0); //Lê canal 0 da entrada analógica com resolução de 10 bits.
```

SetaPWM1(int freqPWM, int duty);

Função: Seta a frequência e o ciclo de trabalho do pino de PWM1 (pin_c2).

Exemplo: for(i = 0 ; i < 100 ; i=i+5) {

```
SetaPWM1(10000, i);SetaPWM2(10000, i);
```

```
tempo_ms(500);
```

```
}
```

Exemplos de aplicação:

<https://www.youtube.com/watch?v=IB21b3zA4Ac>

<https://www.youtube.com/watch?v=KbH3yzPHX4U>

FUNÇÕES DA COMUNICAÇÃO SERIAL RS-232

As funções a seguir são utilizadas na comunicação serial padrão RS-232 para enviar e receber dados, definir a velocidade da comunicação com o oscilador interno 4MHz.

As configurações da comunicação são: sem paridade, 8 bits de dados e 1 stop bit. Esta

configuração é denominada 8N1 e não pode ser alterada pelo usuário.

taxa_serial();

Função: Configura a taxa de transmissão/recepção (*baud rate*) da porta RS-232

Argumentos de entrada: Taxa de transmissão/recepção em bits por segundo (bps)

Argumentos de saída: Não há.

Observações: O usuário deve obrigatoriamente configurar taxa_rs232() da comunicação assíncrona antes de utilizar as funções le_serial e escreve_serial. As taxas programáveis são 1200 bps, 2400 bps, 9600 bps, 19200 bps.

Exemplo:

```
void main() {  
    clock_int_4MHz();  
    habilita_interrupcao(recep_serial);  
    taxa_rs232(2400); // Taxa de 2400 bps  
    while(1); // programa normal parado aqui }
```

le_serial();

Função: Lê o primeiro caractere recebido que está no buffer de recepção RS-232.

Argumentos de entrada: Não há.

Argumentos de saída: Não há.

Observações: Quando outro byte é recebido, ele é armazenado na próxima posição livre do buffer de recepção, cuja capacidade é de 16 bytes. Exemplo:

```
#pragma interrupt interrupcao
```

```
void interrupcao()
```

```
{ unsigned char c;
```

```
    if (serial_interrompeu) {
```

```
        serial_interrompeu=0;
```

```
        c = le_serial();
```

```
        if (c >= '0' && c <= '9')    { c -= '0';    PORTB = c;} }}
```

sendsw((char *));

Função: Transmite caracteres pela serial UART.

Argumentos de entrada: O dado a ser transmitido deve ser de 8 bits do tipo char.

Argumentos de saída: Não há.

Exemplo: const char nome [] ="Serial ";

```
sendsw((char *)nome); // escreve Serial
```

```
tempo_ms(300);
```

```
sendsw((char *)nome);// escreve Serial
```

```
tempo_ms(300);
```

Caso ocorra o erro ***sanusb Error: Odd address at beginning of HEX file line error***, compile e grave o firmware básico pisca.c e tente novamente compilar e gravar o firmware desejado. ***Evitar o uso da função printf()***.

sendnum();

Função: Transmite números de variáveis pela serial UART.

Argumentos de entrada: O dado a ser transmitido deve ser de 8 bits do tipo char.

Argumentos de saída: Não há.

Exemplo: const char nome [] ="Valor= "; unsigned int ADvalue;

```
ADvalue=le_AD10bits(0);
```

```
sendsw((char *)nome);
```

```
sendnum(ADvalue);
```

```
tempo_ms(300);
```

Caso ocorra o erro ***sanusb Error: Odd address at beginning of HEX file line error***, compile e grave o firmware básico pisca.c e tente novamente compilar e gravar o firmware desejado. ***Evitar o uso da função printf()***.

FERRAMENTA DE GRAVAÇÃO VIA USB

O sistema de desenvolvimento *SanUSB* é uma ferramenta composta de *software* e *hardware* básico da família PIC18Fxx5x com interface USB. Esta ferramenta livre se mostra eficiente no desenvolvimento rápido de projetos reais, pois não há necessidade de remover o microcontrolador para a atualização do firmware. Além disso, esta ferramenta se mostra eficaz no ensino e na difusão de microcontroladores, bem como em projetos de eletrônica e informática, pois todos os usuários podem desenvolver projetos reais no ambiente de ensino ou na própria residência sem a necessidade de um equipamento para gravação de microcontroladores. Além disso, o software de gravação de microcontroladores USB é multiplataforma, pois é executável no Windows®, Mac OSX e no Linux e também *plug and play*, ou seja, é reconhecido automaticamente pelos sistemas operacionais sem a necessidade de instalar nenhum *driver*. Dessa forma, ela é capaz de suprimir:

- 1- Um equipamento específico para gravação de um programa no microcontrolador;
- 2- conversor TTL - RS-232 para comunicação serial bidirecional, emulado via USB pelo protocolo CDC, que permite também a depuração do programa através da impressão via USB das variáveis do *firmware*;
- 3- fonte de alimentação, já que a alimentação do PIC provém da porta USB do PC. *É importante salientar que cargas indutivas como motores de passo ou com corrente acima de 400mA devem ser alimentadas por uma fonte de alimentação externa.*
- 4- Conversor analógico-digital (AD) externo, tendo em vista que ele dispõe internamente de **10** ADs de 10 bits;
- 5- *software* de simulação, considerando que a simulação do programa e do *hardware* podem ser feitas de forma rápida e eficaz no próprio circuito de desenvolvimento ou com um *protoboard* auxiliar.

Além de todas estas vantagens, os *laptops* e alguns computadores atuais não apresentam mais interface de comunicação paralela e nem serial EIA/RS-232, somente USB.

Como pode ser visto, esta ferramenta possibilita que a compilação, a gravação e a simulação real de um programa, como também a comunicação serial através da emulação de uma porta COM sem fio, possam ser feitos de forma rápida e eficaz a partir do momento em o microcontrolador esteja conectado diretamente a um computador via USB.

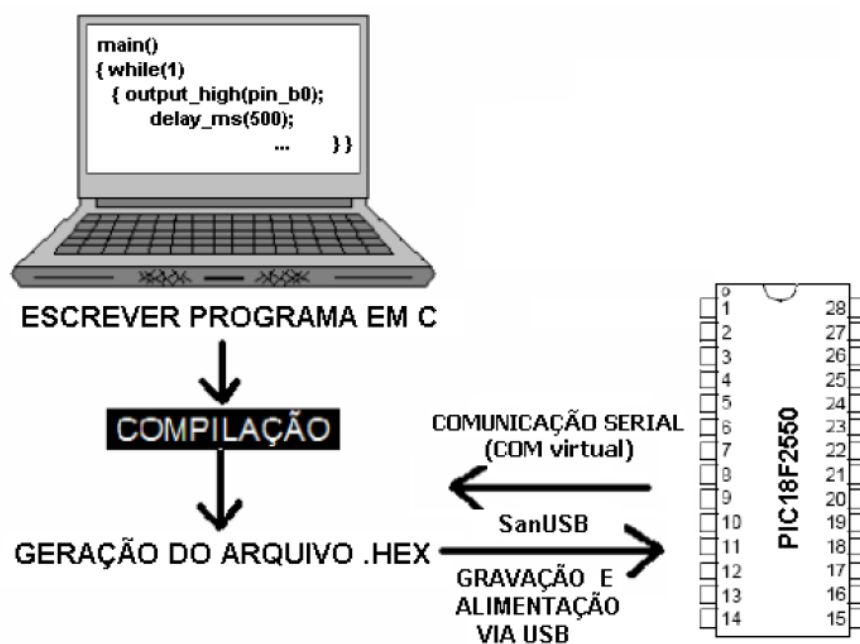


Figura 2. : Gravação do PIC via PC.

Utilizando esta ferramenta, estudantes foram três vezes consecutivas campeões da Competição de Robótica do IFCE (2007, 2008 e 2009) na categoria Localização, campeões da Feira Brasileira de Ciências e Engenharia (FEBRACE09) da USP em São Paulo na Categoria Engenharia (2009), como também obtiveram Prêmio de Inovação em Aplicação Tecnológica na FERIA Explora 2009 em Medellín na Colômbia e foram Campeões na Categoria Supranível do Foro Internacional de Ciencia e Ingenieria 2010 no Chile, terceiro lugar em inovação na Semantec 2011 do IFCE e campeões na V Feira Estadual de Ciências e Cultura do Ceará na categoria robótica educacional em 2011.

2.1 GRAVAÇÃO DE MICROCONTROLADORES

A transferência de programas para os microcontroladores é normalmente efetuada através de um hardware de gravação específico. Através desta ferramenta, é possível efetuar a descarga de programas para o microcontrolador diretamente de uma porta USB de qualquer PC.

Para que todas essas funcionalidades sejam possíveis, é necessário gravar, anteriormente e somente uma vez, com um gravador específico para PIC, o gerenciador de gravação pela USB Gerenciador.hex disponível na pasta completa da ferramenta no link abaixo, onde também é possível baixar periodicamente as atualizações dessa ferramenta e a inclusão de novos programas: <https://dl.dropbox.com/u/101922388/121007SanUSBOrig.zip>

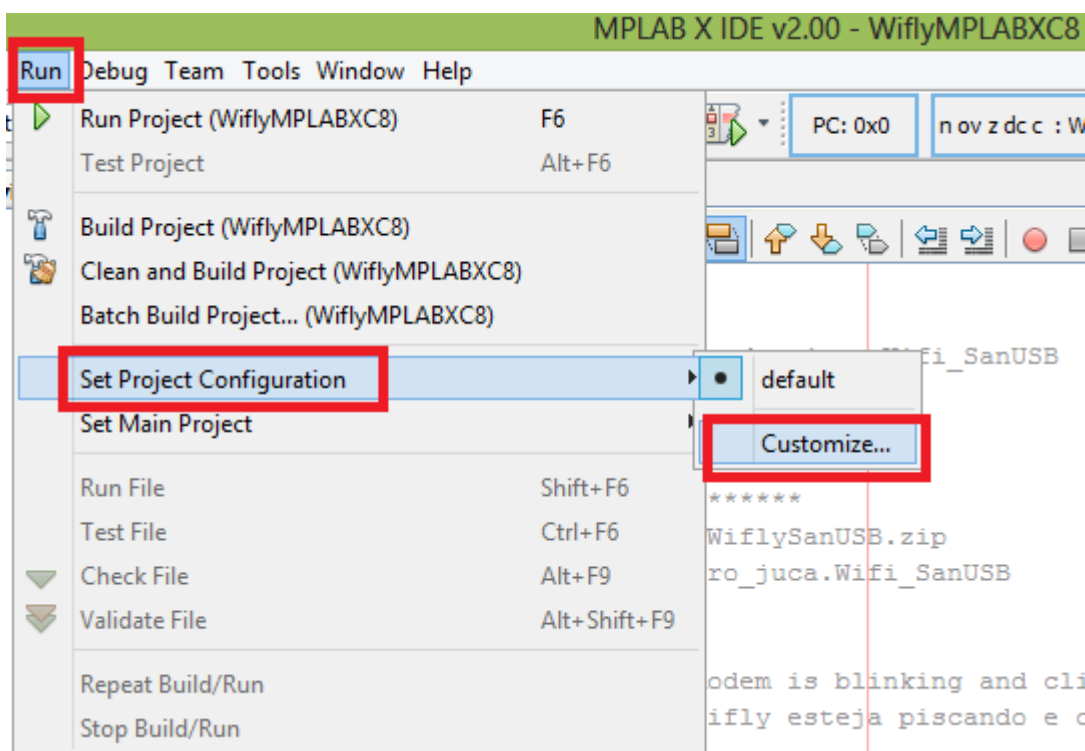
Caso o computador ainda não o tenha o aplicativo Java JRE ou SDK instalado para suporte a programas executáveis desenvolvidos em Java, baixe a Versão Windows@ disponível em: <http://www.4shared.com/file/WKDhQwZK/jre-6u21-Windows@-i586-s.html> ou através do link: http://www.java.com/pt_BR/download/manual.jsp.

Para que os programas em C possam ser gravados no microcontrolador via USB, é necessário compilá-los, ou seja, transformá-los em linguagem de máquina hexadecimal. Existem diversos compiladores que podem ser utilizados por esta ferramenta, entre eles o SDCC, o MPLABXX C18, o Hi-Tech e o CCS. Para compilar com o **MPLABX + C18 Lite** e a placa SanUSB em Linux, Windows® ou Mac OSX é simples. Inicialmente, basta instalar normalmente o MPLABX e o C18 Lite para o S.O. Desejado.

Compilando com XC8 e gravando o microcontrolador via USB na Placa SanUSB

Após instalar o MPLABX IDE v.2.00 e o compilador XC8 v.1.31 ou superior em <https://drive.google.com/?tab=wo&authuser=0#folders/0B5332OAhMe2STE4MFNBQ2dqYWM> descompacte o arquivo .zip e abra, no MPLABX, o projeto disponível em https://dl.dropboxusercontent.com/u/101922388/ProjSanUSB1_MPLABX/Projeto1XC8.X.zip. Para possibilitar a transferência do firmware, via USB com XC8 para a placa SanUSB, configure o MPLABX da seguinte forma:

Se for a primeira vez, acesse *Customize*:



Modifique no XC8 linker, o *code offset* em *Additional options* para 0x1000 e o *memory model* para *default,-0-FFF,-1006-1007,-1016-1017* como abaixo.

×

Project Properties - WiflyMPLABXC8

×

Categories:

- General
- Conf: [default]**
 - PICkit 2
 - Loading
 - Libraries
 - Building
 - XC8 global options
 - XC8 compiler
 - XC8 linker**

Options for xc8 (v1.31)

Option categories: **Additional options** Reset

Extra Linker Options	
Serial	
Codeoffset	0x1000
Checksum	
Errata	
Trace type	(N/A)
Extend address 0 in HEX file	<input type="checkbox"/>
Use response file to link	<input type="checkbox"/>

Additional options:

Option Description

Generated Command Line

Manage Configurations...

×

Project Properties - WiflyMPLABXC8

×

Categories:

- General
- Conf: [default]**
 - PICkit 2
 - Loading
 - Libraries
 - Building
 - XC8 global options
 - XC8 compiler
 - XC8 linker**

Options for xc8 (v1.31)

Option categories: **Memory model** Reset

Size of Double	24 bit
Size of Float	24 bit
External memory	Wordwrite
RAM ranges	
ROM ranges	default,-0-FFF,-1006-1007,-1016-1017

Additional options:

Option Description

Generated Command Line

Manage Configurations...

OK

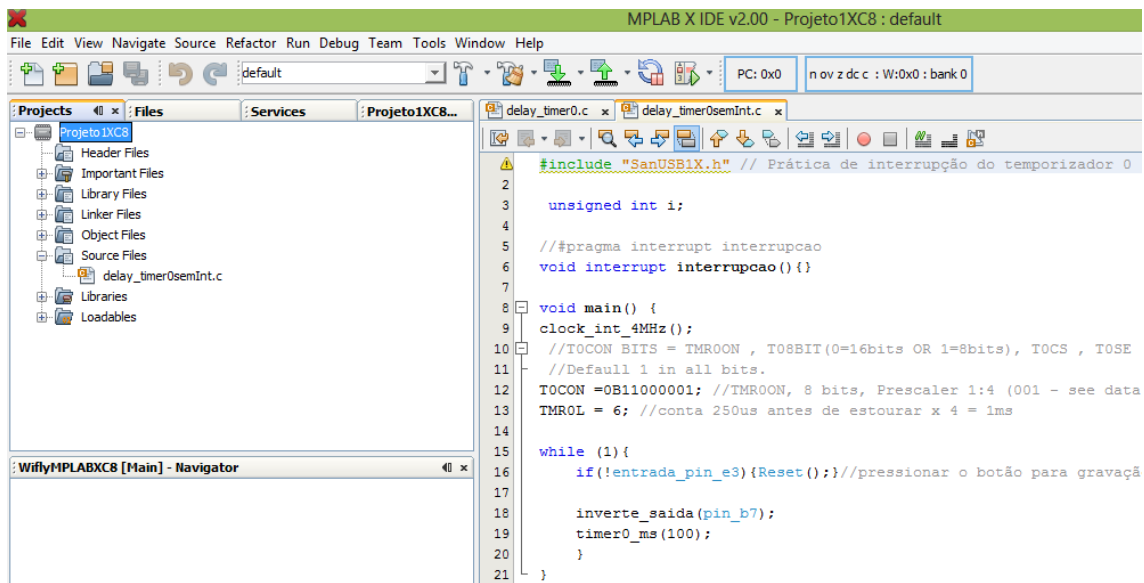
Cancel

Apply

Unlock

Help

Pronto. Agora basta selecionar um arquivo.c em *Source Files* e clicar *Clean and Build Main Project* para compilar.



Depois, basta selecionar o *Projeto1XC8.X.production.hex* e gravar no microcontrolador via USB:

