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

1) Instale o MPLABX e o compilador C18. É possível baixar gratuitamente o MPLABX e a versão C18 Lite para o sistema operacional desejado nos links abaixo. 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

   Compilador Windows:
   http://www.4shared.com/get/7IOxpxIa/mplabx-ide-v110-windows-instal.html
   http://www.4shared.com/get/zNr7oYwX/mplabc18-v340-windows-lite-ins.html

   Compilar Linux:
   http://www.4shared.com/get/eThaIjBM/mplabx-ide-v141-linux-installe.html
   http://www.4shared.com/get/DYhGwftJ/mplabc18-v340-linux-full-insta.html

   Compilador Mac OSX:
   http://www.4shared.com/get/XXEP5oc8/mplabx-ide-v141-osx-installer.html
   http://www.4shared.com/get/eLoh2Vzl/mplabc18-v340.html

2) 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).

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)*.

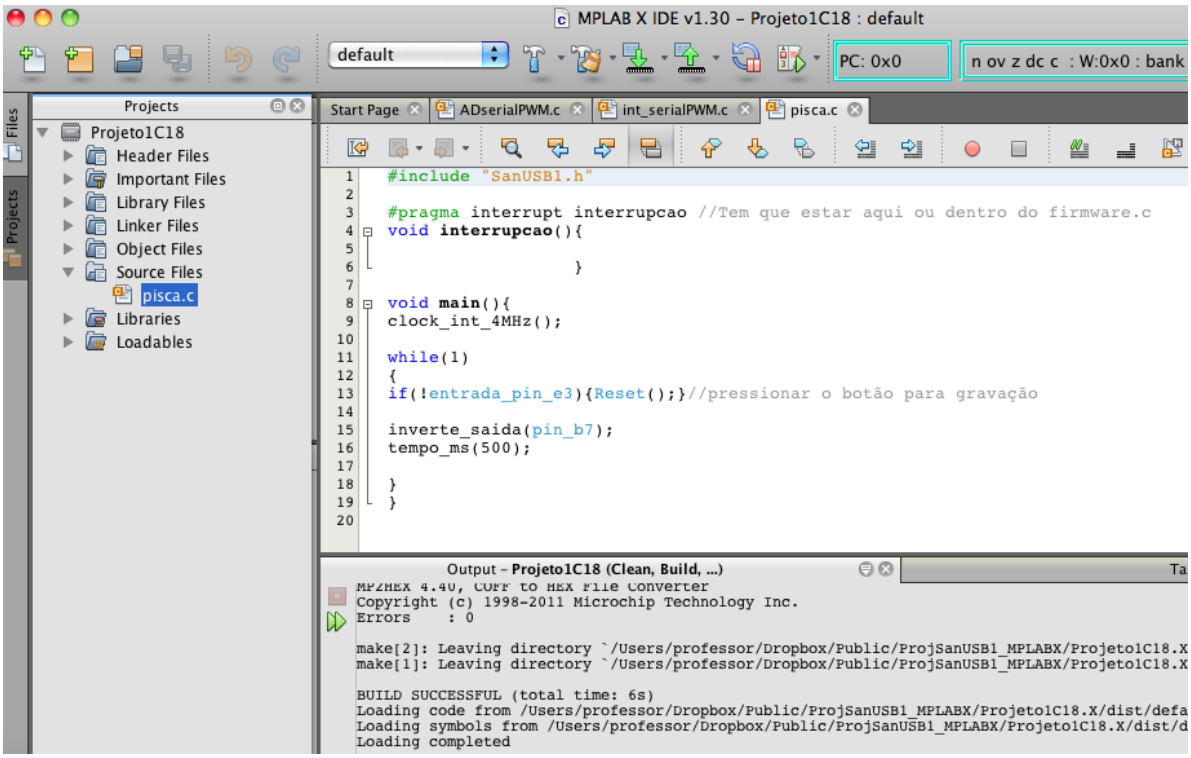

Figura 16. 1: Projeto pisca LED no compilador C18.

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 as funções do compilador C18 detalhadas na pasta instalada C:\MCC18\doc. A fim de facilitar o entendimento, as funções SanUSB foram divididas em oito 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 conseqüentemente a reinicialização do programa. A aplicação deve permanentemente zerar a *flag* limpa_wdt() dentro do laço infinito (while(1)) na 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();
```

nivel_alto()

Função: Força nivel lógico alto (+5V) em uma saída digital.
Argumentos de entrada: Nome da saída digital que irá para nivel lógico alto. Este nome é construído pelo inicio 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 nivel lógico 1 na saída do pino B7
nivel_alto(portb); //Força nivel lógico 1 em toda porta b
```

nivel_baixo()

Função: Força nivel lógico baixo (0V) em uma saída digital.
Argumentos de entrada: Nome da saída digital que irá para nivel lógico baixo. Este nome é construído pelo inicio 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 nivel lógico 0 na saída do pino B7
```

nivel_baixo(portc); //Força nivel 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
```

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 nivel lógico de entrada digital de um pino.
Argumentos de entrada: Não há.
Observações: Este nome é construído pelo inicio 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 interrução 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 (0,1ou 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 (0,1ou 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 reinicializaçção 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”

        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 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 10 bits.
Observações: Não há.
Exemplo:

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

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;}  }}
```

escreve_serial();

Função: Transmite um byte pela RS-232.
Argumentos de entrada: O dado a ser transmitido deve ser de 8 bits do tipo char.
Argumentos de saída: Não há.
Observações: A função escreve_serial não aguarda o fim da transmissão do byte. Como não existe um buffer de transmissão o usuário deve garantir a transmissão com a função envia_byte( ) para enviar o próximo byte.
Exemplo:

```
        escreve_serial('SanUSB'); // escreve SanUSB
        while (envia_byte());
```

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.

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.

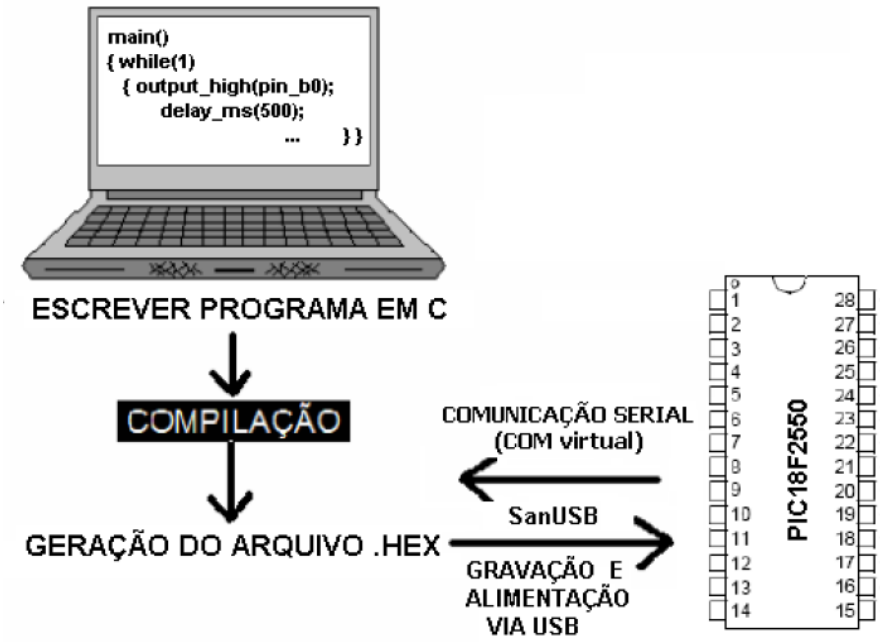

Figura 2. 1: Gravação do PIC via PC.

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 Medelin na Colômbia e foram Campeões na Categoria Supranivel do Foro Internacional de Ciencia e Ingeniería 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 disponivel 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@ disponivel 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 **MPLAX + 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 (http://www.microchip.com/pagehandler/en-us/family/MPLABXx/#downloads).Depois de instalado basta abrir o MPLAX e clicar em Open project e escolher um projeto descompactado.X, como em https://dl.dropbox.com/u/101922388/PWM_AD_Serial.zip.Este projeto já pisca um led no pino B7 e faz a leitura do AD no pino A0 e envia pela serial como no video:http://www.youtube.com/watch?v=lB21b3zA4Ac.

Para modificar o programa exemplo, altere o PWM_AD_serial.c e clique em *Clean und Build Project* (ícone que tem um martelo e uma vassoura,)
O arquivo compilado Projeto1C18.hex, para gravação via USB, está sempre dentro de PWM_AD_Serial/Projeto1C18.X/dist/default/production

Este exemplo, bem como muitos outros, foram compilados em Linux, Windows e Mac OSX, e funcionou normalmente.

A representação básica do circuito *SanUSB* montado em *protoboard* é mostrada a seguir:

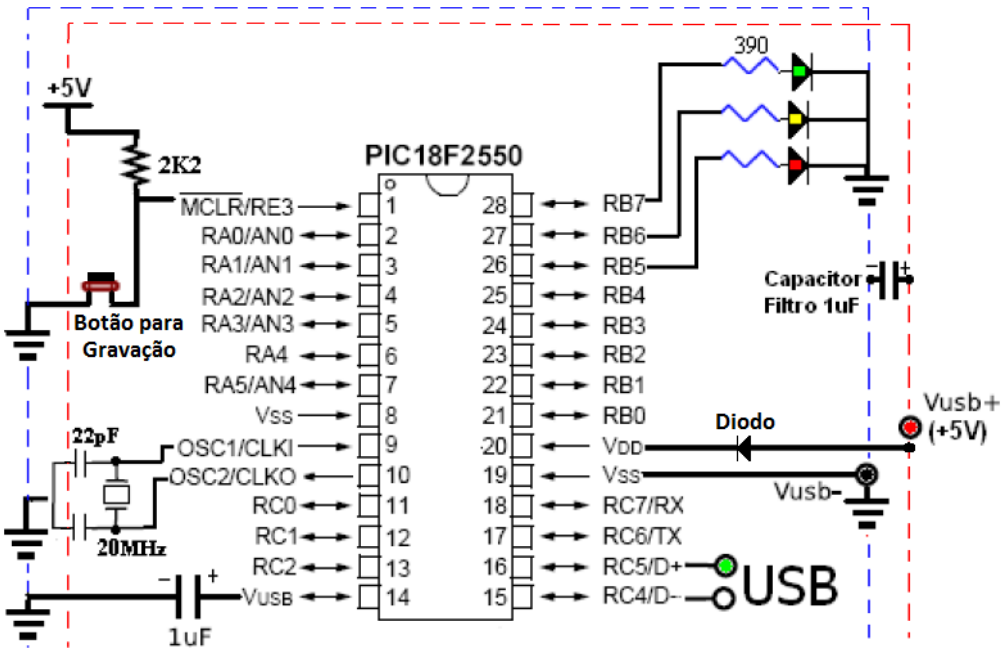

Figura 2. 2: Esquemático de montagem da Ferramenta para 28 pinos.

Para um microcontrolador de 40 pinos, o circuito é mostrado abaixo:

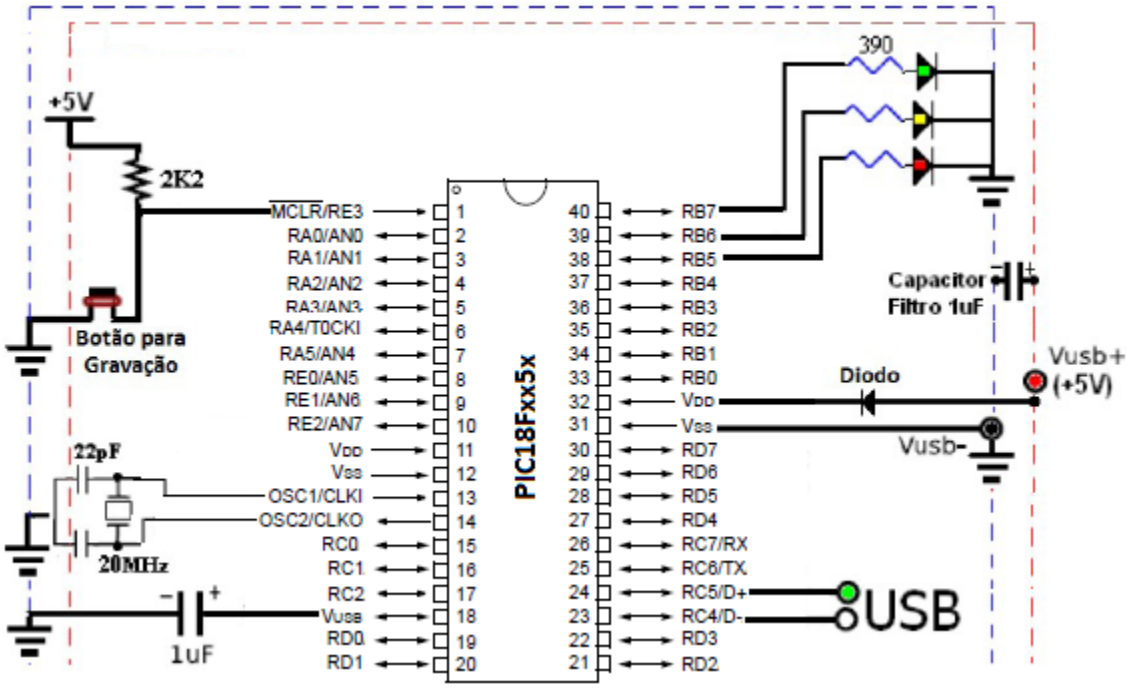

Figura 2. 3: Esquemático de montagem da ferramenta para 40 pinos.

Os componentes básicos do circuito são:

- 1 microcontrolador da família PIC USB (18F2550, 18F2455, 18F4550, etc.);
- 1 cristal de 20MHz;
- 2 capacitores de 22pF;
- 2 capacitores de 1uF (um no pino 14 Vusb e outro entre o +5V e o Gnd) ;
- 3 leds e 3 resistores de 390 (só é necessário um led com resistor no pino B7);
- 1 resistor de 2k2 e um botão ou fio para gravação no pino 1;
- 1 diodo qualquer entre o +5V e o o pino Vdd;
- 1 Cabo USB qualquer.

Note que, ***este sistema multiplataforma(Linux, Windows@ e Mac OSX), compatível com o software de gravação HID USB da Microchip também para Linux e Mac OSX, pode ser implementado também em qualquer placa de desenvolvimento de microcontroladores PIC com interface USB***, pois utiliza o botão de *reset*, no pino 1, como botão de gravação via USB. Ao conectar o cabo USB e alimentar o microcontrolador, com o pino 1 no Gnd (0V), através do botão ou de um simples fio, o microcontrolador entra em Estado para Gravação via USB (*led* no pino B7 aceso) e que, após o *reset* com o pino 1 no Vcc (+5V através do resistor fixo de 2K2 sem o *jump*), entra em Estado para Operação do programa aplicativo (*firmware*) que foi compilado.

O cabo USB apresenta normalmente quatro fios, que são conectados ao circuito do microcontrolador nos pontos mostrados na figura acima, onde normalmente, o fio Vcc (+5V) do cabo USB é vermelho, o Gnd (Vusb-) é marrom ou preto, o D+ é azul ou verde e o D- é amarelo ou branco. Note que a fonte de

alimentação do microcontrolador nos pinos 19 e 20 e dos barramentos vermelho (+5V) e azul (Gnd) do circuito provem da própria porta USB do computador. Para ligar o cabo USB no circuito é possível cortá-lo e conectá-lo direto no *protoboard*, com fios rígidos soldados, como também é possível conectar sem cortá-lo, em um *protoboard* ou numa placa de circuito impresso, utilizando um conector USB fêmea. O diodo de proteção colocado no pino 20 entre o Vcc da USB e a alimentação do microcontrolador serve para proteger contra corrente reversa caso a tensão da porta USB esteja polarizada de forma inversa.

A figura abaixo mostra a ferramenta SanUSB montada em *protoboard* seguindo o circuito anterior e a posição do apdaptador USB a ser ligado no PC via cabo. Você pode ligar de qualquer um dos lados do conector USB, observando a descrição.

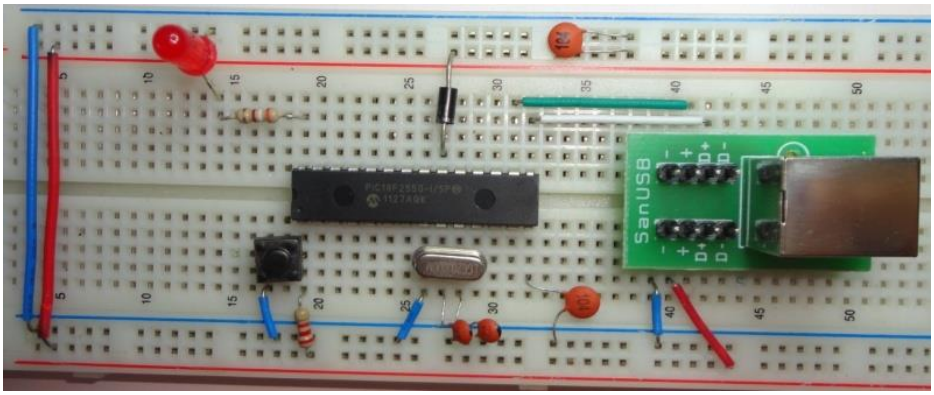

Figura 2. 4: Esquema montado em protoboard e conector USB.

É importante salientar que, para o perfeito funcionamento da gravação via USB, o circuito desta ferramenta deve conter um capacitor de filtro entre 0,1uf e 1uF na alimentação que vem da USB, ou seja, colocado entre os pinos 20 (+5V) e 19 (Gnd) ou no barramento + e – da protoboard..

Caso o sistema microcontrolado seja embarcado como, por exemplo, um robô, um sistema de aquisição de dados ou um controle de acesso, ele necessita de uma fonte de alimentação externa, que pode ser uma bateria comum de 9V ou um carregador de celular. A figura abaixo mostra o PCB, disponivel nos Arquivos do Grupo SanUSB, e o circuito para esta ferramenta com entrada para fonte de alimentação externa. Para quem deseja obter o sistema pronto para um aprendizado mais rápido, é possível também encomendar placas de circuito impresso da ferramenta SanUSB, como a foto da placa abaixo, entrando em contato com o grupo SanUSB através do e-mail: sanusb_laese@yahoo.com.br .

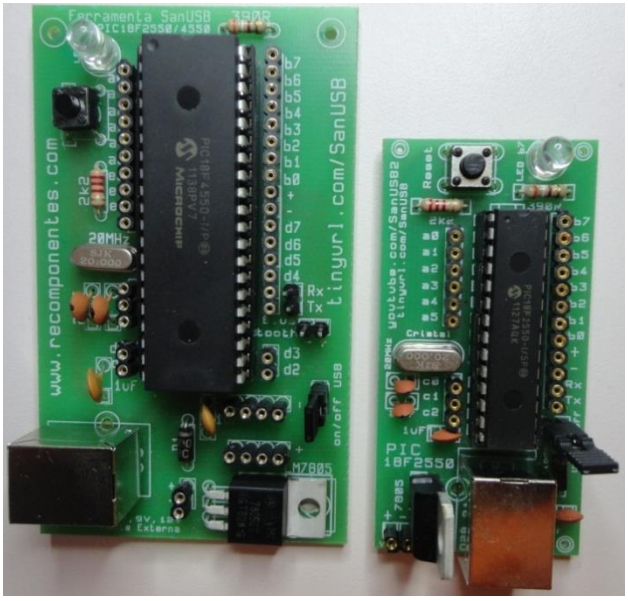

Figura 2. 5: Esquema montado em PCB.

Se preferir confeccionar a placa, é possível também imprimir, em folha de transparência, o *PCB* e o *silk* configurado em tamanho real, como mostra a figura 2.6, transferir para a placa de cobre, corroer, furar e soldar os componentes. Mais detalhes no vídeo disponivel em: **http://www.youtube.com/watch?v=Xm8YJ_XaGA8**.

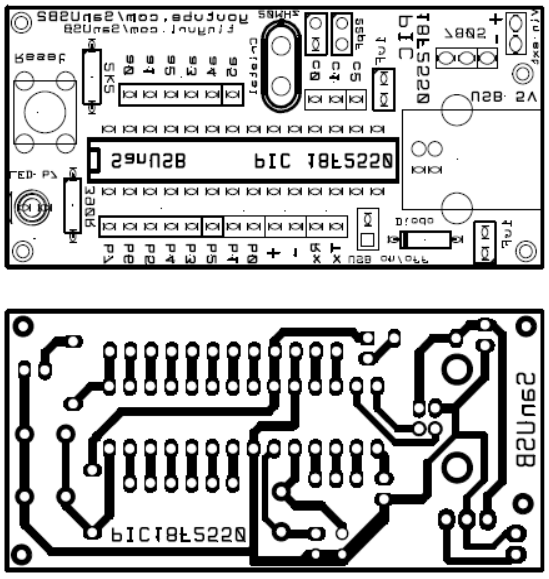

Figura 2. 6: PCB da Ferramenta SanUSB.

Para obter vários programas-fonte e vídeos deste sistema livre de gravação, comunicação e alimentação via USB, basta se cadastrar no grupo de acesso livre www.tinyurl.com/SanUSB e clicar no item Arquivos.

Durante a programação do microcontrolador basta abrir com o MPLABXX o projeto Projeto1.C18.X já configurado. A biblioteca SanUSB1.h contém funções básicas para o compilador, habilitação do sistema *Dual Clock*, ou seja, oscilador RC interno de 4 MHz para CPU e cristal oscilador externo de 20 MHz para gerar a frequência de 48MHz da comunicação USB, através de *prescaler* multiplicador de frequência.

Como a frequência do oscilador interno é de 4 MHz, cada incremento dos temporizadores corresponde a um microssegundo. O programa exemplo1 abaixo comuta um*led* conectado no pino B7 a cada 0,5 segundo.

-

PRÁTICA 1 – PISCA LED

Após montar o circuito da Ferramenta SanUSB (ver figura abaixo), deve-se iniciar a sequência de práticas.

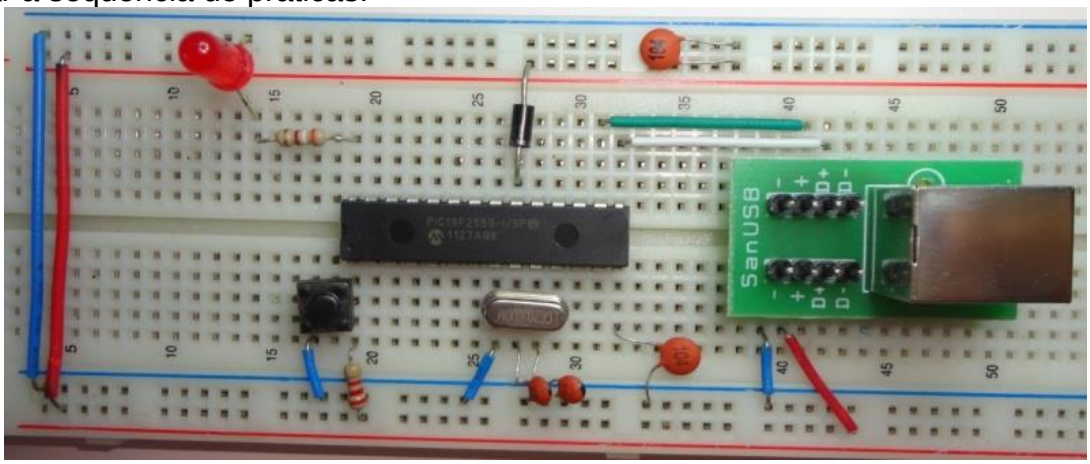

Figura 2. 7: Circuito básico da Ferramenta SanUSB.

Neste exemplo o objetivo é piscar um LED de forma temporizada a cada 0,5 segundos, sem o auxílio de chaves ou botões. Para isso utiliza-se uma única saída do PIC18F2550, que pode ser, por exemplo, o pino 28 (referência B7). Esta saída por sua vez está ligada ao Anodo de um LED com um resistor no valor de

100 ohm a 1k em série, como mostrado na Figura xx. O catodo do LED deve ser aterrado como na Figura.

Programação em Linguagem C:

```
#include "SanUSB1.h"
#pragma interrupt interrupcao //Tem que estar declarado no firmware.c
void interrupcao(){               }
void main(){
clock_int_4MHz();
while (1){//laço infinito

nivel_alto(pin_b7); //coloca a saída B7 em nivel lógico alto, ou seja, acende LED

tempo_ms(500);//aguarda 500 milissegundos = 0,5 segundos

nivel_baixo(pin_b7); //coloca a saída B7 em nivel lógico baixo, ou seja, apaga LED

tempo_ms(500); //aguarda 500 milissegundos = 0,5 segundos

} //fim while

} }//fim main
```

Como a frequência do oscilador interno é de 4 MHz, cada incremento dos temporizadores corresponde a um microssegundo. O programa exemplo1 abaixo comuta um led conectado no pino B7 a cada 0,5 segundo com a função inverte_saida().

```
#include "SanUSB1.h"

#pragma interrupt interrupcao
void interrupcao(){}

void main(){
clock_int_4MHz();//Função necessária para habilitar o dual clock (48MHz para
USB e 4MHz para CPU)

while (1)   {
   inverte_saida(pin_b7); // comuta Led na função principal
   tempo_ms(500);
   }}
```

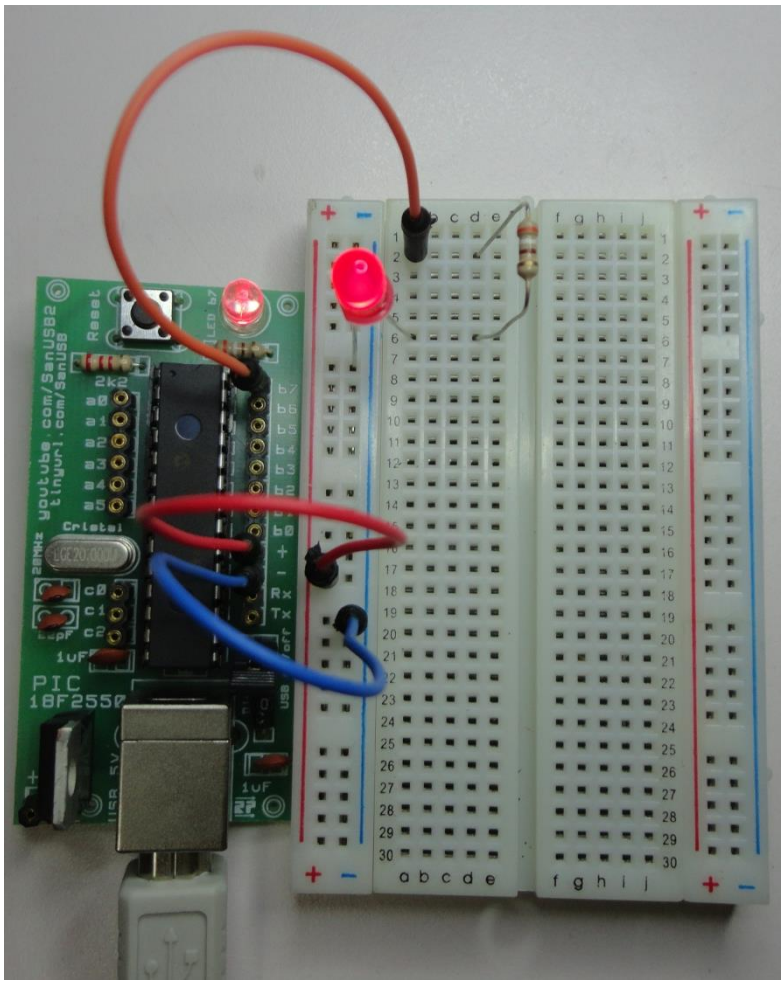

Figura 2.8: Prática 1 - Pisca LED, montada em protoboard.