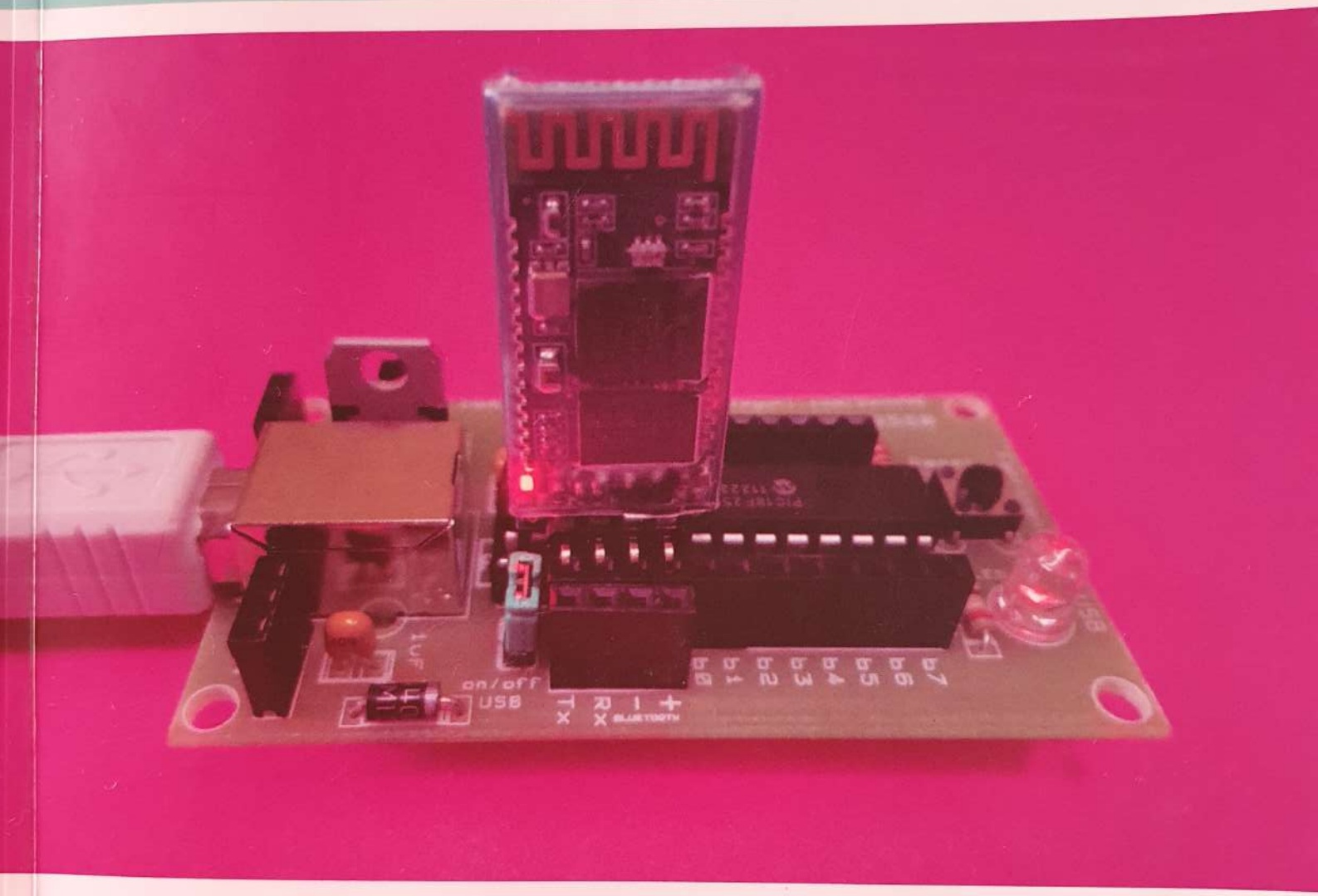
Aplicações práticas de Microcontroladores utilizando Software Livre



Aprenda de forma prática a gravação Wireless e via USB de microcontroladores através da Ferramenta SanUSB

Sandro Jucá e Renata Pereira

Aplicações práticas de Microcontroladores utilizando Software Livre

Aprenda de forma prática a gravação Wireless e via USB de microcontroladores através da Ferramenta SanUSB

Sandro Jucá e Renata Pereira

Aplicações práticas de Microcontroladores utilizando Software Livre

Aprenda de forma prática a gravação Wireless e via USB de microcontroladores através da Ferramenta SanUSB

2017

Aplicações práticas de Microcontroladores utilizando Software Livre

Impresso no Brasil / Printed in Brazil

Organizadores
Sandro Jucá e Renata Pereira

Revisado pelos autores

Editoração eletrônica:
Marta Braga

Impressão:
Imprima Soluções Gráficas Ltda.

J91a Jucá, Sandro
Aplicações práticas de microcontroladores utilizando software livre : aprenda de forma prática a gravação wireless e via USB de microcontroladores através da ferramenta SanUSB / Sandro Jucá e Renato Pereira. – Recife : Imprima, 2017.
198p. : il.

Inclui índice de figuras e tabelas.
Inclui referências.

1. SOFTWARE EDUCACIONAL. 2. INTELIGÊNCIA ARTIFICIAL – APLICAÇÕES EDUCACIONAIS. 3. ARQUITETURA DE COMPUTADOR. 4. ARQUITETURA DE REDE DE COMPUTADOR. 5. LINGUAGEM DE PROGRAMAÇÃO (COMPUTADORES). 6. SOFTWARE LIVRE. 7. SISTEMAS DE TRANSMISSÃO DE DADOS. 8. SISTEMAS OPERACIONAIS (COMPUTADORES). I. Pereira, Renato. I. Título.

CDU 681.3
CDD 371.334

PeR – BPE 17-66

ISBN: 978-85-64778-61-0

Dedicamos este trabalho a Deus,
às nossas famílias
e ao grupo SanUSB.

SUMÁRIO

ÍNDICE DE FIGURAS E TABELAS

Capítulo 1
Introdução

A união entre os meios de comunicação e os computadores está revolucionando a educação e, cada vez mais, as tecnologias estão permeando as ações pedagógicas que colocam os professores diante do desafio de rever os paradigmas sobre a educação, bem como de perder a insegurança a respeito do contato com o novo. Nesse sentido, a Internet surgiu como parte dessa união e, segundo Alava (2002), possibilita atividades pedagógicas inovadoras, como pesquisas em acervos bibliográficos on-line e acesso a grupos de discussão e pesquisa, o que gera novos conceitos e novos modos de aprendizagem.

Com a introdução do computador como mediador didático, desenvolveram-se softwares específicos, programas e os protocolos de comunicação, para serem utilizados em contextos de ensino-aprendizagem, o que não afasta o fato de que vários softwares desenvolvidos para outras finalidades, também sejam utilizados no processo de ensino-aprendizagem.

As novas tecnologias não dispensam a figura do professor, ao contrário, exigem deste que adicione ao seu perfil novas exigências bem mais complexas tais como: saber lidar com ritmos individuais dos seus alunos, apropriar-se de técnicas novas de elaboração de material didático produzido por meios eletrônicos, trabalhar em ambientes virtuais diferentes daqueles do ensino tradicional da universidade, adquirir uma nova linguagem e saber manejar criativamente a oferta tecnológica. Nesse sentido, Litwin (1997) destaca a importância de entendermos as novas tecnologias digitais como sendo um produto sócio-cultural, ferramentas físicas e simbólicas que servem de mediadores na interação do homem com o meio, no sentido de compreendê-lo e transformá-lo.

Segundo Freire e Nogueira (1999), a educação visa não apenas inserir o homem no mundo, mas com o mundo, de uma forma crítica e autônoma. Conseqüentemente, deve-se rever não só os valores e métodos do modelo tradicional de ensino-aprendizagem, como também, avaliar como os softwares educacionais são utilizados, atualmente, no ensino.

1.1 OS SOFTWARES EDUCACIONAIS

Entre as características principais de um software educativo está o seu caráter didático, que possibilita a construção do conhecimento em uma determinada área com ou sem a mediação de um professor. Jucá (2006) defende que a qualidade de um software educativo está relacionado ao poder de interação entre aluno e programa mediado pelo professor e à facilidade de atualização dos conteúdos.

Sancho (1998) também conceitua um software educativo como um programa que possui recursos que foram projetados com a intenção e finalidade de serem usados em contextos de ensino-aprendizagem, sendo projetado para tal. Estes programas se aplicam a diferentes

finalidades que vão desde a aquisição de conceitos, passando pelo desenvolvimento de habilidades, até a resolução de problemas. Nesta categoria entram aqueles que não são desenvolvidos com finalidades educativas, mas são utilizados para esse fim. São os programas de uso geral utilizados em contexto de ensino e desenvolvimento cognitivo, como por exemplo, Banco de Dados, Compiladores de Programas, Processadores de Texto, Planilhas Eletrônicas e Editores Gráficos.

Tajra (2000) defende que os softwares educacionais aplicativos podem ser utilizados com finalidade tecnológica ou com finalidade educativa. Nos softwares aplicativos com finalidade tecnológica a importância é dada aos conceitos relacionados à aplicação prática, e utilizado principalmente em cursos de formação profissionalizante, que é o caso do *software* abordado nesse trabalho. Já os softwares aplicativos com finalidade educativa são os mesmos softwares da classificação anterior, porém são utilizados para demonstrações no contexto educacional, como por exemplo, a programação de planilhas eletrônicas para simulação de equações de um sistema real abordado no contexto educacional.

O conhecimento dos princípios básicos de ferramentas computacionais torna-se indispensável à formação da cidadania contemporânea. Por isso, é necessário que o ensino possa fornecer um conjunto de competências específicas que permitam perceber e interagir com a evolução tecnológica presente no cotidiano.

O termo "aprendizagem" é usado freqüentemente na linguagem diária e, de um modo geral, é tido como sinônimo de "conhecimento". Neste estudo, no entanto, é necessário diferenciar o processo do produto, ou seja, é através do processo de aprendizagem que adquirimos conhecimento, ou seja, o processo de aprendizagem ocorre internamente ao indivíduo. Paralelamente, o resultado deste processo (o conhecimento ou habilidade adquiridos) é externo e, por isso, pode ser mensurado.

Um microcontrolador é um sistema computacional completo, no qual estão incluídos internamente uma CPU (*Central Processor Unit***), memórias RAM (dados),** ***flash*** **(programa) e E²PROM, pinos de I/O (***Input/Output***), além de outros periféricos internos, tais como, osciladores, canal USB, interface serial assíncrona USART, módulos de temporização e conversores A/D, entre outros, integrados em um mesmo componente (chip).**

O microcontrolador PIC® (*Periferal Interface Controler*), da Microchip Technology Inc. (empresa de grande porte, em Arizona, nos Estados Unidos da América), possui uma boa diversidade de recursos, capacidades de processamento, custo e flexibilidade de aplicações.

1.2 ASSEMBLY X LINGUAGEM C

A principal diferença entre uma linguagem montada (como assembly) e a linguagem de programação C está na forma como o programa objeto (HEX) é gerado. Em assembly, o processo usado é a montagem, portanto devemos utilizar um MONTADOR (assembler), enquanto que em linguagem C o programa é compilado. A compilação é um processo mais complexo do que a montagem. Na montagem, uma linha de instrução é traduzida para uma instrução em código de máquina. Já em uma linguagem de programação, não existem linhas de instrução, e sim estruturas de linguagem e expressões. Uma estrutura pode ser condicional, incondicional, de repetição, etc...

As expressões podem envolver operandos e operadores mais complexos. Neste caso, geralmente, a locação dos registros de dados da RAM é feita pelo próprio compilador. Por isso, existe a preocupação, por paret do compilador, de demonstrar, após a compilação, o percentual de memória RAM ocupado, pois neste caso é relevante, tendo em vista que cada variável pode ocupar até 8 bytes (tipo *double*).

Para edição e montagem (geração do código HEX) de um programa em assembly, os softwares mais utilizados são o MPASMWIN (mais simples) e o IDE (ambiente integrado de desenvolvimento) multiplataforma MPLABX. Para edição e compilação em linguagem C (geração do código HEX), o compilador utilizado é a versão livre e multplataforma do compilador C18.

Os microcontroladores PIC possuem apenas 35 instruções para programação em assembly nas família de 12 bits (PIC12) e 14 bits (PIC16), e 77 instruções para a família de 16 bits (PIC18) comentada neste livro.

A família PIC16F (14 bits com aproximadamente 35 instruções) não possui uma instrução em assembly que realize multiplicação ou divisão de dois operandos, o que curiosamente é presente na linguagem assembly da família MCS51 (256 instruções que satisfazem a maioria das aplicações práticas). Portanto, para realizar uma multiplicação, é necessário realizar somas sucessivas, ou seja, em vez de multiplicar uma variável por outra, realizar somas de uma variável em uma terceira área de memória, tantas vezes quando for o valor da segunda variável. (X * 5 = X + X + X + X + X).

Mas em linguagem C é possível se utilizar o operador de multiplicação (*), de forma simples e prática. Ao compilar, a linguagem gerada irá converter a multiplicação em somas sucessivas sem que o programador se preocupe com isso.

1.3 VANTAGENS X DESVANTAGENS DA LINGUAGEM C PARA MICROCONTROLADORES

- O compilador C irá realizar o processo de tradução, permitindo uma programação mais amigável e mais fácil para desenvolvimento de aplicações mais complexas como, por exemplo, uso do canal USB e aplicações com o protocolo I^2C.
- A linguagem C permite maior *portabilidade*, uma vez que um mesmo programa pode ser recompilado para um microcontrolador diferente, com o mínimo de alterações, ao contrário do ASSEMBLY, onde as instruções mudam muito entre os diversos modelos de microcontroladores existentes como PIC e 8051.
- Em C para microcontroladores PIC, não é necessário se preocupar com a mudança de banco para acessar os registros especiais da RAM como, por exemplo, as portas de I/O e os registros TRIS de comando de I/O dos pinos, isto é executado pelo próprio compilador através das bibliotecas.
- É possível incluir, de forma simples e padronizada, outro arquivo em C (biblioteca) para servir como parte do seu programa atual como, por exemplo, incluir o arquivo LCD (#include <lcd.c>), desenvolvido para lidar com escrita no display LCD.
- O ponto fraco da compilação em C é que o código gerado, muitas vezes, é maior do que um código gerado por um montador (assembler), ocupando uma memória maior

de programa e também uma memória maior de dados. No entanto, para a maioria das aplicações sugeridas na área de automação residencial e industrial, a linguagem C para PIC e outros microcontroladores se mostra a mais adequada, tendo em vista que a memória de programa tem espaço suficiente para estas aplicações.

- Outra desvantagem é que o programador não é "forçado" a conhecer as características internas do hardware, já que o mesmo se acostuma a trabalhar em alto nivel, o que compromete a eficiência do programa e também o uso da capacidade de todos os periféricos internos do microcontrolador. Isso provoca, em alguns casos, o aumento do custo do sistema embarcado projetado com a aquisição de novos periféricos externos.

1.4 ARQUITETURAS DOS MICROCONTROLADORES

A arquitetura de um sistema digital define quem são e como as partes que compõe o sistema estão interligadas. As duas arquiteturas mais comuns para sistemas computacionais digitais são as seguintes:

- **Arquitetura de Von Neuman:** A Unidade Central de Processamento é interligada à memória por um único barramento (bus). O sistema é composto por uma única memória onde são armazenados dados e instruções;
- **Arquitetura de Harvard:** A Unidade Central de Processamento é interligada a memória de dados e a memória de programa por barramentos diferentes, de dados e de endereço. O PIC possui arquitetura Harvard com tecnologia RISC, que significa *Reduced Instruction Set Computer* (Computador com Conjunto de Instruções Reduzido). O barramento de dados é de 8 bits e o de endereço pode variar de 13 a 21 bits dependendo do modelo. Este tipo de arquitetura permite que, enquanto uma instrução é executada, uma outra seja "buscada" na memória, ou seja, um *pipeline*(sobreposição), o que torna o processamento mais rápido.

1.5 O CONTADOR DE PROGRAMA (PC)

O contador de programa é responsável de indicar o endereço da memória de programa para que seu conteúdo seja transportado para a CPU para ser executado. Na família PIC16F ele contém normalmente 13 bits, por isso, pode endereçar os 8kwords de 14 bits (o PIC16F877A possui exatamente 8k words de 14 bits, ou seja, 14 kbytes de memória de programa). A família 18F ele possui normalmente 21 bits e é capaz e endereçar até 2 Megas words de 16 bits (o PIC18F2550 possui 16k words de 16 bits, ou seja, 32 kbytes de memória de programa). Cada Word de 14 ou 16 bits pode conter um código de operação (opcode) com a instrução e um byte de dado.

1.6 BARRAMENTOS

Um barramento é um conjunto de fios que transportam informações com um propósito comum. A CPU pode acessar três barramentos: o de endereço, o de dados e o de controle. Como foi visto, cada instrução possui duas fases distintas: o ciclo de busca, quando a CPU coloca o conteúdo do PC no barramento de endereço e o conteúdo da posição de memória é colocado no Registro de instrução da CPU, e o ciclo de execução, quando a CPU executa

o conteúdo colocado no registro de instrução e coloca-o na memória de dados pelo barramento de dados. Isso significa que quando a operação do microcontrolador é iniciada ou resetada, o PC é carregado com o endereço 0000h da memória de programa.

As instruções de um programa são gravadas em linguagem de máquina hexadecimal na memória de programa *flash* (ROM). No início da operação do microcontrolador, o contador de programa (PC) indica o endereço da primeira instrução da memória de programa, esta instrução é carregada, através do barramento de dados, no Registro de Instrução da CPU.

Um opcode (código de instrução), gerado na compilação em hexadecimal, contém uma instrução e um operando. No processamento, a CPU compara o código da instrução alocada no registro de instrução com o Set de Instruções do modelo fabricado e executa a função correspondente. Após o processamento, o operando dessa instrução indica para a CPU qual a posição da memória de dados que deve ser acessada e, através do barramento de controle, a CPU comanda a leitura ou a escrita nesta posição.

Após o processamento de uma instrução, o PC é incrementado para indicar o endereço do próximo código de instrução (opcode), da memória de programa, que deve ser carregado no registro de instrução.

1.7 A PILHA (STACK)

A pilha é um local da RAM (no PIC18F2550 é localizada no final dos Registros de Função Especial entre FFDh e FFFh) onde é guardado o endereço da memória de programa antes de ser executado um pulo ou uma chamada de função localizada em outra posição de memória.

1.8 CICLO DE MÁQUINA

O oscilador externo (geralmente um cristal) ou o interno (circuito RC) é usado para fornecer um sinal de clock ao microcontrolador. O clock é necessário para que o microcontrolador possa executar as instruções de um programa.

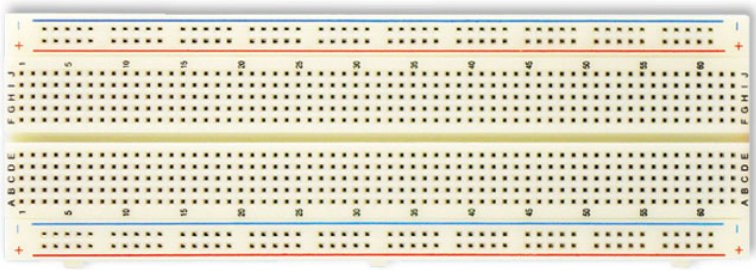
Nos microcontroladores PIC, um ciclo de máquina (CM) possui quatro fases de *clock.* Dessa forma, para uma frequência de*clock* externo (f) de 4MHz, tem-se um ciclo de máquina CM igual a 4 x 1/f, que corresponde a 1μs, ou seja, 1 MIPS (milhões de instruções por segundo).

O Contador de Programa (PC) é incrementado automaticamente na primeira fase do ciclo de máquina e a instrução seguinte é resgatada da memória de programa e armazenada e executada no registro de instruções da CPU no quarto ciclo.

1.9 MATRIZ DE CONTATOS OU PROTOBOARD

Para desenvolver os projetos e exercícos propostos nessa apostila será necessário a uilização de uma Matriz de Contatos (ou *Protoboard* em inglês), mostrada na Figura 1.1, que é uma placa com diversos furos e conexões condutoras para montagem de circuitos eletrônicos. A grande vantagem do *Protoboard*na montagem de circuitos eletrônicos é a facilidade de inserção de componentes (não necessita soldagem).

Figura 1. 1: Protoboard.



Fonte: Elaborado pelo autor.

Na superfície de uma matriz de contatos há uma base de plástico em que existem centenas de orifícios onde são encaixados os componentes ou também por ligações mediante fios. Em sua parte inferior são instalados contatos metálicos que interliga eletricamente os componentes inseridos na placa que são organizados em colunas e canais. De cada lado da placa, ao longo de seu comprimento, há duas colunas completas. Há um espaço livre no meio da placa e de cada lado desse espaço há vários grupos de canais horizontais (pequenas fileiras), cada um com 05 orifícios em curto-circuito.

Em alguns pontos do circuito é necessário limitar a intensidade da corrente elétrica. Para fazer isso utilizamos um componente chamado resistor. Quanto maior a resistência, menor é a corrente elétrica que passa num condutor.

1.10 FONTES DE ALIMENTAÇÃO

As fontes mais comuns em sistemas embarcados com microcontroladores são baterias recarregáveis ou conversores CA-CC como carregadores de baterias de dispositivos computacionais portáteis.As baterias ou pilhas são dispositivos que armazenam energia química e a torna disponivel na forma de energia elétrica.

A capacidade de armazenamento de energia de uma bateria é medida através da multiplicação da corrente de descarga pelo tempo de autonomia, sendo dado em ampére-hora (1 Ah= 3600 Coulombs). Deve-se observar que, ao contrário das baterias primárias (não recarregáveis), as baterias recarregáveis não podem ser descarregadas até 0V pois isto leva ao final prematuro da vida da bateria. Na verdade elas têm um limite até onde podem ser descarregadas, chamado de tensão de corte. Descarregar a bateria abaixo deste limite reduz a vida útil da bateria.

As baterias ditas 12V, por exemplo, devem operar de 13,8V (tensão a plena carga), até 10,5V (tensão de corte), quando 100% de sua capacidade terá sido utilizada, e é este o tempo que deve ser medido como autonomia da bateria.

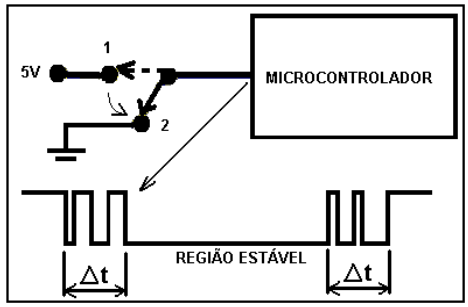
Como o comportamento das baterias não é linear, isto é, quando maior a corrente de descarga menor será a autonomia e a capacidade, não é correto falar em uma bateria de 100Ah. Deve-se falar, por exemplo, em uma bateria 100A.h padrão de descarga 20 horas, com tensão de corte 10,5V. Esta bateria permitirá descarga de 100 / 20 = 5A durante 20 horas, quando a bateria irá atingir 10,5V.

Outro fator importante é a temperatura de operação da bateria, pois sua capacidade e vida útil dependem dela. Usualmente as informações são fornecidas supondo T=25°C ou T=20°C, que é a temperatura ideal para maximizar a vida útil.

1.11 RUÍDO (*BOUNCING)* E FILTRO (*DEBOUNCING*)

Em operações de Liga/Desliga e mudança de nivel lógico (ver Figura 1.2), surge um ruído (*Bouncing*) na transição que, caso uma interrupção esteja habilitada ou até mesmo um contador de evento, pode provocar várias interrupções ou contagens. As formas mais comuns de filtro (*Debouncing*) são via software, programando um tempo (em torno de 100ms, dependendo da chave) após as transições, de modo a eliminar o ruído antes de efetuar uma instrução, ou via hardware, utilizando um capacitor de filtro em paralelo com a chave.

Figura 1. 2: Ruído.



Fonte: Elaborado pelo autor.

1.12 PROTOCOLO DE COMUNICAÇÃO USB

A USB, do inglês *Universal Serial Bus*, é o padrão de interface para periféricos externos ao computador provavelmente mais popular dos já criados. Um sistema USB é composto por hardware mestre e escravo. O mestre é chamado de *host* e o escravo denomina-se dispositivo ou simplesmente periférico. Todas as transferências USB são administradas e iniciadas pelo *host*. Mesmo que um dispositivo queira enviar dados, é necessário que o host envie comandos específicos para recebê-los.

A fase de preparação, conhecida como enumeração, acontece logo depois de quando o dispositivo USB é fisicamente conectado ao computador. Nesse momento, o sistema operacional realiza vários pedidos ao dispositivo para que as características de funcionamento sejam reconhecidas. O sistema operacional, com a obtida noção do periférico USB, atribui-lhe um endereço e seleciona a configuração mais apropriada de acordo com certos critérios. Com mensagens de confirmação do dispositivo indicando que essas duas últimas operações foram corretamente aceitas, a enumeração é finalizada e o sistema fica pronto para o uso.

1.13 MÉTODOS DE COMUNICAÇÃO USB

Os métodos mais comuns de comunicação USB, também utilizados pela ferramenta SanUSB, são:

- ***Human Interface Device*** **(HID)** - O dispositivo USB é reconhecido automaticamente pelo sistema operacional Windows®, Linux ou Mac OSX como um Dispositivo de Interface Humana (HID), não sendo necessário a instalação de driver especiais para a aplicação. Este método apresenta velocidade de comunicação de até 64 kB/s e é utilizado pelo gerenciador de gravação da ferramenta SanUSB no Linux.

- ***Communication Device Class*****(CDC)** – Basicamente o driver emula uma porta COM, fazendo com que a comunicação entre o software e o firmware seja realizada como se fosse uma porta de comunicação serial padrão. É o método mais simples de comunicação bidirecional com velocidade de comunicação é de até 115 kbps, ou seja, aproximadamente 14,4 kB/s.

- ***Mass Storage Device*** (MSD) - Método customizado para dispositivos de armazenamento em massa que permite alta velocidade de comunicação USB, limitado apenas pela própria velocidade do barramento USB 2.0 (480 Mbps). Este método é utilizado por pendrives, scanners, câmeras digitais. Foi utilizado juntamente com a ferramenta SanUSB para comunicação com software de supervisão programado em Java.

Como foi visto, a comunicação USB é baseada em uma central (*host*), onde o computador enumera os dispositivos USB conectados a ele. Para entender mas a fondo el protocolo USB recomiendo los libros de Jan Axelson que son excelentes en la descripcion de todos los elementos que se deben considerar al diseñar un sistema que maneje el USB.Existen varios tipos de transferencias usadas con el USB siendo 3 las mas conocidas: Bulk, Interrupción y Asíncrona. En el mundo de los PICs las mas usadas son la BULK y la de Interrupción, mejor conocidas como CDC o HID respectivamente, básicamente se diferencian en la forma en que transmiten los datos hacia el host y en la forma en que los descriptores están construidos.Existem três grandes classes de dispositivos comumente associados a USB: dispositivos de interface humana (HID), classe de dispositivos de comunicação (CDC) e dispositivos de armazenamento em massa (MSD). Cada uma dessas classes já possui um driver implementado na maioria dos sistemas operacionais. Portanto, se adequarmos o firmware de nosso dispositivo para ser compatível com uma dessas classes, não haverá necessidade de implementar um driver.

Nos sitemas operacionais Windows®, Linux e Mac OSX, o modo mais fácil de comunicar com o PIC USB é o CDC, por uma razão simples, o modo de comunicação convencional utilizado para dispositivos computacionaisé baseados na comunicação serial. O método CDC no Linux/Mac OSX e o HID no Windows® são nativos, ou seja, não é necessário instalar nenhum driver no sistema operacional para que o PC reconheça o dispositivo USB. El programa para la PC puede ir desde Visual Basic, DELPHI, C++ o C, incluso existen em sin fin de ejemplos em la WEB para realizar em comunicación a traves del puerto serie. En el caso del modo CDC LINUX facilita las cosas ampliamente, basta em tener instalado LIBUSB y CDC_ACM dentro del KERNEL, esto viene por default em la mayoria de las distros de LINUX.

Capítulo 2

Utilizando o compilador C18 e a IDE MPLABX multiplataforma com funções em português

O compilador C18 é projetado para gerar arquivos compilados .hex para microcontroladores com suporte ao padrão ANSI C embutido na IDE do MPLABX. O MPLABX é um ambiente integrado de desenvolvimento para microcontroladores PIC em geral. Porém, nativamente ele suporta apenas,de forma padrão, a linguagem assembly. Por isso, existe a necessidade de instalar um compilador em linguagem C. No livro proposto foram desenvolvidos projetos práticos utilizando a versão livre e multiplataforma do compilador C18. Este apresenta os seguintes atributos:

- Compatibilidade com o padrão ANSI;
- Integração com a IDE livre e multiplataforma MPLABX para gerenciamento de projeto e depuração (*debug*);
- Compatibilidade com módulos de objetos gerados pelo montador MPASM, permitindo a liberdade de programar um firmware contendo instruções em linguagem C e assembly no mesmo código.

Em adição aos qualificadores de acesso do padrão ANSI C (const, volatile), o compilador C18 introduz qualificadores de acesso como *const rom* e *ram*para variáveis. Sintaticamente, a forma de uso desses novos qualificadores é parecida com o uso de const e volatile. Estes qualificadores definem o local onde as variáveis e *strings*serão armazenadas, ou seja, na memória de programa (const rom) ou na memória de dados (*ram*). Os qualificadores da declaração de variáveis, caso não seja especificado, é por padrão na memória de dados (*ram*). Como a memória de dados é menor, com somente 2 Kbytes, é aconselhável utilizar o qualificador *const rom* para variáveis com grandes *strings*no intuito de armazenamento da variável na memória de programa (*flash*).

Pro outro lado, o compilador utiliza também a declaração #pragma para alocar uma instrução na memória de programa (*flash*) ou na memória da dados (RAM), entre elas #pragma code (instruções executáveis na *flash*), #pragma romdata (variáveis e constantes na *flash*), #pragma udata (variáveis não inicializadas na RAM) e #pragma idata (variáveis inicializadas na RAM). Os pragmas são geralmente utilizados para definir o endereço fixo da memória de programa (*flash*)nas interrupções do microcontrolador(#pragma code _HIGH_INTERRUPT_VECTOR = 0x1008).

Para utilizar esta ferramenta multiplataforma, é necessário instalar 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 arquivos do grupo SanUSB. Importante enfatizar que estas versões são livres e foram testadas com sucesso.

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

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

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

e em seguida:

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

Deve-se realizar o mesmo método para a instalação o compilador C18.

Para compilar os programas com o C18 e o SanUSB, basta abrir o Projeto1C18.X. Todos esses programas estão disponíveis no Grupo 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).

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 sanusb–wEndereço 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, é necessário copiar todos os arquivos da pasta Projeto1C18.X e colar dentro do novo Projeto2.X. Em seguida, deve-se abrir o Projeto2.X e compilar o arquivo em *Source files*.

2.1 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.

2.2 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();
```

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 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.

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.

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.

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 nivel lógico inverso em uma saída digital.
Argumentos de entrada: Nome da saída digital que irá ser invertida. Este nome é construído pelo inicio 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 nivel 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.

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()
{
//espera a interrupção externa 1 (em B1)
if (ext1_interrompeu){
//limpa a flag de interrupção
ext1_interrompeu = 0;
//inverte o LED em B0
PORTBbits.RB0 =! PORTBbits.RB0;
}
//espera o estouro do timer0
if (timer0_interrompeu)          {
//limpa a flag de interrupção
timer0_interrompeu = 0;
//inverte o LED em B7
PORTBbits.RB0 =! PORTBbits.RB0;
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.

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.
```

timer0_ms(tempo)
Função: Tempo de atarso em ms gerado pelo timer 0 configrado e 8 bits.
Argumentos de entrada: Tempo.
Argumentos de saída: Não há.

Exemplo:

```
while (1){
inverte_saida(pin_b7);
timer0_ms(500);
}
```

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”
#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);
```

Capítulo 3
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.
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.
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);
}
```

3.1 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 de 2400 bps
        taxa_rs232(2400);
        //programa normal parado aqui
                while(1);
    }
```

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;
            }
        }
    }
```

sendrw(rom char ***);**

Função: Transmite pela serial *strings* ou caracteres armazenados no ROM (*memória flash*). Argumentos de entrada: O dado a ser transmitido deve ser de 8 bits do tipo char.
Argumentos de saída: Não há.

```
Exemplo: const rom char nome[] =”Serial “;
        sendsw((rom char *)nome); // escreve Serial
        tempo_ms(300);
sendsw((rom char *)”Outra forma de enviar\r\n”);// 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.hex, tente novamente compilar e gravar o firmware desejado. ***Evitar o uso da função printf().***

sendsw(char ***);**
Função: Transmite caracteres pela serial UART alocados na RAM.
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 *)”Outra forma de enviar\r\n”);// 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.hex, 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().***

Capítulo 4

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:

- Um equipamento específico para gravação de um programa no microcontrolador;
- 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;
- 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.*
- Conversor analógico-digital (AD) externo, tendo em vista que ele dispõe internamente de 10 ADs de 10 bits;
- *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. O diagrama ilustrativo pode ser visto na Figura 4.1.

Figura 4. 1: Gravação do PIC via PC.

Fonte: Elaborado pelo autor.

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, campeões na V Feira Estadual de Ciências e Cultura do Ceará na categoria robótica educacional em 2011 e 3º lugar no Congresso tecnológico Infobrasil 2014.

4.1 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, onde também é possível baixar periodicamente as atualizações dessa ferramenta e a inclusão de novos programas.

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®.

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 MPLABX C18, o Hi-Tech e o CCS. Para compilar com o MPLAX + C18 e a placa SanUSB em Linux, Windows ou Mac OSX é simples. Inicialmente, basta instalar normalmente o MPLABX e o C18 para o sistema operacional desejado. Depois de instalado basta abrir o MPLAX e clicar em Open project e escolher um projeto descompactado.X. Este projeto já pisca um led no pino B7 e faz a leitura do AD no pino A0 e envia pela serial.

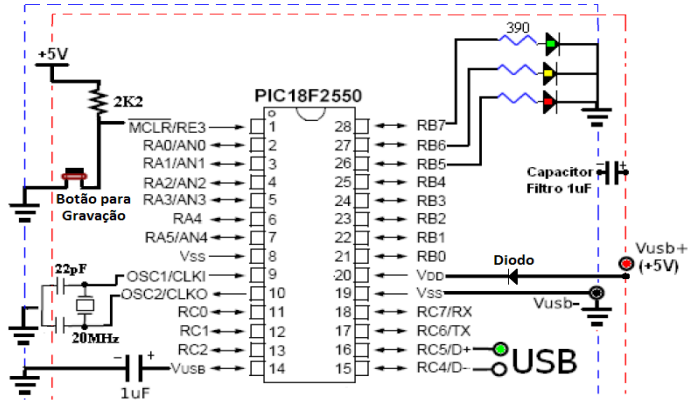
Para modificar o programa exemplo, altere o PWM_AD_serial.c e clique em *Clean und Build Project* (ícone que tem um martelo e uma vassoura,)

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 mostradana Figura 4.2.

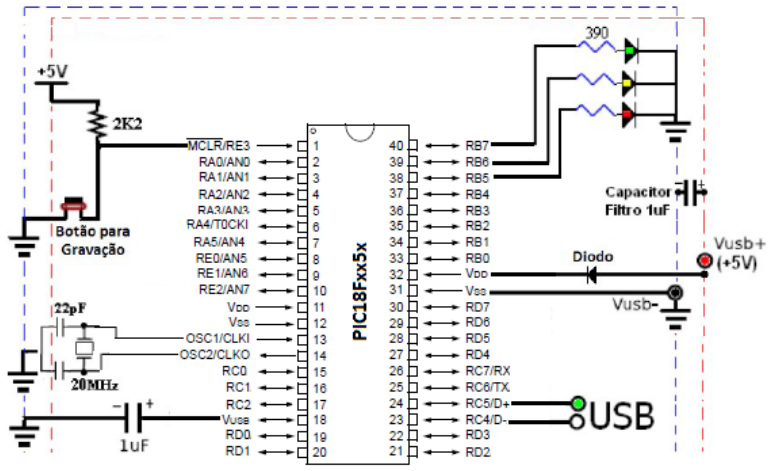
Figura 4. 2: Esquemático de montagem da Ferramenta para 28 pinos.



Fonte: Elaborado pelo autor.

Para um microcontrolador de 40 pinos, o circuito é mostrado na Figura 4.3.

Figura 4. 3: Esquemático de montagem da ferramenta para 40 pinos.



Fonte: Elaborado pelo autor.

Os componentes básicos do circuito são:

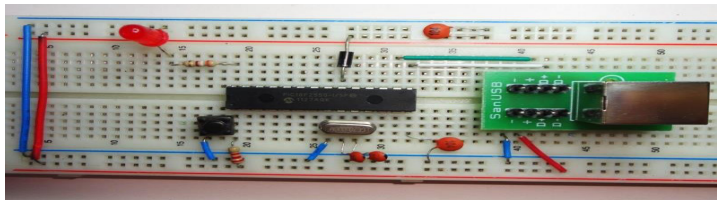
- 1 microcontrolador da família PIC USB (18F2550, 18F2455, 18F4550, etc.);
- 1 cristal de 20MHz;
- 2 capacitores de 22pF;
- 2 capacitores de 1uF (um no pino 14 Vusb e outro entre o +5V e o Gnd) ;
- 3 leds e 3 resistores de 390 (só é necessário um led com resistor no pino B7);
- 1 resistor de 2k2 e um 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 4.4 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 4. 4: Esquema montado em protoboard e conector USB.

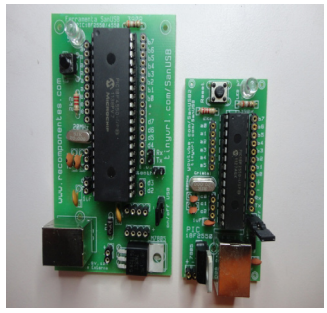


Fonte: Elaborado pelo autor.

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 4.5 mostra o PCB, disponivel nos Arquivos do Grupo SanUSB, e o circuito para esta ferramenta com entrada para fonte de alimentação externa. Para desenvolver a ferramenta SanUSB, como as placaa da Figra 4.5, deve-se entrar em contato com o grupo SanUSB.

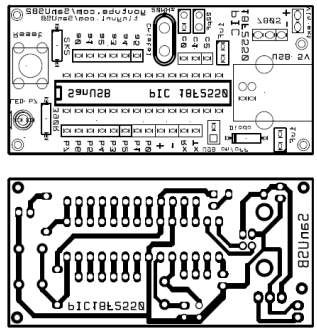
Figura 4. 5: Placa SanUSB montada em PCB.



Fonte: Elaborado pelo autor.

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 4.6, transferir para a placa de cobre, corroer, furar e soldar os componentes.

Figura 4. 6: PCB da Ferramenta SanUSB.



Fonte: Elaborado pelo autor.

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 e clicar no item Arquivos.

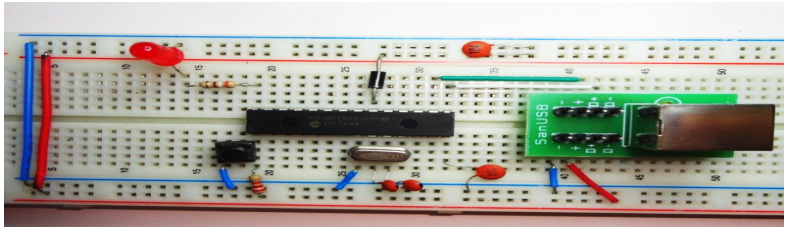
Durante a programação do microcontrolador basta abrir com o MPLABXX o projeto Projeto1.C18.X já configurado. A biblioteca SanUSB1.h contém funções básicas para o

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.

4.2 Prática pisca LED

Após montar o circuito da Ferramenta SanUSB, conforme Figura 4.7, deve-se iniciar a sequência de práticas.

Figura 4. 7: Circuito básico da Ferramenta SanUSB.



Fonte: Elaborado pelo autor.

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 4.8. O catodo do LED deve ser aterrado.

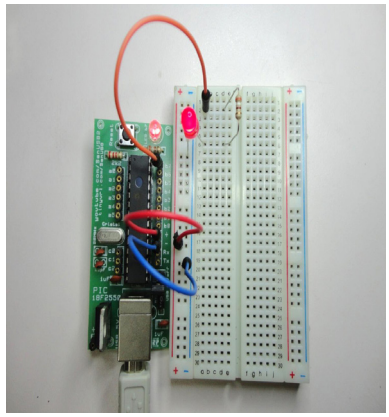
Programação em Linguagem C:

```
#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(){
        //Função necessária para habilitar o dual clock (48MHz para USB e 4MHz para CPU)
        clock_int_4MHz();
        while(1){
                // comuta Led na função principal tempo_ms(500);
                inverte_saida(pin_b7);
        }}
```

Figura 4. 8: Prática Pisca LED, montada em protoboard.



Fonte: Elaborado pelo autor.

Para modificar o tempo de pisca do LED é necessário alterar o valor 500 entre parênteses na função tempo, como por exemplo para: tempo_ms(1000);

Com esta nova programação, o LED piscará a cada 1 segundo, considerando que o comando está em milissegundos.

OBS: para inserir comentários no programa, deve-se inserir antes: //

Pode-se reduzir o programa em linguagem C substituindo as funções:

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

Pelas funções:

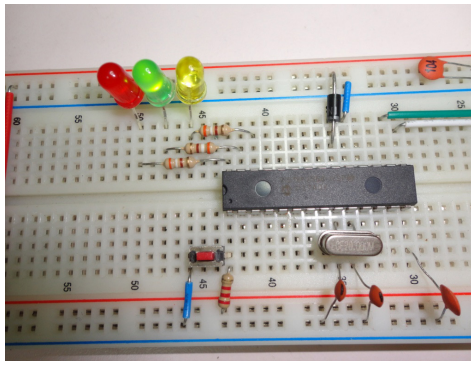
inverte_saida(pin_b7); //alterna pino B7 entre nivel lógico baixo e alto: pisca o LED

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

4.3 Prática pisca 3 LEDS

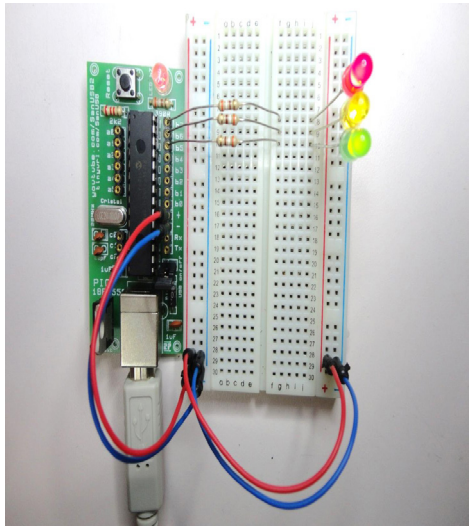
Considerando o aprendizado das funções da prática anterior, insira mais 2 LEDs ao circuito SanUSB (pinos b6 e b5, por exemplo) e programem o PIC para piscar os 3 LEDs em sequência. O esquemático pode ser visto nas Figuras 4.9 e 4.10.

Figura 4. 9: Prática pisca 3 LEDs.



Fonte: Elaborado pelo autor.

Figura 4. 10: Prática Pisca 3 LEDs, montada em protoboard.



Fonte: Elaborado pelo autor.

O código a seguir é uma sugestão:

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

```
        while (1) {
                nivel_alto(pin_b7); //SAIDA ALTA NO PINO B7 - LED ACENDE
                tempo_ms(500); //ATRASO 0,5 SEG
                nivel_baixo(pin_b7); //SAIDA BAIXA NO PINO B7 - LED APAGA
                tempo_ms(500); //ATRASO 0,5 SEG
                nivel_alto(pin_b6); //SAIDA ALTA NO PINO B6 - LED ACENDE
                tempo_ms(500); //ATRASO 0,5 SEG
                nivel_baixo(pin_b6); //SAIDA BAIXA NO PINO B6 - LED APAGA
                tempo_ms(500); //ATRASO 0,5 SEG
                nivel_alto(pin_b5); //SAIDA ALTA NO PINO B5 - LED ACENDE
                tempo_ms(500); //ATRASO 0,5 SEG
                nivel_baixo(pin_b5); //SAIDA BAIXA NO PINO B5 - LED APAGA
                tempo_ms(500); //ATRASO 0,5 SEG
        }
}
```

É possível também alterar a frequência dos LEDs. Como exercício para avaliar o aprendizado, tente elaborar um programa para piscar o LED do pino b7 uma vez a cada 1 segundo, o LED do pino b6 duas vezes a cada 0,5 segundos e o LED do pino b5 três vezes a cada 0,1 segundos, utilizando apenas as funções vistas até o momento.

Após isto deve ser passado aos alunos o conceito de laço de repetição finito, como o exemplo a seguir que usa a função FOR.

```
#include "SanUSB1.h"
//Tem que estar declarado no firmware.c
#pragma interrupt interrupcao
void interrupcao(){        }
void main(){
        clock_int_4MHz();
        while (1){//LAÇO INFINITO
                nivel_alto(pin_b7); //SAIDA ALTA NO PINO B7 - LED ACENDE
                tempo_ms(1000); //ATRASO 0,5 SEG
                nivel_baixo(pin_b7); //SAIDA BAIXA NO PINO B7 - LED APAGA
                tempo_ms(1000); //ATRASO 0,5 SEG
                //LAÇO DE REPETIÇÃO POR 2 VEZES
                for (x=0;x<2;x++) {
                        nivel_alto(pin_b6); //SAIDA ALTA NO PINO B6 - LED ACENDE
                        tempo_ms(500); //ATRASO 0,5 SEG
                        nivel_baixo(pin_b6); //SAIDA BAIXA NO PINO B6 - LED APAGA
                        tempo_ms(500); //ATRASO 0,5 SEG
                }
                //LAÇO DE REPETIÇÃO POR 3 VEZES
                for (x=0;x<3;x++){
                        nivel_alto(pin_b5); //SAIDA ALTA NO PINO B5 - LED ACENDE
                        tempo_ms(100); //ATRASO 0,5 SEG
                        nivel_baixo(pin_b5); //SAIDA BAIXA NO PINO B5 - LED APAGA
                        tempo_ms(100); //ATRASO 0,5 SEG
                }       }       }
```

Também é possível utilizar a função *WHILE*, como no exemplo abaixo:

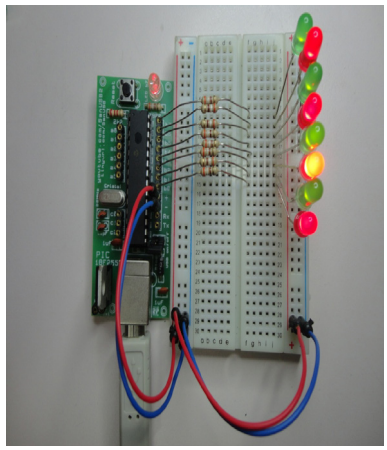
```
#include "SanUSB1.h"

//reseta variáveis para iniciar a contagem
int x=0,y=0;

//Tem que estar declarado no firmware.c
#pragma interrupt interrupcao
```

```
void interrupcao(){ }
void main(){
        clock_int_4MHz();
        // declaração de variáveis auxiliares de nome 'x' e ‘y’ do tipo inteiro
        int x=0,y=0;

        //LAÇO INFINITO
        while (1){
                nivel_alto(pin_b7);//SAIDA ALTA NO PINO B7 - LED ACENDE
                tempo_ms(1000);//ATRASO 0,5 SEG
                nivel_baixo(pin_b7);//SAIDA BAIXA NO PINO B7 - LED APAGA
                tempo_ms(1000);//ATRASO 0,5 SEG

                //enquanto a variável x for menor que 2, repete o ciclo
                while (x<2) {
                        nivel_alto(pin_b6);//SAIDA ALTA NO PINO B6 - LED ACENDE
                        tempo_ms(500);//ATRASO 0,5 SEG
                        nivel_baixo(pin_b6);//SAIDA BAIXA NO PINO B6 - LED APAGA
                        tempo_ms(500);//ATRASO 0,5 SEG
                        x++;//INCREMENTA VARIÁVEL x DE UM EM UM
                }
                //enquanto a variável y for menor que 3, repete o ciclo
                while (y<3){
                        nivel_alto(pin_b5);//SAIDA ALTA NO PINO B5 - LED ACENDE
                        tempo_ms(100);//ATRASO 0,5 SEG
                        nivel_baixo(pin_b5);//SAIDA BAIXA NO PINO B5 - LED APAGA
                        tempo_ms(100);//ATRASO 0,5 SEG
                        y++;//INCREMENTA VARIÁVEL y DE UM EM UM
                }
        }
}
```

Abaixo um exemplo de firmware para rotacionar leds na porta B, de acordo com esquemático da Figura 4.11.

```
//rotacionar leds
#include "SanUSB1.h"
unsigned char d=0b10000000;  // 8 bits
#pragma interrupt interrupcao //Tem que estar  dentro do firmware.c
void interrupcao(){ }
void main(){
        clock_int_4MHz();
        TRISB=0;// porta B como saída
        while(1) {
                if(!entrada_pin_e3){
                        Reset();//pressionar o botão para gravação via USB
                }
                d=d>>1;
                if (d == 0b00000001) {
                        d=0b10000000;//0b -> valor binário
                }
                PORTB=d;
                tempo_ms(100);
        }
}
```

Figura 4. 11: Botões b0b1 com placa SanUSB.



Fonte: Elaborado pelo autor.

4.4 Gravando o microcontrolador via USB no Windows

Para executar a gravação com a ferramenta SanUSB, é importante seguir os seguintes passos:

Baixe o a pasta da ferramenta de desenvolvimento SanUSB, para um diretório raiz C ou D.

Grave no microcontrolador, somente uma vez, com um gravador específico para PIC ou com um circuito simples de gravação ICSP mostrado nas próximas seções, o novo

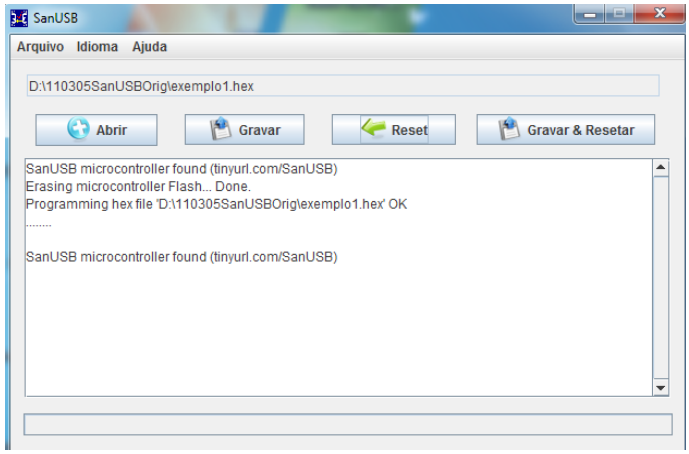
gerenciador de gravação pela USB GerenciadorPlugandPlay.hex disponivel na pasta Gerenciador, compatível com os sistemas operacionais Windows®, Linux e Mac OSX.

Pressione o botão ou conecte o *jump* de gravação do pino 1 no Gnd para a transferência de programa do PC para o microcontrolador.

Conecte o cabo USB, entre o PIC e o PC, e solte o botão ou retire o *jump*. Se o circuito SanUSB estiver correto acenderá o *led* do pino B7.

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® e execute o aplicativo SanUSB da pasta SanUSBwinPlugandPlay. Surgirá a tela da Figura 4.12.

Figura 4. 12: Interface de gravação do microcontrolador via USB.



Fonte: Elaborado pelo autor.

1. Clique em *Abrir* e escolha o programa *.hex* que deseja gravar, como por exemplo, o programa compilado *exemplo1.hex* da pasta ExemploseBibliotecasSanUSB e clique em *Gravar*. Este programa pisca o *led* conectado no pino B7;

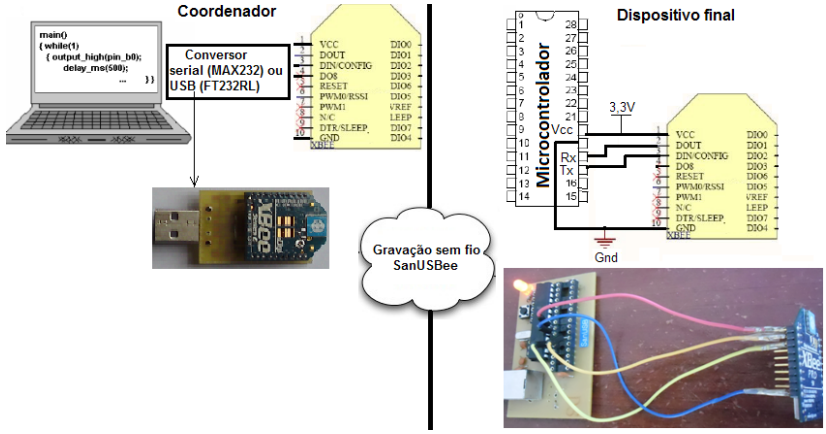
2. Após a gravação do programa, lembre-se de soltar o botão ou retirar o *jump* do pino de gravação e clique em Resetar. Pronto o programa estará em operação. Para programar novamente, repita os passos anteriores a partir do passo 3.

4.5 Gravação wireless de microcontroladores

A gravação *wireless* descrita nesta apostila pode ser feita com modems Zigbee ou Bluetooth. Para a gravação Zigbee são utlizados dois módulos XBee® da Série 1 (S1). De um lado, um módulo é conectado a um PC coordenador conectado ao PC via USB do PC através do chip FTDI FT232RL ou através de uma porta serial real com o MAX-232 e, do outro lado da rede, um módulo Zigbee é conectado ao microcontrolador do dispositivo final. Esta conexão permite a programação sem fio no microcontrolador PIC.

Na Figura 4.13, é possível visualizar uma ilustração para realizar gravação de microcontrolador de forma *wireless* com tensão de alimentação de 3,3V.

Figura 4. 13: Ilustração do circuito de gravação ***wireless*** Zigbee.



Fonte: Elaborado pelo autor.

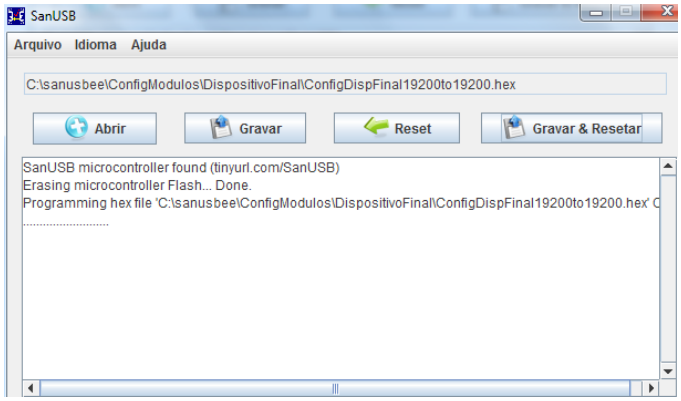
Para mais detalhes basta acompanhar os vídeos da gravação sem fio de microcontroladores e gravação sem fio de microcontroladores via Zigbee disponíveis na internet. Procedimento para gravação *wireless*:

1- **Circuito básico**: Conecte o módulo Xbee® ao microcontrolador da placa SanUSB, com alimentação entre 3V e 3,6V e apenas 4 fios: Vcc (3,3V), Gnd, Tx e Rx, como mostra a figura abaixo. Na figura, o fio vermelho é ligado ao pino 20 (Vcc) do microcontrolador e ao pino 1 (Vcc) do modem Zigbee, o fio azul é ligado ao 19 (Gnd) do microcontrolador e ao pino 10 (Gnd) do modem Zigbee, o fio laranja é ligado ao pino 18 (Rx) do microcontrolador e ao pino 2 (D_{OUT}) do modem Zigbee, e o fio amarelo é ligado ao 17 (Tx) do microcontrolador e ao pino 3 (D_{IN}) do modem Zigbee.

2- **Configuração dos Módulos**: A gravação *wireless* só vai acontecer se os módulos Xbee® da série 1 (coordenador e dispositivo final) estiverem configurados com o mesmo *baud rate* do microcontrolador (19200 bps). Para o coordenador**,** basta conectar, o módulo coordenador ao microcontrolador, ver circuito básico acima, gravar via USB e examinar em qual firmware (*ConfigCoord9600to19200.hex ou ConfigCoord19200to19200.hex)* o led no pino B7 irá piscar intermitentemente. Se o led não piscar, provavelmente existe um erro na ligação do circuito. Após a configuração, coloque o módulo Coordenador no conversor USB-serial e conecte ao PC.

Faça posteriormente o mesmo para o módulo Dispositivo final, gravando o firmware (*ConfigDispFinal9600to19200.hex ou ConfigDispFinal19200to19200.hex)* e deixe-o conectado ao microcontrolador. Quando o led do pino B7 estiver piscando, significa que os módulos estão conectados corretamente e estão aptos para gravação *wireless*. A Figura 4.14 mostra a gravação do código através da interface SanUSB.

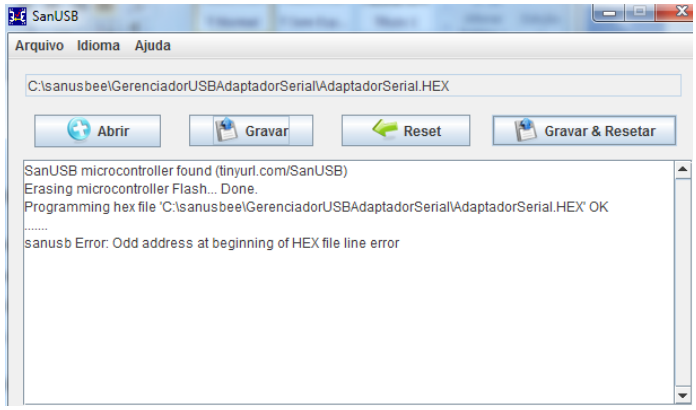
Figura 4. 14: Gravação via USB de Configuração ***wireless***.



Fonte: Elaborado pelo autor.

3- **Adaptador *Wireless***: Agora grave, novamente via USB, o firmware *AdaptadorSerial.hex* da pasta Adaptador*Wireless*. Se, após a gravação do Adaptador, apresentar o erro *Odd address at beginning of HEX file error*, como na figura abaixo, é necessário gravar novamente o gerenciador.hex, com qualquer gravador especifico (ver tutorial), e em seguida, realizar novamente a gravação via U*SB* do firmware aplicativo AdaptadorSerial.hex. Após a transferência deste firmware, o microcontrolador está apto para gravação *wireless*. A Figura 4.15 mostra a agravação do adaptador *wireless*.

Figura 4. 15: Gravação via USB de Adaptador ***wireless***.



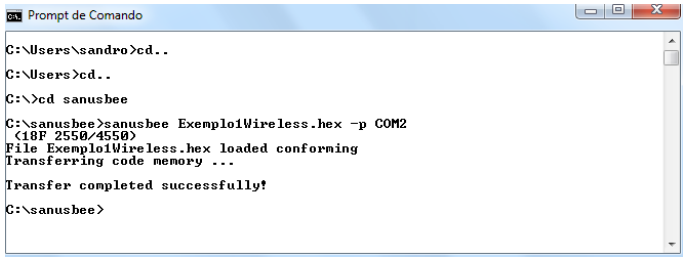
Fonte: Elaborado pelo autor.

Agora basta acessar a pasta sanusbee pelo Prompt do Windows® (Iniciar -> Pesquisar -> Prompt de Comando), como na Figura 4.16, e digitar, como mostrado no vídeo Gravação sem fio de microcontroladores via Zigbee, as linhas de comando, para transferir os progra-

mas aplicativos.hex como o Exemplo1wireless.hex contido na pasta sanusbee.

Exemplo: *sanusbee Exemplo1Wireless.hex –p COM2*

Figura 4. 16: Gravação ***wireless*** zigbee pelo prompt do Windows.



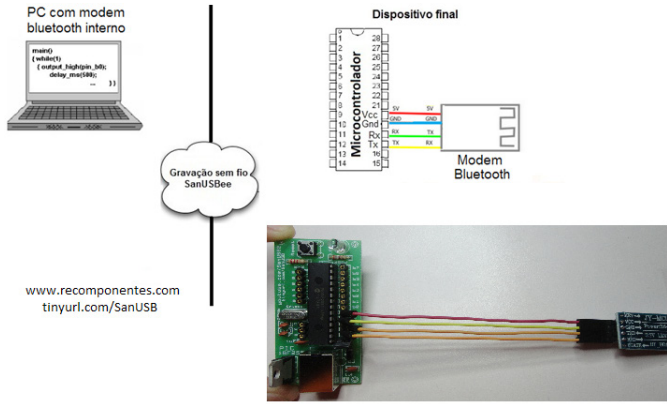
Fonte: Elaborado pelo autor.

A gravação *wireless*via Bluetooth pode ser realizada com apenas um módulo Bluetooth conectado ao microcontrolador, pois normalmente no PC coordenador, como em *laptops* e *desktops*, já existe um módulo bluetooth interno. A tensão do módulo Bluetooth encapsulado, mostrado na figura abaixo, suporta até 6V, diferentemente do módulo Xbee® que suporta de 3,3V. Dessa forma, pode-se conectar o módulo Bluetooth diretamente ao microcontrolador alimentado pela tensão da porta USB de 5V.

De um lado um PC coordenador e, do outro lado da rede, um módulo bluetooth é conectado ao microcontrolador do dispositivo final. Esta conexão permite a programação sem fio no microcontrolador PIC. Pode ser adquirido o modem Bluetooth mostrado neste tutorial.

A Figura 4.17 mostra uma ilustração para realizar gravação de microcontrolador de forma *wireless* Bluetooth com tensão de alimentação de 5V. e a Figura 4.18 mostra o modem Bluetooth conectado à placa SanUSB.

Figura 4. 17: Ilustração do Circuito de gravação ***wireless*** Bluetooth.



Fonte: Elaborado pelo autor.

Deve-se conectar o módulo Bluetooth à placa SanUSB, de acordo com os pinos mostrados na Tabela 4.1:

Tabela 4.1: Conexão dos pinos da placa SanUSB ao módulo Bluetooth.

PLACA SANUSB	MÓDULO BLUETOOTH
VCC (+5V) +	VCC
GND (0V) -	GND
RX	TX
TX	RX

Fonte: Elaborado pelo autor.

Figura 4. 18: Esquema módulo BLUETOOTH com placa SanUSB 4550.

Fonte: Elaborado pelo autor.

Se o seu módulo Bluetooth já foi configurado pelo *Grupo SanUSB*, será necessário apenas plugá-lo à placa e colocar o cabo USB para gravação do firmware. Caso não, realize a configuração mostrada anteriormente. O seguinte código em C aciona um LED/relé conectado ao pino B7 via Bluetooth utilizando as Letras L para ligar, D para desligar e P para piscar. Lembre-se que as letras são maiúsculas, mas você pode alterar na programação.

Mas onde digitar? Caso esteja usando smartphone/tablet Android, deve-se instalar o aplicativo SanUSB, disponível no Grupo SanUSB ou algum outro disponível na *Playstore/ Googleplay* em que seja possível enviar/receber dados de forma serial. Caso esteja utilizando Windows, ou seja, queira usar o Bluetooth do PC, também é possível, para isso instale ou TERMINAL ZUCHI, ou algum outro semelhante.

4.6 Cálculo de taxa de transmissão serial no modo assíncrono

Analisando o *datasheet* do microcontrolador, o valor de n é inserido no registro SPBRG. É possível gerar com 4 MHz na condição (bits BRG16=0 e BRGH=1) tanto 9600 bps como também 19200 bps, pois neste caso de 8 bits (bits BRG16=0 e BRGH=1), o valor

de n obtido na fórmula pode ser colocado somente em um byte, no SPBRG.

MODO 8 BITS

Para 19200: SPBRG = n= (4.000.000 / 19200 / 16) - 1 =>SPBRG = 12;
Para 9600: SPBRG = n= (4.000.000 / 9600 / 16) - 1 =>SPBRG = 25;

Considerando agora uma frequência de clock de 48 MHz na condição (bits BRG16=0 e BRGH=1):

Para 19200:SPBRG = n= (48.000.000 / 19200 / 16) - 1 =>SPBRG = 155;
Para 9600: SPBRG = n= (48.000.000 / 9600 / 16) - 1 = SPBRG = 311;

MODO 16 BITS

Como em 9600bps 48MHz, o SPBRGH é 311, ou seja, maior que 255, não é possível utilizar somente um byte. Por isso é necessário habilitar o byte baixo SPBRG, setando o bit BRG16 em BAUDCON, entrando na condição de 16 bits assíncrono (bits BRG16=1 e BRGH=1). Calculando agora os valores na fórmula de 16 bits, tem-se que:

Para 19200 e 48 MHz: n=(48.000.000 / 19200 / 4) - 1 = 624 = 0x270 ->
SPBRGH = 0x02;
SPBRG=0x70;

Para 9600 e 48 MHz: n=(48.000.000 / 9600 / 4) - 1 = 1249 = 0x4E1->
SPBRGH = 0x04;
SPBRG=0xE1;

Para 19200 e 4 MHz: n=(48.000.000 / 19200 / 4) - 1 = 624 = 0x33 ->
SPBRGH = 0x00;
SPBRG=0x33;

Para 9600 e 4 MHz: n=(48.000.000 / 9600 / 4) - 1 = 1249 = 0x67->
SPBRGH = 0x00;
SPBRG=0x67;

Deste modo, é possível utilizar a seguinte função em C18 para 19200 bps com frequência de 4MHz e de 48MHz no modo de 16 bits:

```
void taxa_serial(unsigned long taxa) { //Modo 16 bits(bits BRG16=1 e BRGH=1)
unsigned long baud_sanusb; //es klappt nut mit long

TRISCbits.TRISC7=1; // RX
TRISCbits.TRISC6=0; // TX
TXSTA = 0x24;       // TX habilitado e BRGH=1
RCSTA = 0x90;       // Porta serial e recepcao habilitada
BAUDCON = 0x08;     // BRG16 = 1

baud_sanusb =REG+ ((48000000/4)/ taxa) - 1;
```

SPBRGH = (unsigned char)(baud_sanusb >> 8);
SPBRG = (unsigned char)baud_sanusb; }
Código em C para MPLAB X:

```
//Utiliza interrupcao serial para receber comandos enviados via bluetooth ou zigbee

#include "SanUSB1.h"
short int pisca=0;
unsigned char comando;
char nome[]

#pragma interrupt interrupcao
void interrupcao()
{
 if (serial_interrompeu)  {
    serial_interrompeu=0;
    comando = le_serial();

        switch (comando){
           case 'L':
           pisca=0; nivel_alto(pin_b7); //Não imprime (printf) dentro da interrupcao
break;

           case 'D':
           pisca=0; nivel_baixo(pin_b7);
           break;

case 'P':
           pisca=1;nivel_alto(pin_b7);
break;
        }
  }
}
void main(){
   clock_int_4MHz();
habilita_interrupcao(recep_serial);
   taxa_serial(19200);

   while(1){
      if (!entrada_pin_e3){Reset();} // pressionar o botão no pino 1 para gravação
  while (pisca==1){
         inverte_saida(pin_b7);tempo_ms (300);
              }//pisca rapido
sendsw("SanUSB\r\n"); //envia de forma sem fio a palavra para o PC ou Android
    tempo_ms (2000);
   }
}
```

É possível realizar recepção serial sem interrupção veirificando o estado da flag serial, que caso seja setada indica a chegada de um byte na USART, como mostrado no firmware a seguir:

```
#include "SanUSB1.h"

#pragma interrupt interrupcao
void interrupcao(){}

void main(){
clock_int_4MHz();
taxa_serial(19200);

while(1)
{
if(!entrada_pin_e3){Reset();}//pressionar o botão para gravação

if(PIR1bits.RCIF)        //Flag que indica byte na USART - Serial_avaliable()
                  {
          PIR1bits.RCIF = 0;   //reset flag
switch(RCREG)// byte recebido
                                  {
                                  case 'L': nivel_alto(pin_b7);
                                   break;//Chega L acende o led

                                   case 'D' : nivel_baixo(pin_b7);
                                   break;//Chega D apaga o led       }}}}
```

Procedimento para gravação *wireless*:

1- Circuito básico: Conecte o módulo bluetooth ao microcontrolador da placa SanUSB com alimentação entre 3V e 6V e apenas 4 fios: Vcc (3,3V), Gnd, Tx e Rx, como mostra a figura acima do circuito. Na figura, o fio vermelho é ligado ao pino 20 (Vcc) do microcontrolador e ao pino Vcc do modem bluetooth, o fio azul é ligado ao 19 (Gnd) do microcontrolador e ao pino Gnd do modem bluetooth, o fio verde é ligado ao pino 18 (Rx) do microcontrolador e ao pino Tx modem bluetooth, e o fio amarelo é ligado ao 17 (Tx) do microcontrolador e ao pino Rx do modem bluetooth.

2- Parear o modem Bluetooth: Após alimentar o modem Bluetooth com 3,3V ou 5V, conectado ao microcontrolador, realizar o pareamento com o PC indo em:

2.1- Iniciar -> Painel de controle -> Adicionar um dispositivo de bluetooth -> linvor -> senha padrão: 1234;

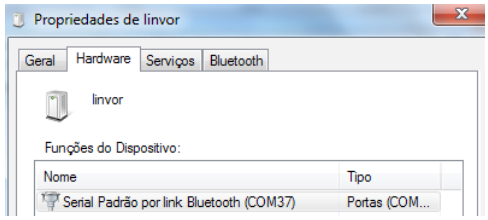
2.2- Após o pareamento, clique em Iniciar -> Painel de controle -> exibir impressoras e dispositivos. Irá aparecer o modem pareado, como, por exemplo, o linvor (ver Figura 4.19).

Figura 4. 19: Pareamento do modem bluetooth.

Fonte: Elaborado pelo autor.

1.3- Clicar em cima, por exemplo, do modem de linvor, e verificar qual porta criada pelo modem Bluetooth, em Hardware, que será utilizada para a gravação *wireless* (ver Figura 4.20).

Figura 4. 20: Verificação da porta serial criada pelo modem bluetooth.



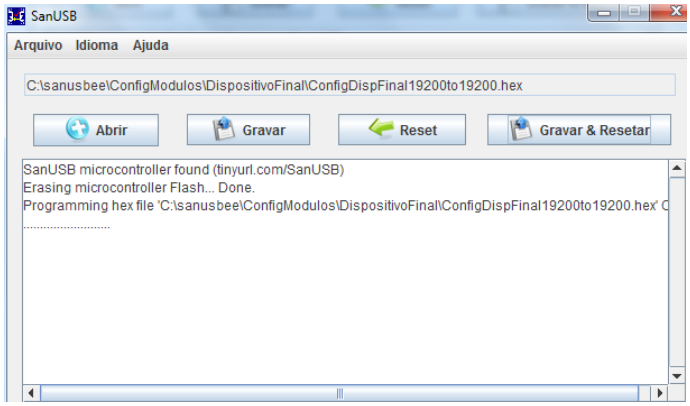
Fonte: Elaborado pelo autor.

O número da porta *Serial Padrão por Link Bluetooth (COM37)* pode ser modificada, por exemplo, para COM9 como neste tutorial, através do Gerenciador de Dispositivos, clicando com o botão direito em cima da porta -> propriedades -> Configuração de Porta -> Avançado -> Número da Porta COM.

3- Configuração do Módulo bluetooth: A gravação *wireless* só vai acontecer se o módulo Bluetooth estiver configurado com o mesmo *baud rate* do microcontrolador (19200 bps). Para isto**,** basta conectar, o módulo bluetooth ao microcontrolador, ver circuito básico acima, gravar via USB o firmware *Configbluetotth9600to19200.hex* (ver Figura 4.21)e verificar seo led no pino B7 irá piscar intermitentemente. Se o led não piscar, provavelmente existe um erro na ligação do circuito.

Quando o led do pino B7 estiver piscando, significa que os módulos estão conectados corretamente e estão aptos para gravação *wireless*.

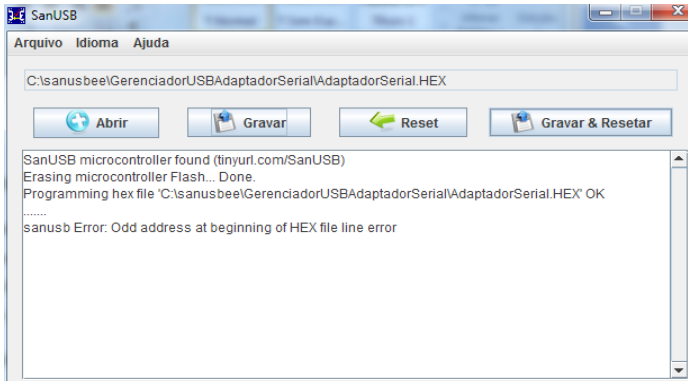
Figura 4. 21: Gravação via USB de Configuração ***wireless***.



Fonte: Elaborado pelo autor.

4- **Adaptador *Wireless***: Agora grave, novamente via USB, o firmware *AdaptadorSerial.hex* da pasta AdaptadorWireless (ver Figura 4.22). Se, após a gravação do Adaptador, apresentar o erro *Odd address at beginning of HEX file error*, é necessário gravar novamente o gerenciador.hex, com qualquer gravador especifico (ver tutorial), e em seguida, realizar novamente a gravação via U*SB* do firmware aplicativo AdaptadorSerial.hex. Após a transferência deste firmware, o microcontrolador está apto para gravação *wireless*.

Figura 4. 22: Gravação via USB de Adaptador ***wireless***.

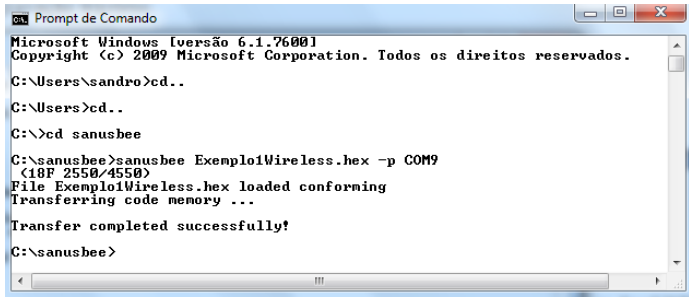


Fonte: Elaborado pelo autor.

Agora basta acessar a pasta sanusbee pelo Prompt do Windows® (Iniciar -> Pesquisar -> Prompt de Comando), como na figura abaixo, e digitar, como mostrado no vídeo *PIC wireless Zigbee programming II*, as linhas de comando, para transferir os programas aplicativos.hex como o Exemplo1wireless.hex contido na pasta sanusbee.

Conforme exemplo mostrado na Figura 4.23: *sanusbee Exemplo1Wireless.hex –p COM9.*

Figura 4. 23: Gravação ***wireless*** bluetooth pelo prompt do Windows.



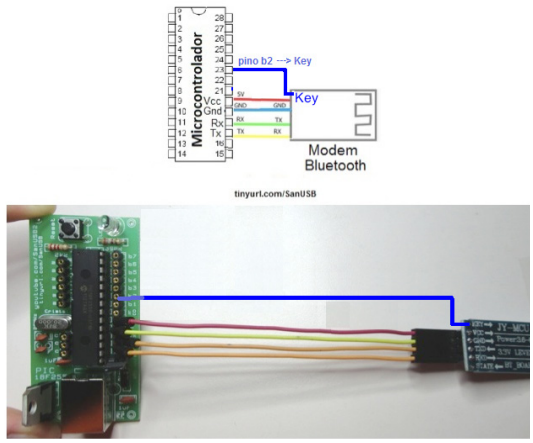
Fonte: Elaborado pelo autor.

As vantagens do modem Bluetooth em relação ao Zigbee, são o preço e a disponibilidade de modems Bluetooth já disponíveis em vários sistemas computacionais como computadores e celulares. A desvantagem em relação ao Zigbee é a distância para gravação de microcontroladores máxima de 10 metros.

4.7 Programa SanUSB para modificar o nome do Módulos Bluetooth via Android

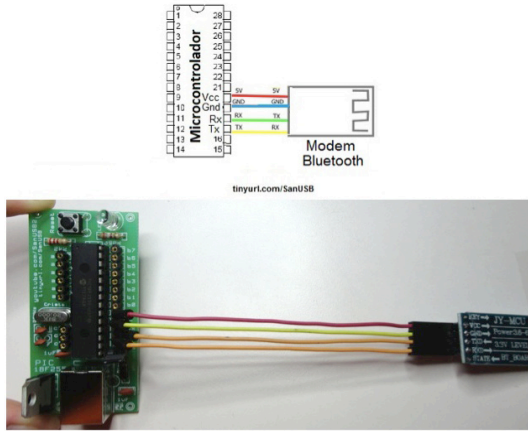
Após gravar o microcontrolador com o firmware dispoível e abrir os aplicativos BTName.apk, para módulos Slave JY-MCU e Linvor v1.04 e v1.06, e o aplicativo BT4 Bluetooth HC05 para módulos Master/Slave Linvor JY-MCU v1.05, basta clicar em "Cadastar BT" e selecionar o modem bluetooth desejado. Após cadastrar o modem bluetooth pelo aplicativo, insira o nome do modem na caixa de texto, com o celular conectado, e apertar o botão **Enviar nome**. As Figuras 4.24 e 4.25 mostram a conexão do modem Bluetooth à placa SanUSB.

Figura 4. 24: Esquema de ligação para módulos master/slave hc-05 e linvor v1.05.



Fonte: Elaborado pelo autor.

Figura 4. 25: Esquema de ligação para módulos slave JY-MCU hc-04, hc-06.

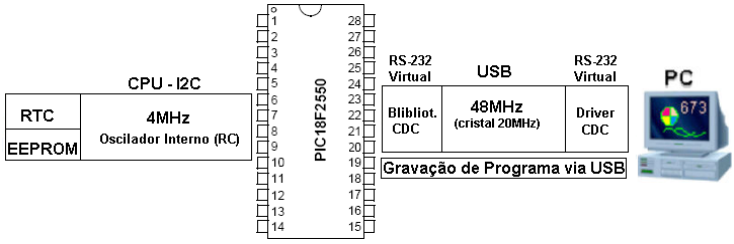


Fonte: Elaborado pelo autor.

4.8 Sistema Dual Clock

Devido à incompatibilidade entre as frequências necessárias para a gravação e emulação serial via USB e a frequência padrão utilizada pela CPU, temporizadores e interface I²C, esta ferramenta adota o princípio *Dual Clock*, ou seja, utiliza duas fontes de *clock*, uma para o canal USB de 48MHz, proveniente do cristal oscilador externo de 20MHz multiplicada por um *prescaler* interno, e outra para o CPU de 4 MHz, proveniente do oscilador RC interno de 4 MHz, como é ilustrado na Figura 4.26.

Figura 4. 26: Comunicação PIC com PC e via I²C.



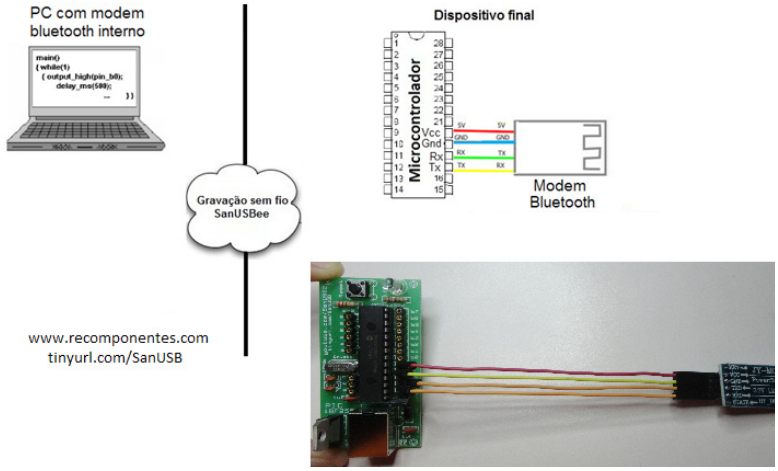
Fonte: Elaborado pelo autor.

Esse princípio de *clock* paralelo, realizado pela instrução clock_int_4MHz(), permite que um dado digitado no teclado do computador, trafegue para o microcontrolador em 48 MHz via USB, depois para periféricos como um relógio RTC ou para a memória EEPROM em 4 MHz via I²C e vice-versa.

4.9 Comunicação serial via Bluetooth ou Zigbee

Neste tópico é mostrado um método de comunicação serial bidirecional através de módulos bluetooth ou zigbee nos sitemas operacionais Windows®, Linux, mac OSX e android. A comunicação é realizada com envio de caracteres ASCII através de qualquer software monitor serial RS-232 como o Bray's Terminal, o Teraterm, Cutecom, etc., e no S.O. android comoo blueterm, S2 bluetooth Terminal, etc.. O Circuito de comunicação *wireless* Bluetooth é o mesmo para a gravação *wireless* e está ilustrado na Figura 4.27.

Figura 4. 27: Ilustração do Circuito de comunicação ***wireless*** Bluetooth.



Fonte: Elaborado pelo autor.

Abaixo segue um programa exemplo que utiliza interrupção serial para receber comandos enviados via bluetooth ou zigbee.

```
#include "SanUSB1.h" // LigaLedBluetooth.c
short int pisca=0;
unsigned char comando;
const char nome [] ="SanUSB ";

#pragma interrupt interrupcao
void interrupcao()
{
 if (serial_interrompeu)  {
    serial_interrompeu=0;
    comando = le_serial();

       switch (comando){
          case 'L':
          pisca=0; nivel_alto(pin_b7); //Não imprime (printf) dentro da interrupcao
```

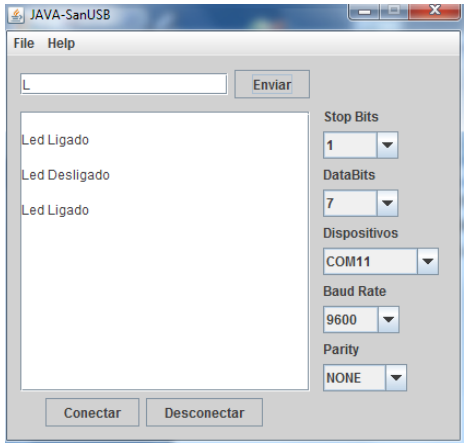
```
break;

            case 'D':
            pisca=0; nivel_baixo(pin_b7);
            break;

case 'P':
            pisca=1;nivel_alto(pin_b7);
break;
        }  }}
void main(){
   clock_int_4MHz();
habilita_interrupcao(recep_serial);
   taxa_serial(19200);
   while(1){
  while (pisca==1){
          inverte_saida(pin_b7);tempo_ms (300);
               }//pisca rapido
sendsw((char *)nome); //envia de forma sem fio a palavra para o PC ou Android
    tempo_ms (2000);    }}
```

O exemplo abaixo mostra a leitura e escrita em um buffer da EEPROM interna do microcontrolador com emulação da serial via bluetooth:

Para utilizar o programa de comunicação Java-SanUSB para emulação serial virtual entre o computador e o microcontrolador, é necessário baixá-lo.

Após executar o programa de comunicação serial Java-SanUSB, verifique a porta COM virtual gerada (COM3,COM4,COM11,etc.) no Windows®, em Painel de Controle\ Todos os Itens do Painel de Controle\Sistema e altere no programa serial Java- SanUSB em Dispositivos e depois clique em Conectar, como mostra a Figura 4.28.

Figura 4. 28: Interface em Java de comunicação serial.



Fonte: Elaborado pelo autor.

4.10 Gravando o microcontrolador via USB no Linux

Esta aplicação substitui a gravação via USB pelo terminal do Linux, pois é uma forma mais simples e direta de gravação. Com apenas dois cliques no instalador automático SanUSB.deb é possível instalar este aplicativo em qualquer máquina com Linux (Ubuntu 10.04, equivalente ou posterior). Depois de instalado, a interface de gravação é localizada em Aplicativos -> acessórios.

Se você já tem o Java instalado (JRE ou SDK) baixe o instalador automático.deb. Se ainda não tem o Java (JRE ou SDK) ou ocorreu algum erro na instalação, baixe o instalador SanUSB, já configurado com o Java JRE.

A Figura 4.29 mostra a interface gráfica desenvolvida para gravação direta de microcontroladores via USB.

Figura 4. 29: Mensagem de programa gravado.

Fonte: Elaborado pelo autor.

Neste aplicativo, estão disponíveis botões para Abrir o programa em hexadecimal compilado, para Gravar o programa hexadecimal no microcontrolador via USB e para Resetar o microcontrolador no intuito de colocá-lo em operação. A interface apresenta ainda um botão para gravar e resetar automaticamente.

É importante salientar que para utilizar esta ferramenta no Linux é necessário estar *logado* com permissão para acessar a porta USB como, por exemplo, super-usuário (*sudo su*), e que para estabelecer comunicação com o microcontrolador é necessário gravar anteriormente no microcontrolador, somente uma vez, com qualquer gravador específico para PIC, o gerenciador de gravação pela USB GerenciadorLinux.hex.

Após gravar o GerenciadorLinux.hex com um gravador convencional para PIC, coloque o circuito SanUSB em modo de gravação pela USB (pino 1 ligado ao Gnd (0V) através de botão ou fio) e conecte o cabo USB do circuito no PC. Se o circuito SanUSB estiver

correto, acenderá o led do pino B7. Pronto, o sistema já está preparado para gravar no microcontrolador, de forma simples e direta, quantos programas .hex você desejar utilizando a interface USB.

Para programar novamente, basta pressionar o botão de gravação no pino 1, desconecte e conecte o cabo USB de alimentação, selecione o programa.hex desejado em Abrir e pressione o botão Gravar&Resetar.

4.11 Gravando o PIC via USB pelo terminal do Linux ou MAC OSX

Esta aplicação é realizada de forma simples em linha de comando no terminal do Mac OSX. Para abrir o terminal é necessário baixar e instalar o software Xcode. Para iniciar a gravação com linhas de comando é importante seguir os seguintes passos:

1. Grave no microcontrolador, somente uma vez, com um gravador específico para PIC com o circuito simples de gravação COM84 descrito nesta apostila ou outro gravador qualquer, o gerenciador de gravação pela USB *Gerenciador.hex*, que é multiplataforma (Linux, Mac OSX e Windows®).
2. Pelo Terminal do Linux ou Mac OSX acesse onde está o executável sanusb, instalado pelo arquivo sanusb.deb, e no Mac OSX acesse a pasta de arquivos SanUSBMacPlugandPlay, onde está o executável sanusb.
3. Após entrar na pasta do exectável sanusb, acesse informações do conteúdo deste arquivo digitando:

. / sanusb-h

A Figura 4.30mostra o *printscreen* de exemplo de um processo de acesso à pasta e também do processo de gravação pelo terminal:

4. Com o circuito SanUSB montado, coloque-o em modo de gravação (pino 1 ligado ao Gnd com botão pressionado ou *jump*) e conecte o cabo USB do circuito no PC.
5. Para gravar no microcontrolador, o firmware desejado, como o exemplo1.hex, deve estar mesmo diretório do executável sanusb, então para a gravação via USB, digita-se:

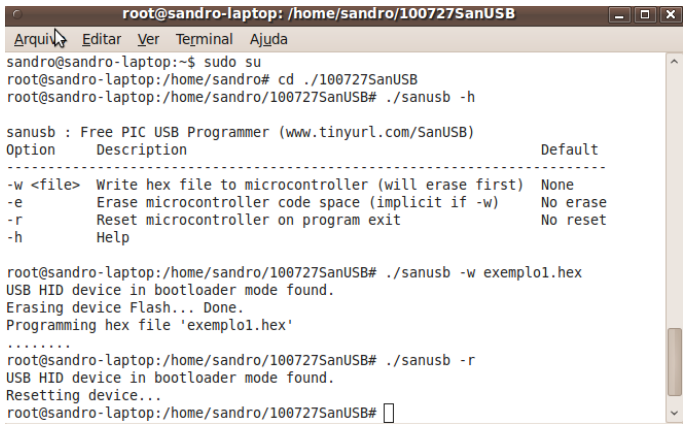
. / sanusb –w exemplo1.hex

6. Depois de gravar, remova o botão ou *jump* de gravação, então reset digitando:

. / sanusb –r

ou simplemente: **. / sanusb –w exemplo1 –r**

Figura 4. 30: Acesso à pasta pelo terminal do LINUX.



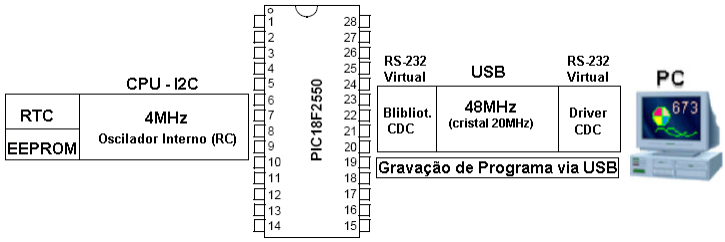
Fonte: Elaborado pelo autor.

Para programar novamente, basta colocar o *jump* de gravação, desconecte e conecte o cabo USB de alimentação, e repita os passos anteriores a partir do passo 6. Se o microcontrolador não for reconecido, feche o terminal, conecte o microcontrolador em outra porta USB, abra um novo terminal e repita repita os passos anteriores a partir do passo 3.

4.12 Sistema Dual Clock

Devido à incompatibilidade entre as frequências necessárias para a gravação e emulação serial via USB e a frequência padrão utilizada pela CPU, temporizadores e interface I^2C, esta ferramenta pode adotar o princípio *Dual Clock* realizado pela instrução *clock_int_4MHz()*, ou seja, utiliza duas fontes de *clock*, uma para o canal USB de 48MHz, proveniente do cristal oscilador externo de 20MHz multiplicada por um *prescaler* interno, e outra para o CPU de 4 MHz, proveniente do oscilador RC interno de 4 MHz, como é ilustrado na Figura 4.31.

Figura 4. 31: Comunicação PIC com PC e via I^2C.



Fonte: Elaborado pelo autor.

Esse princípio de *clock* paralelo,permite que um dado digitado no teclado do computador, trafegue para o microcontrolador em 48 MHz via USB, depois para periféricos como um relógio RTC ou para a memória EEPROM em 4 MHz via I^2C e vice-versa.

4.13 Emulação de Comunicação Serial no Linux

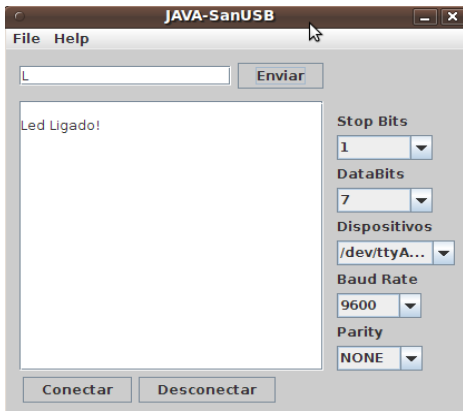
Neste tópico é mostrado um método de comunicação serial bidirecional através do canal USB do PIC18F2550. Uma das formas mais simples, é através do protocolo *Communications Devices Class* (CDC), que é padrão no Linux e que emula uma porta COM RS-232 virtual com o microcontrolador, através do canal USB. Dessa forma, é possível se comunicar com caracteres ASCII via USB através de qualquer software monitor serial RS-232 como o Cutecom, o minicom ou outros aplicativos com interface serial. A biblioteca CDC_ACM padrão no Linux e o programa aplicativo gravado no PIC com a biblioteca CDC (#include <usb_san_cdc.h>), são os responsáveis por esta emulação da porta RS-232 virtual através da USB. A emulação serial é muito utilizada também para "debugar", ou seja, depurar as variáveis de um programa.c, imprimindo-as pela USB durante a execução real do programa. Dessa forma, o programador pode encontrar possíveis erros na programação do firmware.

Após gravar o firmware via USB com o executável linux *sanusb*, instale um software de comunicação serial,como o cutecom, digitando pelo terminal do linux ***#sudo apt-get install cutecom***.Verifique a porta serial virtual criada digitando *dmesg* no terminal. Abra o Cutecom, digitando cutecom no terminal e direcione a porta virtual criada em Device do Cutecom, geralmente a porta é ttyACM0 ou ttyACM1.

É possível também utilizar o programa de comunicação serial Java-SanUSB para emulação serial virtual entre o computador e o microcontrolador.

Após conectar o microcontrolador e abrir o programa de comunicação serial Java-SanUSB em Aplicativos -> Outros, aparecerá a porta serial virtual gerada no Linux (ttyACM0) em Dispositivos. Para listar a porta serial virtual gerada, utilizando o Terminal do Linux, basta digitar ***ls /dev/ttyACM**** . É possível realizar a comunicação depois de clicar em Conectar, como mostra a Figura 4.32.

Figura 4. 32: Interface de comunicação serial em Java para LINUX.



Fonte: Elaborado pelo autor.

4.14 Programa com interrupção externa por botão

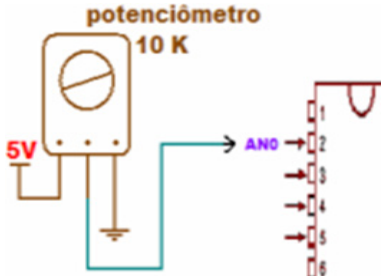
```
#include “SanUSB1.h”
char comando;
short int led;
int x;

#pragma interrupt interrupcao
void interrupcao()
{
        for(x=0;x<5;x++){ // pisca 5 vezes após o pino ser aterrado (botão pressionado)
                nivel_alto(pin_B5); // Pisca Led em B5 tempo_ms(1000);
                nivel_baixo(pin_B5);
                tempo_ms(1000);
        }
}
void main() {
        clock_int_4MHz();
        habilita_interrupcao(ext1);

        while (1){
                nivel_alto(pin_B6); // Pisca Led na função principal tempo_ms(500);
                nivel_baixo(pin_B6);
                tempo_ms(500);
        }
}
```

4.15 Obtenção de um voltímetro através do conversor AD com a variação de um potenciômetro

A Figura 4.33 mostra o esquemático para ligação de um potenciômetro.

```
#include “SanUSB1.h” //Leitura de tensão em mV com variação de um potenciômetro

#pragma interrupt interrupcao
void interrupcao(){}
long int tensao,resultado;
void main() {
clock_int_4MHz();

habilita_canal_AD(AN0);

while(1){                       //ANALÓGICO       DIGITAL(10 bits)
// 5000 mV              1023
// tensao               le_AD10bits(0)

resultado = le_AD10bits(0);
tensao= (5000* resultado)/1023;
sendsw ((char *) "A tensao e' "); // Imprime pela serial
sendnum(tensao); sendsw ((char *) " \r\n"); // Imprime pela serial
nivel_alto(pin_b7);
tempo_ms(500);
nivel_baixo(pin_b7);
tempo_ms(500);          }}
```

Figura 4. 33: Uso de potenciômetro no conversor AD do PIC.

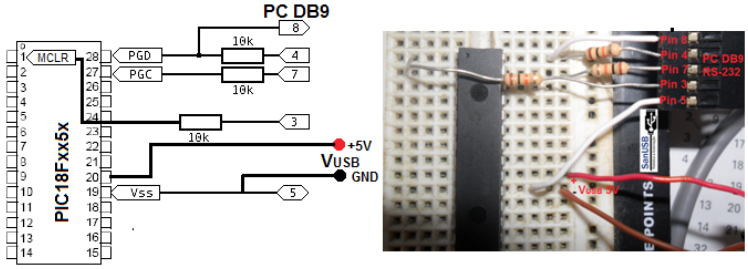


Fonte: Elaborado pelo autor.

4.16 Circuito para gravação do gerenciador.hex

Para este circuito simples de gravação só é necessário 3 resistores de 10k, um cabo serial DB9 (RS-232) e uma fonte externa de 5V, que pode ser obtida da porta USB. O circuito e a foto da Figura 4.34 mostram o esquema simples de ligação dos pinos.

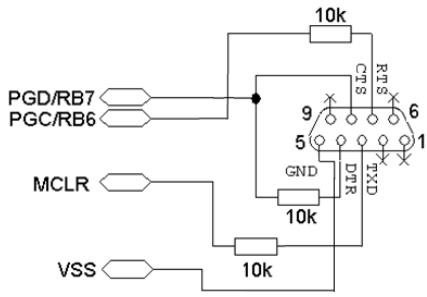
Figura 4. 34: Circuito para gravação do gerenciador.hex.



Fonte: Elaborado pelo autor.

O circuito gravador serial a partir da porta COM DB9 pode ser visualizado na Figura 4.35.

Figura 4. 35: Esquema de ligação do conector serial.



Fonte: Elaborado pelo autor.

Este circuito de gravação funciona com o software PICPgm (detectado como JDM Programmer) ou com WinPic. Com estes softwares gratuitos, é possível gravar o microcontrolador, e mesmo indicando *ERROR: Programming failed*, o arquivo gerenciador.hex mostrou-se gravado corretamente para gerenciar gravações no microcontrolador pela porta USB nos sistemas operacionais Windows®, Linux e Mac OSX.

Após a instalação, execute o programa. Na guia "*Device, Config*", escolha o microcontrolador. Uma vez que o microcontrolador é conectado à porta COM RS-232 de 9 pinos do PC, vá para "*Interface*", selecione " COM84 programmer for serial port", e pressione "Initialize". Se o software disser que a inicialização foi um êxito "*Success*", então o programa está pronto para gravar o gerenciador.hex no microcontrolador.

Para a gravação, selecione em *File Load & ProgramDevice* e depois selecione o arquivo gerenciador.hex. Como citado anteriormente, mesmo que, após a gravação e verificação apareça "*Programmed Failed*", é provável que o gerenciador.hex tenha sido gravado corretamente.

4.17 RGB – controle de cores via Bluetooth

O aplicativo "BT4 SanUSB Bluetooth voice RGB" tem como objetivo controlar as cores RGB (abreviatura do sistema de cores aditivas formado por Vermelho (Red), Verde (Green) e Azul (Blue)) de um ambiente via Bluetooth utilizando um celular Android, podendo realizar a mudança real de cores e brilho de um determinado ambiente de acordo com a simbologia das cores (significado psicológico de cada cor), via Bluetooth, por controle PWM (modulação por largura de pulso) através da Ferramenta SanUSB.

Totalmente gratuito, além de realizar a mudança de cor de um determinado ambiente, permite que uma determinada configuração desejada de cor seja salva em memória permanente do aplicativo, pressionando o botão salvar.

Além disso, este aplicativo possibilita que o usuário modifique o nome do dispositivo Bluetooth ligado a ele, diretamente do aplicativo, inserindo o nome na caixa de texto e pressionando o botão "Enviar Nome BT".

4.17.1 Fundamentação das cores RGB

RGB é a abreviatura do sistema de cores aditivas formado por Vermelho (Red), Verde (Green) e Azul (Blue), conforme Figura 4.36. O propósito principal do sistema RGB é a reprodução de cores em dispositivos eletrônicos como monitores de TV e computador, "data shows", scanners e câmeras digitais, assim como na fotografia tradicional. No sistema RGB, cada cor é definida pela quantidade de vermelho, verde e azul que a compõem. Por conveniência, a maioria dos arquivos digitais atuais usam números inteiros entre 0 e 255 para especificar essas quantidades. O número 0 indica ausência de intensidade e o número 255 indica intensidade máxima.

Figura 4. 36: Processo de formação das cores RGB.

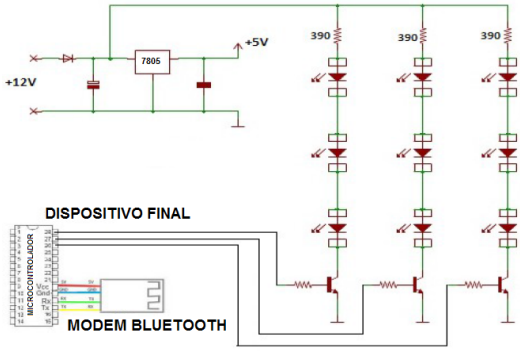
Fonte: Elaborado pelo autor.

Existe a crença de que as cores teriam diferentes efeitos psicológicos sobre as pessoas. Profissionais, principalmente da área da publicidade acreditam que cada cor desperte um efeito psicológico diferente nas pessoas. Existem cores que se apresentam como estimulantes, alegres, otimistas, serenas e tranquilas, dentre outras. A ideia central do controle de cores RGB é fazer com que pessoas utilizem uma determinada cor de acordo com suas necessidades em um determinado ambiente.

4. 17.2 Ligando RGB na Placa SanUSB - Fita de LED RGB

A fita RGB possui saídas R, G e B que devem ser ligadas nas portas B0, B1 e B2 do microcontrolador, elrgbas que vão ser responsáveis por mandar o sinal PWM para a mudança de cor, quando for solicitado pelo aplicativo do celular, a Figura 4.37 ilustra o desenho desse circuito que também deve ter 3 transistores e 3 resistores em série com as saídas RGB.

Figura 4. 37: Ligação completa do circuito RGB e SanUSB.



Fonte: Elaborado pelo autor.

4.17.3 LED RGB Catodo Comum

É necessário identificar com um multímetro as cores de cada pino do LED RGB. É visto naFigura 4.38, um LED RGB catodo comum. Este possui um terminal negativo (GND ou -) e os demais representam as 3 cores: **Red, Green, Blue,** que acendem ao enviar "*output_high*" no pino do PIC. No LED RGB Anodo comum, deve conectar Vcc (+) no terminal comum. Assim, os terminais R, G e B acendem ao enviar "*output_low*" no pino do PIC.

Figura 4. 38: Pinos Led RGB catodo comum.

LED RGB	PLACA SanUSB
R - Vermelho	Pino b2
G - Verde	Pino b1
B - Azul	Pino b0

Fonte: Elaborado pelo autor.

4.17.4 O Compilador

Foi escrito um código em linguagem de programação C, que deve ser gravado no microcontrolador para que ele possa receber o sinal do modem Bluetooth, manipulando as portas B0, B1 e B2, que estão conectadas ao circuito RGB, então o microcontrolador recebe o sinal PWM e o transforma em uma determinada cor, o compilador a ser é usado é MPLABX IDE que é um software para desenvolver aplicações para microcontroladores da Microchip e controladores de sinal digital.

4.17.5 O Aplicativo

O aplicativo a ser usado no projeto é o BT4SanUSB, que é um aplicativo gratuito que pode ser baixado livremente na internet, é um aplicativo Android para comunicação com modem Bluetooth slave HC-04 ou HC-06. Este aplicativo realiza também a mudança de cores e brilho, via Bluetooth, por controle PWM de fita/LED RGB conectado ao PIC e permite que uma determinada configuração desejada de cor seja salva em memória permanente do aplicativo, pressionando o botão salvar.

Este aplicativo é capaz também de ligar leds ou aparelhos por comando de voz. O comando de voz pode ser gravado na memória permanente do aplicativo pressionando o botão correspondente por dois segundos. Telas do aplicativo são mostradas na Figura 4.39.

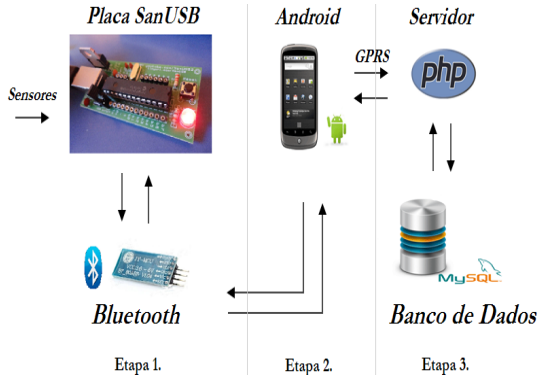
Figura 4. 39: O aplicativo BT4SanUSB - RGB.

Fonte: Elaborado pelo autor.

4.17.6 A aplicação

Após a montagem de todo o circuito, o celular deve enviar o sinal do aplicativo para o microcontrolador através do modem bluetooth, que por sua vez se comunica com a fita RGB através das portas B0, B1 e B2, a Figura 4.40 ilustra bem esse processo.

Figura 4. 40: A aplicação do projeto Bluetooth, Android e RGB.



Fonte: Elaborado pelo autor.

A placa RGB construída para esta aplicação está ilustrada na Figura 4.41.

Figura 4. 41: Led RGB.

Fonte: Elaborado pelo autor.

A Figura 4.42 ilustra a operação do software educacional durante a formação de cores real no LED RGB e virtual no dispositivo Android, nos momentos em que são selecionadas somente a cor vermelha, somente a cor azul, somente a cor verde e a seleção de todas as cores formando e emissão da cor branca no aplicativo e no LED RGB.

Figura 4. 42: Operação do software educacional BT4SanUSB durante a formação de cores real no LED RGB e virtual no dispositivo Android.

Fonte: Elaborado pelo autor.

Capítulo 5

Modem WIFLY e o protocolo de comunicação WIFI

A Figura 5.1 ilustra um processo de monitoramento e aquisição de dados *online* via WiFi, baseado no módulo Wifly RN-XV 171. Para promover a comunicação entre a placa de aquisição e o servidor *online*, são necessários apenas quatro pinos conectados entre a placa e o módulo Wifly, dos quais dois destes são para tensão de alimentação e outros dois para transferência de dados.

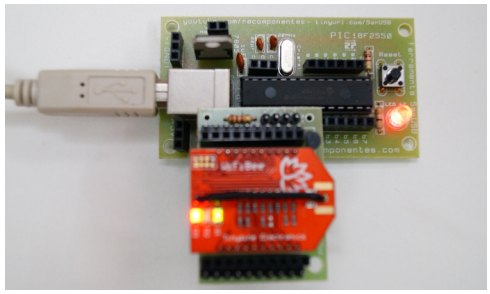
Figura 5. 1: Ilustração de processo de monitoramento e aquisição de dados ***online*** com módulo Wifly.

Fonte: Elaborado pelo autor.

Segue abaixo o passo a passo para implementar a comunicação WiFi com a placa SanUSB e o modem Wifly RN-XV.

Conecte o modem WiFly na ferramenta SanUSB (placa ou circuito protoboard), como a foto da Figura 5.2 (observe os leds do modem piscando).

Figura 5. 2: Modem WiFi e placa SanUSB.



Fonte: Elaborado pelo autor.

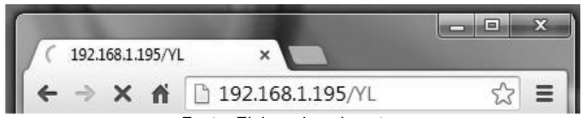
1. Insira no firmware destacado a seguir o nome da sua rede, a senha, o IP intranet e o gateway. Compile o código no compilador CCS e em seguida grave no PIC.

2. Escolha um valor para o IP do seu dispositivo, neste caso foi inserido 195.

3. Observe, após alguns segundos, que somente o led verde mantem-se piscando. Isso indica que o seu modem já encontra-se conectado à rede.

4. Abra o Google Chrome e digite no *browser*: 192.168.1.195/YL, conforme Figura 5.3. Observe se o led b7 da placa SanUSB é aceso.

Figura 5. 3: Comando WiFi usando o ***browser***.



Fonte: Elaborado pelo autor.

Fonte: Elaborado pelo autor.

1. Os comandos, conforme destacados em azul e vinho no firmware abaixo, podem ser inseridos para testar a comunicação:

   a. 192.168.1.195/YL → liga o led b7;
   b. 192.168.1.195/YD → desliga o led b7;
   c. 192.168.1.195/YT → comuta o led b7;

Verifique que, durante a comunicação, o led verde do modem Wifly pára de piscar e permanece aceso.

Conecte um relé no pino b7 e insira a carga. Pronto! Você já pode acionar dispositivos/equipamentos (lâmpadas, motores...) via internet de qualquer lugar do mundo via IP.

O código principal (main) do *firmware* para esta aplicação, foi desenvolvido pelo grupo SanUSB, utilizando o ambiente de desenvolvimento MPLABX, com a versão livre do compilador C18 multiplataforma, e também para CCS, inseridoa seguir. Mais detalhes podem ser conferidos em https://github.com/SanUSB.

```
#include "SanUSB48.h"
 const rom unsigned char pg[] =
{144,168,168,160,94,98,92,98,64,100,96,96,64,158,150,26,20,134,222,220,232,202,220,232,90,168,242
,224,202,116,64,232,202,240,232,94,208,232,218,216,26,20,26,20,120,198,202,220,232,202,228,124,1
20,210,204,228,194,218,202,64,230,228,198,122,68,208,232,232,224,230,116,94,94,200,222,198,230,9
2,206,222,222,206,216,202,92,198,222,218,94,230,224,228,202,194,200,230,208,202,202,232,94,204,2
22,228,218,164,202,230,224,222,220,230,202,126,204,222,228,218,214,202,242,122,200,136,180,238,
172,216,164,104,164,142,106,230,168,174,106,156,202,216,222,100,164,216,156,160,172,138,162,100
,178,218,198,108,154,162,76,210,204,226,74,100,96,76,202,220,232,228,242,92,96,92,230,210,220,20
6,216,202,122,0};

const rom unsigned char pg1[] = {76,202,220,232,228,242,92,98,92,230,210,220,206,216,202,122,0};

 const rom unsigned char pg2[] =
{76,230,234,196,218,210,232,122,166,234,196,218,210,232,68,124,120,94,210,204,228,194,218,202,12
4,120,144,98,124,148,222,210,220,64,232,210,220,242,234,228,216,92,198,222,218,94,166,194,220,17
0,166,132,120,94,144,98,124,120,160,124,120,94,144,98,124,130,198,198,202,230,230,64,120,194,64,
208,228,202,204,122,68,208,232,232,224,230,116,94,94,200,222,198,230,92,206,222,222,206,216,202,
92,198,222,218,94,230,224,228,202,194,200,230,208,202,202,232,94,198,198,198,126,214,202,242,12
2,96,130,224,102,102,100,158,130,208,220,154,202,100,200,136,180,238,172,216,164,104,164,142,10
6,230,168,174,106,156,202,216,222,100,164,216,156,160,172,138,162,100,178,218,198,68,124,152,22
2,206,230,120,94,194,124,120,94,160,124,120,94,198,202,220,232,202,228,124,26,20,0};

#pragma udata
char rede[] = "SanUSB";      // Your WiFi SSID name
char senha[] = "Laese";      // Your WiFi password
char ip[] = "192.168.1.195";  // Set intranet static IP (DHCP OFF). In this the case the static IP .195 is
based on the gateway IP 192.168.1.1. // To check the IP of your gateway, type ipconfig or ifconfig at the
prompt or terminal on the computer
// Default Wifly RN-XV SanUSB IP: 192.168.1.195 Port: 80. The Wifly IP must be xxx.xxx.x.1xx , with "1"
in last byte for the gateway IP.

short int pisca=0, AT1=0,  AT2=0, start=0, flagsw=0;
char  comando[64], n=0, m=0;
unsigned int ADvalue = 0, i=0;
unsigned char smid;

#pragma interrupt interrupcao
void interrupcao(){
if (serial_interrompeu)  {
    serial_interrompeu=0;
    comando[n] = le_serial(); m=n;

switch (comando[n]){

            case 'C':
            AT1=1;
            break;

            case 'I':
            AT2=1;
            break;

            case 'A':
            AT1=1;
            break;

            case 'S':
            AT1=1;
            break;

            case '*':
            AT1=1;
            break;
           case 'Y':
            {n=0;comando[0] = 'Y';}
             break;
```

```
      }
 //***************possibility 1->character***********************
                    if (comando[0]== 'Y' && n==1){comando[0]== 32;
                            switch(RCREG)
                                 {
                                case 'L':    nivel_alto(pin_b7); //type in google chrome address bar:
192.168.1.195/YL
                                        flagsw=1;
                                break;

                                case 'D': nivel_baixo(pin_b7); //type in google chrome address bar:
192.168.1.195/YD
                                break;
                                case 'P':       nivel_alto(pin_b7);   //type in browser address bar:
192.168.1.195/YP
                                break;

                                case 'T': inverte_saida(pin_b7);  //type in browser address bar:
192.168.1.195/YT to toggle Led and
                                //open a HTML page
                                        flagsw=1;
                                break;
                                 }
                                      }
//***************possibility 2-> word*****************************
                    if ( comando[1]== 'O' && comando[2]== 'N')
{
                                 nivel_alto(pin_b7);  // ON
                                }

                    if ( comando[1]== 'O' && comando[2]== 'F' && comando[3]== 'F')
                                {
                                 nivel_baixo(pin_b7); // OFF
                                }
       ++n; if(n>=64){n=0;}
}
               }

#include "confws.h"

void main(void)
{
   clock_int_48MHz();
   habilita_interrupcao(recep_serial);
   taxa_serial(9600);
   habilita_canal_AD(AN0);
iniws();
  while(1)
   {
        if (flagsw==1) {flagsw=0;
                    ADvalue=le_AD10bits(0);
                    tempo_ms(500);
sendr((rom char*)pg);
                    for(i=0;rede[i]!='\0';i++)
                    {smid=(rede[i]*(('@'+REG)>>5));sputc(smid);}
                    sendr((rom char *)pg1);
                    sendnum(ADvalue);
                    sendr((rom char*)pg2);
                    tempo_ms(500);
                       }
   }
}
```

Capítulo 6
Periféricos internos do microcontrolador

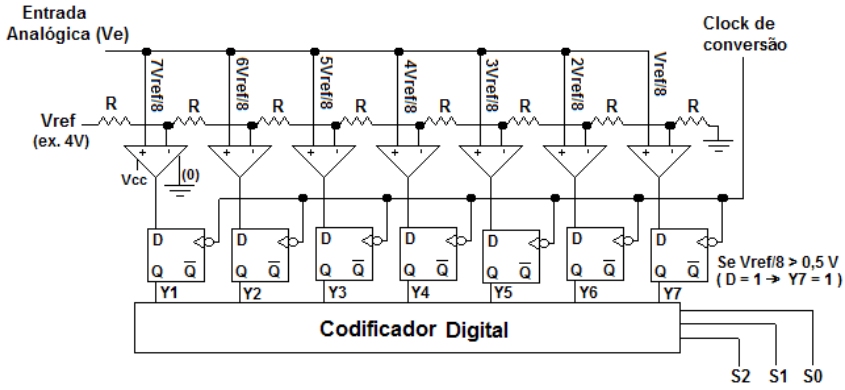
6.1 Definições deconversão AD (analógico/digital)

Uma forma de representar um método de conversão AD é através de um conjunto de amplificadores comparadores de tensão, conectados em paralelo à tensão analógica a ser convertida em valor digital. Cada comparador possui na primeira entrada uma fração da tensão de referência em função de R e, na outra entrada, a tensão analógica a ser convertida. À medida que a tensão analógica de entrada for aumentando e for maior que a tensão de referência, os amplificadores comparadores são saturados e apresentam o nível lógico alto na saída (vcc). As saídas de todos os comparadores entram em um codificador de prioridade, que indica na saída, o valor binário correspondente à entrada analógica ativa. Este conversor A/D, baseado em comparadores paralelos é um dos mais rápidos conversores A/D e necessita de 2^N-1 amplificadores comparadores para um conversor de N bits. A Figura 6.1 apresenta esse tipo de conversor AD para conversão de 3 bits com 7 comparadores.

Caso e tensão de referência (Vref) seja 4V, a resolução desse conversor AD de 3 bits é dada por Vref/(2^N-1), ou seja, 0,5 V. Dessa forma, a cada 0,5 V na tensão de entrada, as saídas Y são *setadas* como ilustrado na Tabela 6.1.

A resolução deste conversor é de apenas 3 bits. Para se ter uma resolução de 8 bits são necessários 255 comparadores. Como se pode notar a complexidade do conversor aumenta à razão 2^N onde 'n'é o número de bits desejado como resolução. Uma das grandes vantagens deste método é a sua rapidez na conversão, uma vez que só está limitada pela velocidade do comparador e do codificador digital.

Figura 6. 1: Conversor A/D de 3 bits.



Fonte: Elaborado pelo autor.

Tabela 6. 1: Relação entre a tensão de entrada e os bits de saída do decodificador digital.

Ve(V)	Y7	Y6	Y5	Y4	Y3	Y2	Y1	S2	S1	S0
<0,5	0	0	0	0	0	0	0	**0**	**0**	**0**
>=0,5	0	0	0	0	0	0	**1**	**0**	**0**	**1**
>=1	0	0	0	0	0	**1**	1	**0**	**1**	**0**
>=1,5	0	0	0	0	**1**	1	1	**0**	**1**	**1**
>=2	0	0	0	**1**	1	1	1	**1**	**0**	**0**
>=2,5	0	0	**1**	1	1	1	1	**1**	**0**	**1**
>=3	0	**1**	1	1	1	1	1	**1**	**1**	**0**
>=3,5	**1**	1	1	1	1	1	1	**1**	**1**	**1**

Fonte: Elaborado pelo autor.

6.2 Conversor A/D

Com foi visto, o objetivo do conversor analógico-digital (AD) é converter um sinal analógico. No caso de microcontroladores PIC, estes são geralmente de 10 bits eVref igual a 5V. Algumas configurações permitem ainda que os pinos A3 e A2 sejam usados como referência externa positiva e negativa, fazendo com que uma leitura seja feita em uma faixa de tensão mais restrita como, por exemplo, de 1 a 3 Volts.

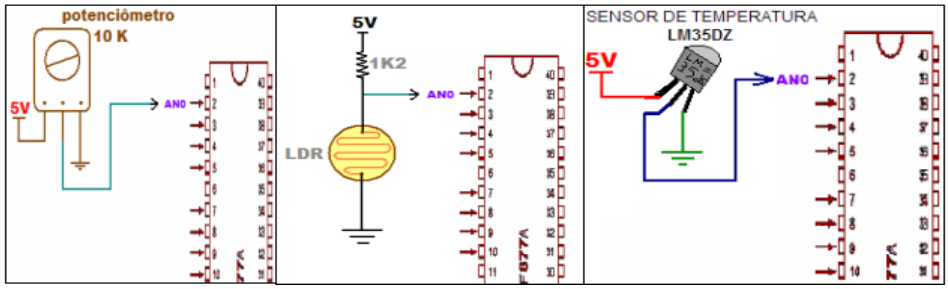
Em linguagem C, o conversor AD pode ser ajustado para resolução de 8, armazenando o resultado somente no registro ADRESH, ou 10 bits, armazenando o resultado nos registros ADRESH (dois bits mais significativos) e ADRESL (oito bits menos significativos).

Para um conversor A/D com resolução de 10 bits e tensão de referência padrão de 5V, o valor de cada bit será igual a $5/(2^{10} - 1) = 4{,}8876$ mV, ou seja, para um resultado igual a 100 (decimal), teremos uma tensão de 100* 4,8876 mV = 0,48876 V. Note que a tensão de referência padrão (Vref) depende da tensão de alimentação do PIC que normalmente é 5V. Se a tensão de alimentação for 4V, logo a tensão de referência (Vref) também será 4V.

Para um conversor A/D com resolução de 10 bits e tensão de referência de 5V, o valor de cada bit será igual a $5/(2^{8} - 1) = 19{,}6078$ mV, ou seja, para um resultado igual a 100 (decimal), é necessário uma tensão de 100 * 19,6078 mV = 1,96078 V, quatro vezes maior.

É comum se utilizar o conversor AD com sensores de temperatura (como o LM35), luminosidade (como LDRs), pressão (STRAIN-GAGE), tensão, corrente, humidade, entre outros (ver Figura 6.2).

Figura 6. 2: Uso de periféricos no conversor A/D.



Fonte: Elaborado pelo autor.

Para utilizar este periférico interno, basta:

habilita_canal_AD(0); { //(Seleção dos pinos analógicos 18F2550)

Observa-se que a nomenclatura dos canais analógicos de cada modelo, dentro da biblioteca do MPLABX na pasta *Device*. Depois, no laço infinito, basta selecionar o canal para leitura, esperar um tempo para a seleção física e então ler o canal AD.

valor=le_AD10bits (0);

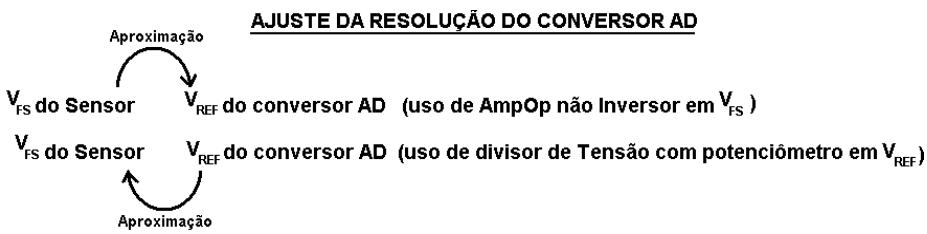
tempo_ms (1); // aguarda um milisegundo para comutar para o canal 0

valor=le_AD10bits (1);

6.3 Ajuste de resolução do sensor e do conversor AD de 8 bits

O ajuste da resolução do conversor AD se dá aproximando a tensão de fundo de escala do sensor (V_{FS}) à tensão de referencia do conversor (V_{REF}). Para isso existem duas técnicas de ajuste por *Hardware*, conforme Figura 6.3.

Figura 6. 3: Ajuste da resolução do conversor AD.



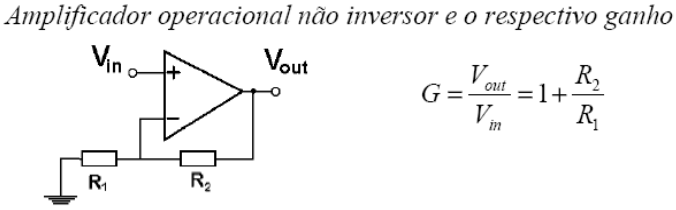
Fonte: Elaborado pelo autor.

Para este tópico é utilizado como exemplo de ajuste da resolução do conversor AD, o sensor de temperatura LM35 que fornece uma saída de tensão linear e proporcional com uma resolução de 10mV a cada °C.

6.4 Ajuste da tensão de fundo de escala com AMPOP

Para conversores AD de 8 bits e V_{REF} de 5V, a resolução máxima é de 19,6mV (R= V_{REF} / (2^n-1). Dessa forma, como a Resolução do sensor é 10mV/°C (R_S), é necessário aplicar um ajuste de resolução com um ganho na tensão de fundo de escala do sensor para que cada grau possa ser percebido pelo conversor do microcontrolador. A forma mais comum de ganho é a utilização de amplificadores operacionais não inversores. Veja mais detalhes no material de apoio no final dessa apostila. A tensão de fundo de escala (V_{FS}) está relacionada à Temperatura Máxima desejada de medição (T_{MAX}), onde $V_{FS} = R_S(10mV/°C)* T_{MAX}$e o Ganho (G) de aproximação da tensão de fundo de escala (V_{FS}) à tensão de referencia (V_{REF}) é dado por G = V_{REF} / V_{FS}(ver Figura 6.6), ou seja, para uma Temperatura Máxima desejada de 100°C, o ganho deve ser aproximadamente 5.

Figura 6. 4: AMP-OP não inversor.



Fonte: Elaborado pelo autor.

A aproximação da tensão de fundo de escala (V_{FS}) à tensão de referencia (V_{REF}) é realizada para diminuir a relevância de ruídos em determinadas faixas de temperatura.

6.5 Ajuste da tensão de referência com Potenciômetro

Outra forma mais simples de ajuste por Hardware (aumento da resolução do conversor AD) é a aproximação da tensão de referencia (V_{REF}) à tensão de fundo de escala (V_{FS}) através da diminuição da tensão de referência (V_{REF}) com o uso de um potenciômetro (divisor de tensão). Por exemplo, um conversor AD de 8 bits com uma tensão de referência (V_{REF}) de 2,55V no pino A3, apresenta uma resolução de 10mV por bit (2,55/(2^8-1)), ou seja, a mesma sensibilidade do sensor LM35 de 10mV/°C . Percebe variação de cada °C.

6.6 Conversor AD de 10 bits

Para conversores de 10 bits, com maior resolução (4,89 mV), o ajuste (escalonamento) é realizado geralmente por software, em linguagem C, que possui um elevado desempenho em operações aritméticas.

OBS.: O ganho de tensão de um circuito poderia ser simulado por software com os comandos:**Int32 valorsensor= read_adc();**

Int32 VFS = 4 * Valorsensor;

A fórmula utilizada pelo programa no PIC para converter o valor de tensão fornecido pelo sensor LM35 em uma temperatura é:

ANALÓGICO		**DIGITAL**
5V = 5000mV	->	1023
T(°C)* 10mV/°C	->	(int32)read_adc()

T (ºC) = 500 * (int32)read_adc()/1023

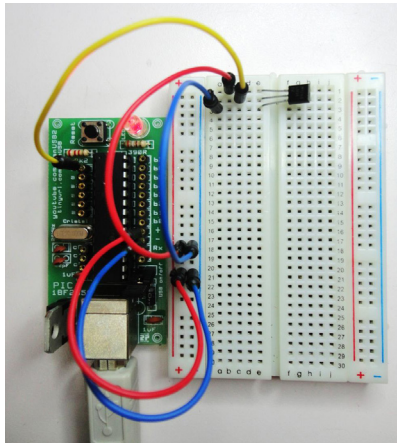
Onde (int32)read_adc() é o valor digital obtido a partir da temperatura (T(ºC)) analógica medida. Esta variável é configurada com 32 bits (int32), porque ela recebe os valores dos cálculos intermediários e pode estourar se tiver menor número de bits, pois uma variável de 16 bits só suporta valores de até 65.535. A tensão de referência do conversor é 5V e como o conversor possui 10 bits de resolução, ele pode medir 1023 variações.

6.7 Leitura de temperatura com o LM35 através do conversor AD

```
#include “SanUSB1.h”
#pragma interrupt interrupcao
void interrupcao(){}
long int temperatura,resultado;

void main() {
clock_int_4MHz();
habilita_canal_AD(AN0);
while(1){
resultado = le_AD10bits(0);

temperatura=430*read_adc()/1023; //Vref = 4,3V devido à queda no diodo, então (430*temp)
sendsw ((char *) " Temperatura do LM35 = "); // Imprime pela serial
sendnum(tensao); sendsw ((char *) " \r\n"); // Imprime pela serial
nivel_alto(pin_b7);
tempo_ms(500);
nivel_baixo(pin_b7);
tempo_ms(500);            }}
```

Na Figura 6.5 é possível visualizar a conexão do sensor à placa SanUSB.

Figura 6. 5: Sensor de temperatura LM35 montado em protoboard.

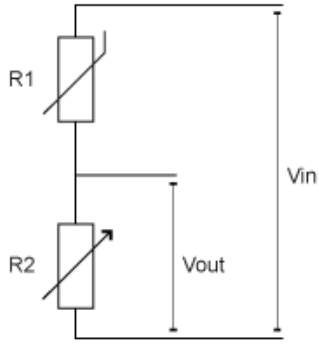


Fonte: Elaborado pelo autor.

6.8 Termistor

Um termistor (R1) é uma resistência variável com a temperatura. Na realidade todas as resistências variam com a temperatura, só que os termistores são feitos para terem uma grande variação com a temperatura. A ligação da tensão de saída (Vout) do termistor (R1) ao microcontolador é muito simples através do acoplamento de um potenciômetro (R2) de 10k, por exemplo, como mostra a Figura 6.6.

Figura 6. 6: Termistor.



Fonte: Elaborado pelo autor.

Convém lembrar que a resposta de um termistor não é linear. Uma forma muito comum de linearização do termistor é por modalidade da tensão, onde um termistor NTC é conectado em série com um resistor normal formando um divisor de tensão. O circuito do divisor contém uma fonte de tensão de referência (Vref) igual a 2,5V. Isto tem o efeito de produzir uma tensão na saída que seja linear com a temperatura. Se o valor do resistor R_{25C} escolhida for igual ao da resistência do termistor na temperatura ambiente (25ºC), então a região de tensão linear será simétrica em torno da temperatura ambiente.

Capítulo 7
Memórias do microcontrolador

O microcontrolador apresenta diversos tipos de memória, entre elas:

7.1 Memória de programa

A memória de programa *flash*, é o local onde são gravados o códigohexadecimal do programa compilado. Essa memória é uma espécie de EEPROM (memória programável e apagável eletronicamente), mas só pode ser gravada e apagada completamente e não byte a byte, o que a torna mais econômica.

7.2 Memória de instruções

A memória de instruções, que é uma espécie de BIOS (*binary input and output system*), se localiza dentro da CPU para comparação com o código hexadecimal do programa que está sendo processado e execução de uma ação correspondente.

7.3 Memória EEPROM interna

A maioria dos modelos da família PIC apresenta memória EEPROM interna, com dimensões de 128 ou 256 bytes. Em algumas aplicações, a EEPROM interna é muito útil para guardar parâmetros de inicialização ou reter valores medidos durante uma determinada operação de sensoreamento.

O PIC18F2550 contém 256 bytes (posições 0 a 255) de EEPROM interna, que podem ser escritas facilmente utilizando a instrução **write_eeprom(posicao, valor);** e lidas com a instrução **valor2=le_eeprom (posicao);**. O projeto de controlde de acesso com teclado matricial abaixo mostra o uso desta memória.

7.4 Memória de dados (RAM) e registros de funções especiais

A memória RAM do microcotrolador 18F2550 possui 2 kbytes disponíveis para propósito geral (entre 000h a 7FFh). No final da RAM (entre F60h e FFFh) estão localizados os registros de funções especiais (SFR), que servem para configurar os periféricos internos do microcontrolador.

Esses registros podem ser configurados bit a bit através do seu nome de identificação ou utilizando a função **REGISTRO**bits.**BIT**. Mais detalhes dos bits de cada registro podem ser verificados no *datasheet* do microcontrolador.

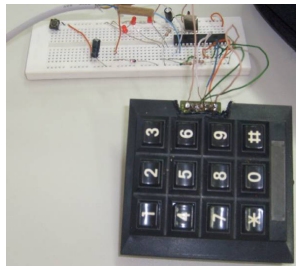
7.5 Exemplo de aplicação com memória

7.5.1 Controle de acesso com teclado matricial

O teclado matricial é geralmente utilizado em telefones e em controle de acesso de portas com senhas pré-definidas. O controle de acesso é feito, na maioria das vezes, com sistemas microcontrolados por varredura das linhas, aterrando individualmente as colunas

do teclado. Caso alguma tecla seja pressionada, o pino da tecla correspondente também será aterrado e indicará a tecla digitada. A Figura 7.1 mostra um exemplo de teclado matricial e a conexão de linhas e colunas.

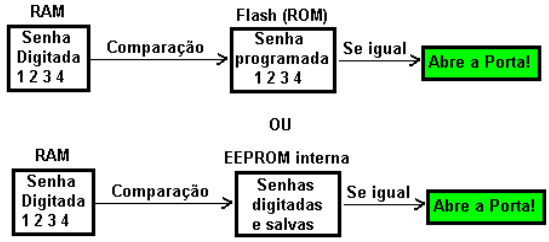
Figura 7. 1: Teclado Matricial.



Fonte: Elaborado pelo autor.

Para reconhecer uma senha digitada em um teclado matricial é necessário armazenar o valor das teclas digitadas em seqüência em alguma memória, como por exemplo, na memória de dados RAM (pode-se utilizar quaisquer posições dos 2Kbytes disponíveis entre 000h a 7FFh), depois compará-la com uma senha pré-programada contida na memória de programa *flash* ("ROM") ou na EEPROM interna (ver Figura 7.2).

Figura 7. 2: Formas de comparação com a senha pré-programada.



Fonte: Elaborado pelo autor.

Note que o programa de controle de acesso em anexo utiliza a EEPROM interna para possibilitar a inserção de novas senhas na EEPROM, sem a necessidade de gravar novamente a memória de programa do microcontrolador.

7.5.2 Ponteiros

Ponteiros armazenam endereços de memória de programa.

Exemplo para declarar um ponteiro:

unsigned int16 p=100; //ponteiro igual a posição 100

***p='7';** // Conteúdo endereçado por p é igual a '7'(ASC II) ou 0x37.

++p; //Incrementa a posição para receber próximo dado.

Programa de controle de acesso com armazenamento de senhas na EEPROM interna pelo próprio teclado através de uma senha de administrador (mestre):

```
///Teclado Matricial insere novas senhas pelo teclado com a  senha mestre//
/////oscilador interno 4 de MHz//////////////////////////////////////////
//Se faltar energia ou existir um reset, as senhas armazenadas não são
//perdidas e é possivel armazenar novas senhas após a última senha gravada
//É possível apagar senhas lendo a EEPROM e zerando as posições da senha ex.:write_eeprom( endereco, 0 );

#include “SanUSB1.h”

char caract,tecla0,tecla1,tecla2,tecla3;
unsigned int16  p=100,i,j;
unsigned int mult=8,k=0,n=0;
int1 led,flag=0,flag2=0,flag3=0;

////////////////////////////////////////////////////////////////////////////
#pragma interrupt interrupcao
void interrupcao()
{
--mult;
if (!mult)
{mult=8; // 8 x0,5s - 4 seg
p=100; tecla0='F';tecla1='F';tecla2='F';tecla3='F'; // volta a posição de origem a cada 4 seg
}}
////////////////////////////////////////////////////////////////////////////
void main() {
clock_int_4MHz();

enable_interrupts (global); // Possibilita todas interrupcoes
enable_interrupts (int_timer1); // Habilita interrupcao do timer 1
setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8);// inicia o timer 1 em 8 x 62500 = 0,5s
set_timer1(3036);

//write_eeprom( 239, 0);//Pode apagar toda a memória de senhas zerando k
if(le_eeprom(239)>0 &&le_eeprom(239)<40) {k=le_eeprom(239);} // Carrega a útima posição livre da eeprom (k) antes do reset armazenada em 239

while (1)
{
// Reconhecimento de tecla por varredura
nivel_baixo(pin_b0);nivel_alto(pin_b1);nivel_alto(pin_b2);
if(input(pin_b3)==0) {*p='1'; flag=1; while(input(pin_b3)==0);tempo_ms(200);}
if(input(pin_b4)==0) {*p='4'; flag=1; while(input(pin_b4)==0);tempo_ms(200);}
if(input(pin_b5)==0) {*p='7'; flag=1; while(input(pin_b5)==0);tempo_ms(200);}
if(input(pin_b6)==0) {*p='*'; flag=1; while(input(pin_b6)==0);tempo_ms(200);}

nivel_alto(pin_b0);nivel_baixo(pin_b1);nivel_alto(pin_b2);
if(input(pin_b3)==0) {*p='2'; flag=1; while(input(pin_b3)==0);tempo_ms(200);}
if(input(pin_b4)==0) {*p='5'; flag=1; while(input(pin_b4)==0);tempo_ms(200);}
if(input(pin_b5)==0) {*p='8'; flag=1; while(input(pin_b5)==0);tempo_ms(200);}
if(input(pin_b6)==0) {*p='0'; flag=1; while(input(pin_b6)==0);tempo_ms(200);}

nivel_alto(pin_b0);nivel_alto(pin_b1);nivel_baixo(pin_b2);
if(input(pin_b3)==0) {*p='3'; flag=1; while(input(pin_b3)==0);tempo_ms(200);}
if(input(pin_b4)==0) {*p='6'; flag=1; while(input(pin_b4)==0);tempo_ms(200);}
if(input(pin_b5)==0) {*p='9'; flag=1; while(input(pin_b5)==0);tempo_ms(200);}
if(input(pin_b6)==0) {*p='!'; flag=1; while(input(pin_b6)==0);tempo_ms(200);}
```

```
// Guarda tecla pressionada
if (flag==1) {
if(p==100){tecla0=*p;}
if(p==101){tecla1=*p;}
if(p==102){tecla2=*p;}
if(p==103){tecla3=*p;flag2=1;} //A flag2 avisa que senha foi digitada completamente
mult=4; //cada tecla tem 2 seg para ser pressionada a partir da primeira
printf ("\r\nValor das teclas digitadas: %c %c %c %c\r\n",tecla0,tecla1,tecla2,tecla3);
printf "Endereco para onde o ponteiro  p aponta: %lu\r\n",p);
++p;  // Incrementa posição para próxima tecla
if(p>103){p=100;}                                          }

//*************************************************************************

if (tecla0=='3' && tecla1=='6'&& tecla2=='9'&& tecla3=='!'&& flag2==1) {
flag3=1; //Indica que a senha mestre autorizou e a nova senha pode ser armazenada
flag2=0; //Garante que somente a próxima senha, diferente da senha mestre, será armazenada
nivel_alto(pin_c0);printf ("\r\nSenha Mestre!\r\n");tempo_ms(1000); nivel_baixo(pin_c0);}

//*************************************************************************

if (flag2==1 && flag3==1) { //Se a senha mestre já foi digitada (flag3) e a nova senha do usuário também
foi digitada (flag2)
write_eeprom( 4*k, tecla0 ); //Grave a nova senha
write_eeprom( (4*k)+1, tecla1 );
write_eeprom( (4*k)+2, tecla2 );
write_eeprom( (4*k)+3, tecla3 );
++k; // incremente as posições para armazenar nova senha

printf ("\r\nSenha armazenada\r\n");

write_eeprom( 239, k); printf("proximo k=%u\r\n",k);//Guarda a útima posição livre antes do reset na
posição 239 da EEPROM

flag3=0; //Zera a flag3 da nova senha

//*************************************************************************
for(i=0; i<6; ++i) //Lê EEPROM
{
for(j=0; j<40; ++j) {printf("%2x ", le_eeprom(i*40+j) );}//Leitura da eeprom interna

printf("\r\n");
}
}
//*************************************************************************
// Compara conjunto de teclas pressionadas com senhas armazenadas na eeprom
if (flag2==1) {
for(n=0;n<=k;n++)
{
if    (tecla0==le_eeprom(4*n)    &&    tecla1==le_eeprom(4*n+1)    &&    tecla2==le_eeprom(4*n+2)&&
tecla3==le_eeprom(4*n+3))
{ nivel_alto(pin_c0); printf ("\r\nAbre a porta!\r\n");tempo_ms(3000); nivel_baixo(pin_c0);} } }// abre a porta
//*************************************************************************
flag=0; flag2=0; //Zera as flags para que novas senhas possam ser digitadas
led = !led; // inverte o led de operação
output_bit (pin_b7,led);
tempo_ms(100);                }}
```

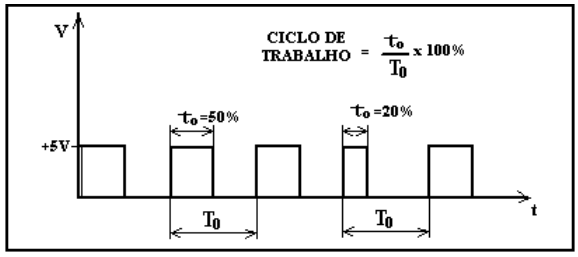
Capítulo 8
Modulação por largura de pulso pelo CCP

O Periférico interno independente CCP (Capture/Compare/PWM) é responsável por comparação de sinais, bem como, gerar um sinal de sáida em PWM (CCP-1CON=0x0C), ou seja, modulação por largura de pulso (*pulse width modulation*). Dessa forma, é possível realizar o PWM pelo periférico interno CCP do microcontrolador ou por software através de emulação pelo processador do microcontrolador.

A geração do PWM é produzida, geralmente, pelo chaveamento de uma tensão com amplitude constante (+5V por exemplo), tendo em vista que quanto menor a largura dos pulsos emitidos na base de uma chave eletrônica, como um transistor, menor é a saturação do transistor e, consequentemente, menor a tensão lado da carga, resultante do chaveamento.

O período T_0 é o intervalo de tempo que se registra o período repetitivo do pulso e o τ_0 é o ciclo de trabalho (*duty-cicle*), ou seja, o tempo em que o pulso permanece com a amplitude em nivel lógico alto, conforme Figura 8.1.

Figura 8. 1: Ciclo de trabalho.



Fonte: Elaborado pelo autor.

No módulo CCP, o registro PR2 é carregado para servir de referência de contagem para o timer 2 de 8 bits com prescaler (multiplicador de tempo de geralmente 4 ou 16). Quando o timer 2chega ao valor de PR2, seta o flip-flop do CCP para gerar o período (T_0)e, consequentemente, a frequência desejada do PWM (freqPWM).

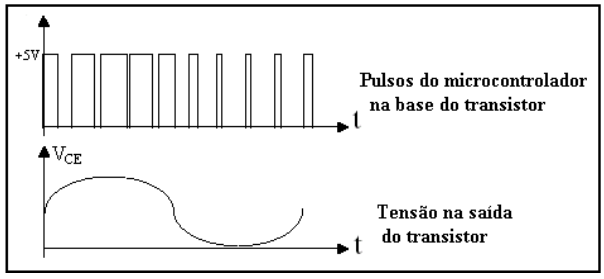
O valor digital (Vdig) de 10 bits do ciclo de trabalho, obtido do valor percentual (0 a 100)inserido pelo usuário: **Vdig = (Valor% / 100) * (PR2+1) * 4**. O Vdig é carregado em CCPR1L,que guarda os 8 bits mais significativos do valor digital, e nos bits5 e 4 do CCP-1CON são carregados os dois bits menos significativos do valor digital, e serve de referência para o timer2 resetar o ciclo de trabalho. É possível verificar no datasheet do microcontrolador os componentes para setar e resetar a onda PWM gerada.

Capítulo 9

Controle PWM por software de velocidade de um motor CC

A finalidade deste controle de velocidade com a modulação de tensão por largura de pulsos (PWM) é realizar uma conversão digital-analógica que acontece devido à impedância inerente do circuito em alta frequência (ver Figura 9.1).

Figura 9. 1: PWM.



Fonte: Elaborado pelo autor.

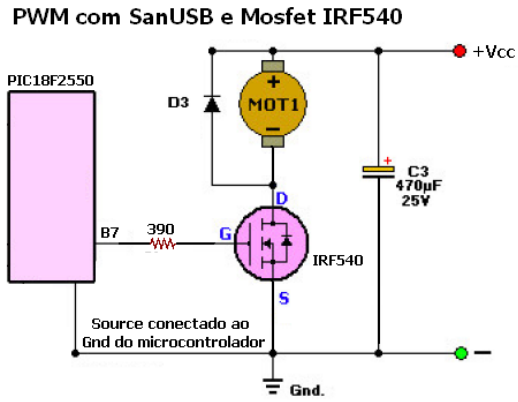
O programa abaixo mostra o controle de velocidade de um motro CC por PWM com período constante de 20ms, onde cada incremento ou decremento da onda quadrada corresponde a 1ms, ou seja, ou seja, um acréscimo ou decréscimo de 5% no ciclo de trabalho.

```
#include “SanUSB1.h”
#pragma interrupt interrupcao
void interrupcao(){}
#define motor pin_b7
#define led pin_b0
unsigned int ton,toff,incton,inctoff,guardaton,guardatoff;
int1 flag1, flag2;
void main() {
clock_int_4MHz();
incton=2; inctoff=18; //Período de 20ms - Passo mínimo de tempo = 1ms (5% (1/20) do duty cicle )
guardaton=le_eeprom(10);guardatoff=le_eeprom(11);
if (guardaton>0 && guardaton<=20) {incton=guardaton; inctoff=guardatoff;}

while (1) {
ton=incton; toff=inctoff;
```

```
if (!input(pin_b1)) {flag1=1;}
if (incton<20 && flag1==1 && input(pin_b1) ) {flag1=0;++incton;--inctoff;nivel_alto(led);
//se não chegou no máximo (incton<50),
write_eeprom(10,incton);write_eeprom(11,inctoff);
//se o botão foi pressionado (flag1==1) e se o botão já foi solto (input(pin_b1)) incremente
}// a onda quadrada e guarde os valores na eeprom

    if (!input(pin_b2)) {flag2=1;}
    if (inctoff<20 && flag2==1 && input(pin_b2) ) {flag2=0;++inctoff;--incton;nivel_baixo(led);
    write_eeprom(10,incton);write_eeprom(11,inctoff);
    }
        nivel_alto(motor);
        while(ton) {--ton;tempo_ms(1); } //Parte alta da onda quadrada
        nivel_baixo(motor);
        while(toff) {--toff;tempo_ms(1); } //Parte baixa da onda quadrada
}}
```

A Figura 9.2 mostra aplicação de Mosfet para implementação em PWM de motor DC.

Figura 9. 2: PWM com Mosfet.



Fonte: Elaborado pelo autor.

Capítulo 10

Interrupções e temporizadores

10.1 Interrupções

As interrupções são causadas através de eventos assíncronos (podem ocorrer a qualquer momento) causando um desvio no processamento. Este desvio tem como destino um endereço para tratamento da interrupção. Uma boa analogia para melhor entendermos o conceito de interrupção é a seguinte: você está trabalhando digitando uma carta no computador quando o seu telefone toca. Neste momento você, interrompe o que está fazendo, para atender ao telefone e verificar o que a pessoa do outro lado da linha está precisando. Terminada a conversa, você coloca o telefone no gancho novamente e retoma o seu trabalho do ponto onde havia parado. Observe que não precisamos verificar a todo instante, se existe ou não alguém na linha, pois somente quando o ramal é chamado, o telefone toca avisando que existe alguém querendo falar com você.

Após do atendimento das interrupções, o microcontrolador retorna exatamente ao ponto onde parou no programa antes de atendê-la. As interrupções mais comuns na família PIC18F são:

- pela interrupção externa 0 (Pino B0) ->*enable_interrupts(int_ext);*
- pela interrupção externa 1 (Pino B1) ->*enable_interrupts(int_ext1);*
- pela interrupção externa 2 (Pino B2) ->*enable_interrupts(int_ext2);*
- pelo contador/temporizador 0 ->*enable_interrupts(int_timer0);*
- pelo contador/temporizador 1 ->*enable_interrupts(int_timer1);*
- pelo contador/temporizador 2 ->*enable_interrupts(int_timer2);*
- pelo canal de comunicação serial ->*enable_interrupts(int_rda); //serial*

As interrupções do PIC são *vetorizadas*, ou seja, *têm endereços de início da interrupção fixos para a rotina de tratamento.* No PIC18F2550 o endereço de tratamento é 0x08. No programa em C basta escrever a função de tratamento da interrupção após #, e o compilador fará o direcionamento do códico automaticamente para essa posição.

10.2 Interrupções externas

O modelo PIC18F2550 possui três interrupções externas, habilitadas nos pinos B0 (ext) , B1 (ext1) e B2 (ext2), que atuam (modo *default)* quando os pinos são aterrados. Quandos atuados o processamento é desviado para **#int_ext**, **#int_ext1** ou **#int_ext2**, respectivamente, para que a interrupção possa ser tratada por uma função específica, que no caso do exemplo é *void bot_ext()*.

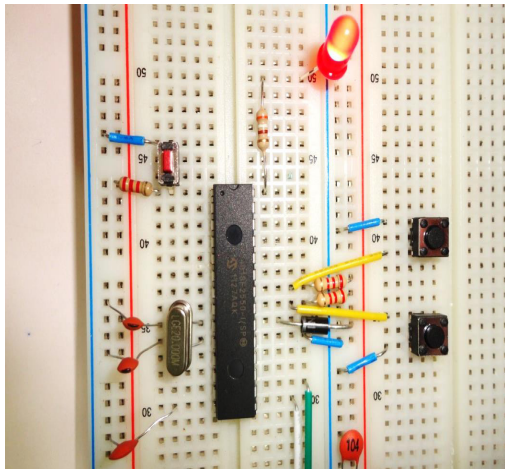
Dentro da função principal deve-se habilitar o "disjuntor" geral das interrupções, *enable_interrupts(global);* e depois a interrupção específica, por exemplo *enable_interrupt-*

s(int_ext); como mostra o exemplo com aplicação de interrupção externa e também interrupção do temporizador 1.

```
#include “SanUSB1.h”

#pragma interrupt interrupcao
void interrupcao(){
if (timer0_interrompeu)          {          //espera o estouro do timer0
timer0_interrompeu=0;              //limpa a flag de interrupÁ„o
PORTBbits.RB7 =! PORTBbits.RB7;    //Pisca o LED EM B7
TMR0H=0X0B ; TMR0L=0xDC  ; }//Carrega 3036 = 0x0BDC (65536-3036 -> conta 62500us x
8 = 0,5seg)

if (ext1_interrompeu)     {        //espera a interrupÁ„o externa 1 (em B1)
ext1_interrompeu = 0;                //limpa a flag de interrupÁ„o
PORTBbits.RB6 =! PORTBbits.RB6; }       //altera o LED em B0
while(PORTBbits.RB1==0){};
tempo_ms(100);           //Delay para mascarar o ruido do bot„o(Bouncing)
}
void main()        {
clock_int_4MHz();
TRISB = 0b00111111;          //B6 e B7 como Saìda
habilita_interrupcao(timer0);    // habilita a interrupÁ„o do TMR0
multiplica_timer16bits(0,8);     //setup_timer0 - 16 bits com prescaler 1:8 - multiplica o tempo
por 8
habilita_interrupcao(ext1);          // habilita a interrupÁ„o externa 1 (pino B1)
INTCON2bits.INTEDG1 = 0;     // a interrupÁ„o externa 1 ocorre somente na borda de descida
while (1){
if(!entrada_pin_e3){Reset();}//pressionar o bot„o para gravaÁ„o
tempo_ms(100);}
          }
```

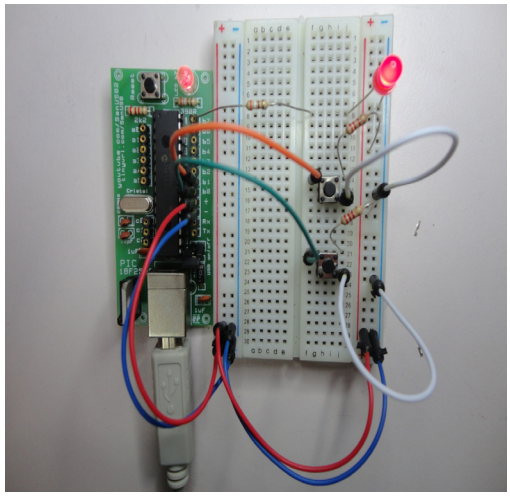
A prática com botões pode ser vista nas Figuras 10.1 e 10.2.

Figura 10. 1: Prática botões em protoboard.



Fonte: Elaborado pelo autor.

Figura 10. 2: Prática com botões em protoboard com placa SanUSB.



Fonte: elaboração do autor

Para habilitar a nomenclatura das as interrupções estão disponíveis em *view* > *valid interrupts*. Quando for utilizada alguma interrupção externa, é necessário inserir um resistor de *pull-up* externo de 1K a 10K para elevar o nivel lógico do pino quando o mesmo for liberado evitando outra interrupção, pois o processador entende *tristate* e níveis intermediários de tensão como nivel lógico baixo.

Segue código exemplo utilizando o comando IF INPUT ao invés da interrupção externa:

```
        #include “SanUSB1.h”//BIBLIOTECA DE INSTRUÇÕES

#pragma interrupt interrupcao
void interrupcao(){}


        main()//PROGRAMA PRINCIPAL

        {

        while (1)//LAÇO INFINITO

        {

        if (input(pin_b0)==0){// SE BOTÃO NO PINO B0 FOR ATERRADO

        nivel_alto(pin_b7);}// LIGA LED NO PINO B7

        if (input(pin_b1)==0){// SE BOTÃO NO PINO B1 FOR ATERRADO

        nivel_baixo(pin_b7); }// APAGA LED NO PINO B7

        }}
```

Faça um exemplo em que ao pressionar o botão do pino b0, o LED do pino b7 pisque a cada 500 ms e ao pressionar o botão do pino b1, o mesmo LED pisque a cada 100 ms.

Sugestão de código:

```
        #include “SanUSB1.h”//BIBLIOTECA DE INSTRUÇÕES

#pragma interrupt interrupcao
void interrupcao(){}

void main(void){//PROGRAMA PRINCIPAL
clock_int_4MHz();
while (1){//LAÇO INFINITO
    if (input(pin_b0)==0){// SE BOTÃO NO PINO B0 FOR ATERRADO
    while(input(pin_b1)==1){//ENQUANTO O OUTRO NÃO FOR ATERRADO...
    inverte_saida(pin_b7);// LIGA LED NO PINO B7
    tempo_ms(500);}}
    if (input(pin_b1)==0){// SE BOTÃO NO PINO B1 FOR ATERRADO
    while(input(pin_b0)==1){//ENQUANTO O OUTRO NÃO FOR ATERRADO...
    inverte_saida(pin_b7);
    tempo_ms(50);
    }    }}}
```

É possível também com 3 LEDs (pinos b7, b6 e b5) piscando, de acordo com o programa abaixo:

```
#include <SanUSB.h>//BIBLIOTECA DE INSTRUÇÕES
#pragma interrupt interrupcao
void interrupcao(){}
        main(){//PROGRAMA PRINCIPAL
        clock_int_4MHz();
        while (1)//LAÇO INFINITO
        {
        if (input(pin_b0)==0){// SE BOTÃO NO PINO B0 FOR ATERRADO
        while(input(pin_b1)==1){
        nivel_alto(pin_b7);// LIGA LED NO PINO B7
        tempo_ms(500);
        nivel_baixo(pin_b7);// LIGA LED NO PINO B7
        tempo_ms(500);
        nivel_alto(pin_b6);// LIGA LED NO PINO B6
        tempo_ms(500);
        nivel_baixo(pin_b6);// LIGA LED NO PINO B6
        tempo_ms(500);
        nivel_alto(pin_b5);// LIGA LED NO PINO B5
        tempo_ms(500);
        nivel_baixo(pin_b5);// LIGA LED NO PINO B5
        tempo_ms(500);}}
        if (input(pin_b1)==0){// SE BOTÃO NO PINO B1 FOR ATERRADO
        while(input(pin_b0)==1){
        nivel_alto(pin_b7);// LIGA LED NO PINO B7
        tempo_ms(50);
        nivel_baixo(pin_b7);// LIGA LED NO PINO B7
        tempo_ms(50);
        nivel_alto(pin_b6);// LIGA LED NO PINO B6
        tempo_ms(50);
        nivel_baixo(pin_b6);// LIGA LED NO PINO B6
        tempo_ms(50);
        nivel_alto(pin_b5);// LIGA LED NO PINO B5
        tempo_ms(50);
        nivel_baixo(pin_b5);// LIGA LED NO PINO B5
        tempo_ms(50);
        } }     }}
```

10.3 Interrupção dos temporizadores

O microcontrolador PIC 18F2550 tem quatro temporizadores, que são os timers 0, 1, 2 e 3. O *timer* 0 pode ser configurado como 8 ou 16 bits, ou seja, pode contar até 65535µs (2^{16}) e um *prescaler* (divisor de frequência ou multiplicador de tempo) de até 256 (RTCC_DIV_256). Os *timers* 1 e 3 são idênticos de 16 bits e um prescaler de até 8 (RTCC_DIV_8). Por sua vez, O *timer* 2 possui 8 bits e um prescaler de até 16 (RTCC_DIV_16). Como o timer 2 é utilizado para outros periféricos internos como PWM, os timers 0, 1 e 3 são mais utilizados para temporização e contagem.

Os timers incrementam até estourar, quando estouram, caso a interrupção esteja habilitada, o processamento é desviado para **void interrupcao(){}**, para que a interrupção possa ser tratada por uma função específica, indicada pela *flag* do timer que interrompeu.

O firmware a seguir pisca um led em b5 na função principal main(), outro pela interrupção do timer 1 em b6 e um terceiro led em b7 pela interrupção do timer0.

```
#include "SanUSB1.h"
// inserir o desvio _asm goto interrupcao _endasm  na funÁ„o void _high_ISR (void){   } em SanUSB.h
#pragma interrupt interrupcao
void interrupcao()
{
if (timer1_interrompeu)    {         //espera o estouro do timer0
timer1_interrompeu=0;                  //limpa a flag de interrupÁ„o
PORTBbits.RB7 =! PORTBbits.RB7;    //Pisca o LED em B7
tempo_timer16bits(1,50000); }        //Conta 8 x 50000us = 0,4 seg.
if (timer0_interrompeu)    {         //espera o estouro do timer0
timer0_interrompeu = 0;                //limpa a flag de interrupÁ„o
PORTBbits.RB7 =! PORTBbits.RB7;    //Pisca o LED em B7
tempo_timer16bits(0,62500); }

if (ext1_interrompeu)     {        //espera a interrupÁ„o externa 1 (em B1)
ext1_interrompeu = 0;                //limpa a flag de interrupÁ„o
PORTBbits.RB7 =! PORTBbits.RB7;  //altera o LED em B0
tempo_ms(100);               }       //Delay para mascarar o ruido do bot„o(Bouncing)
}
void main()      {
clock_int_4MHz();
TRISB = 0b01111110;          //B0 e B7 como SaÌda

habilita_interrupcao(timer0);
multiplica_timer16bits(0,16);          //liga timer0 - 16 bits com multiplicador (prescaler) 8
tempo_timer16bits(0,62500);       //Conta 16 x 62500us = 1 seg.

//habilita_interrupcao(timer1);
multiplica_timer16bits(1,8);          //liga timer3 - 16 bits com multiplicador (prescaler) 8
tempo_timer16bits(1,50000);                    //Conta 8 x 50000us = 0,4 seg.

//habilita_interrupcao(ext1);     // habilita a interrupÁ„o externa 1 com borda de descida

while(1){if(!entrada_pin_e3) {Reset();
          tempo_ms(100);}}
}
```

É possível também gerar um tempo utilizando o timer 0 na configuração de 8 bits sem interrupção como é mostrado no firmware abaixo:

```
#include "SanUSB1.h" // Prática de interrupção do temporizador 0

unsigned int i;

#pragma interrupt interrupcao
void interrupcao(){}

void timer0_ms (unsigned int cx)
      {
      unsigned int i;
      TMR0L = 0;
      T0CON =0B11000001; //TMR0ON, 8 bits, Prescaler 1:4 (001 - see datasheet)
                  //T0CON BITS = TMR0ON , T08BIT(0=16bits OR 1=8bits), T0CS , T0SE , PSA , T0PS2
T0PS1 T0PS0.
                  //Defaull 1 in all bits.
      for (i = 0; i < cx; i++) {
       TMR0L = TMR0L + 6; // load time before plus 250us x 4 (prescaler 001) = 1000us = 1ms into
TMR0 so that it rolls //over (for 4MHz oscilator clock)
        INTCONbits.TMR0IF = 0;
        while(!INTCONbits.TMR0IF); // wait until TMR0 rolls over
                        }
       }
void main() {
clock_int_4MHz();
 //T0CON BITS = TMR0ON , T08BIT(0=16bits OR 1=8bits), T0CS , T0SE , PSA , T0PS2 T0PS1 T0PS0.
 //Defaull 1 in all bits.
T0CON =0B11000001; //TMR0ON, 8 bits, Prescaler 1:4 (001 - see datasheet)
TMR0L = 6; //conta 250us antes de estourar x 4 = 1ms

while (1){
   if(!entrada_pin_e3){Reset();}//pressionar o botão para gravação

inverte_saida(pin_b7);
   timer0_ms(500);
   }
}
```

Este mesmo princípio utilizando interrupção está descrito abaixo:

```
#include "SanUSB1.h" // Prática de interrupção do temporizador 0

 unsigned int i;

#pragma interrupt interrupcao
void interrupcao()
   {
   if (INTCONbits.TMR0IF)     {  //espera o estouro do timer0 -> TMR0L=0
   INTCONbits.TMR0IF = 0;        //limpa a flag de interrupção
++i; if(i>=500){i=0;
                inverte_saida(pin_b7);}
   TMR0L = TMR0L + 6; // load time before plus 250us x 4 (prescaler 001) = 1000us = 1ms into TMR0 so
that it rolls over //(for 4MHz oscilator clock)
                          }
   }

void main() {
clock_int_4MHz();
 //T0CON BITS = TMR0ON , T08BIT(0=16bits OR 1=8bits), T0CS , T0SE , PSA , T0PS2 T0PS1 T0PS0.
 //Default 1 in all bits.
T0CON =0B11000001; //TMR0ON, 8 bits, Prescaler 1:4 (001 - see datasheet)
TMR0L = 6; //conta 250us antes de estourar x 4 = 1ms

RCONbits.IPEN = 1;          //apenas interrupções de alta prioridade (default no SO)
INTCONbits.GIEH = 1;       //Habilita interrupções de alta prioridade
INTCONbits.TMR0IE = 1;   //Habilita interupção timer 0

while (1){
   if(!entrada_pin_e3){Reset();}//pressionar o botão para gravação

//  inverte_saida(pin_b7);
  //  timer1_ms(500);
   }}
```

10.4 Multiplexação de displays por interrupção de temporizadores

O programa abaixo mostra uma multiplexação de displays de 7 segmentos por interrupção dos temporizadores 0 e 1. O timer 0 incrmena a variável a ser multiplexada pelos displays e o timer 1 multiplexa a porta B dígitos de dezenas e dígitos de unidades até 99.

```
#include "SanUSB1.h"
#define DIGIT1 PORTCbits.RC0
#define DIGIT2 PORTCbits.RC1
unsignedchar Cnt = 0;
unsignedchar Flag = 0;
unsignedchar Display(unsignedchar);
#pragma interrupt interrupcao
void interrupcao()
{
unsignedchar Msd, Lsd;
TMR0L = 100; // Recarrega TMR0 para contar 156 x 32 us
INTCON = 0x20; // Set T0IE and clear T0IF
Flag = ~ Flag; // Inverte a Flag
if(Flag == 0) // Do digit 1
{
        DIGIT2 = 0;
        Msd = Cnt / 10; // MSD digit
//*/if(Msd != 0) // blank MSD
//{
        PORTB = Display(Msd); // Send to PORT C
DIGIT1 = 1; // Enable digit 1
//}
    }
else
{ // Faz o digito 2
DIGIT1 = 0; // Disable digit 1
        Lsd = Cnt % 10; // LSD digit
        PORTB = Display(Lsd); // Send to PORT C
        DIGIT2 = 1; // Enable digit
}
}
unsignedchar Display(unsignedchar i)
{
unsignedchar Pattern;
unsignedchar SEGMENT[ ] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07, 0x7F,0x6F};
Pattern = SEGMENT[i]; // Pattern to return
return (Pattern);
}
void main()    {
clock_int_4MHz();
//tempo_us(100);
TRISB = 0; // PORT B are outputs
TRISC = 0; // RC0, RC1 are outputs
DIGIT1 = 0;
// Configura TMR0 timer interrupt
T0CON = 0xC4; // Prescaler = 32
TMR0L = 100; // Load TMR0L with 156 x 32 us
INTCON = 0xA0; // Enable TMR0 interrupt
tempo_ms(1000);
while(1)
{
Cnt++; // Incrementa Cnt
if(Cnt == 100) Cnt = 0; // Conta de 0 a 99
tempo_ms(1000);
}}
```

Capítulo 11

Emulação de portas lógicas

Os operadores lógicos descritos abaixo adotam o padrão ANSI C, ou seja, podem ser utilizados por qualquer compilador em linguagem C direcionado à microcontroladores.

11.1 Instruções lógicas para testes condicionais de números

Nesse caso, os operadores (ver Tabela 11.1) são utilizados para realizar operações de testes condionais geralmente entre números inteiros.

Tabela 11. 1: Operadores lógicos em linguagem C.

OPERADOR	COMANDO
&&	Operação E (AND)
ll	Operação OU (OR)
!	Operação NÃO (NOT)

Fonte: Elaborado pelo autor.

Exemplos:

if (segundodec==05 **&&**(minutodec==00|| minutodec==30)) {flagwrite=1;}//Analisando um relógio para setar a flagwrite

if (x>0 **&&** x<20) (y=x;) // Estabelecendo faixa de valores para y.

11.2 Instruções lógicas Boolanas Bit a Bit

É importante salientar que emulação é a reprodução via software das funções de um determinado sistema real. Através do circuito SanUSB, é possível emular as portas lógicas físicas e as diversas combinações das mesmas com apenas uma função booleana no programa. A Tabela 11.2 mostra algumas expressões booleanas literais e a representação em linguagem C.

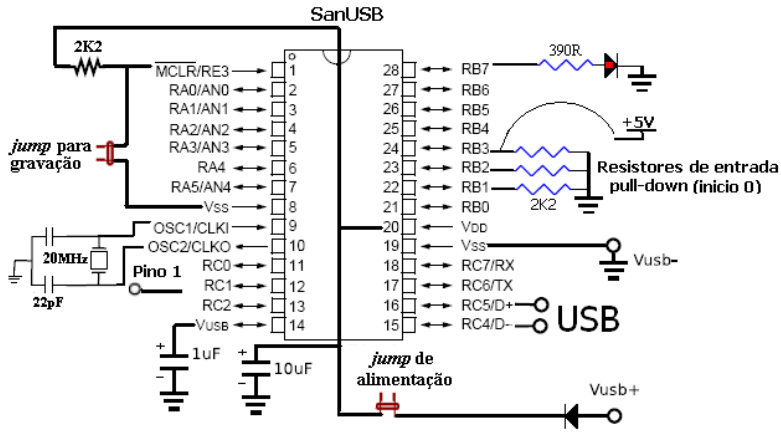
Tabela 11. 2: Expressão booleana literal e em linguagem C.

OPERAÇÃO	EXPRESSÃO BOOLEANA LITERAL	EXPRESSÃO BOOLEANA EM C
Operação E (AND)	S= A . B	S= A & B
Operação OU (OR)	S= A + B	S= A | B
Operação NÃO (NO)	S= $\overline{A}$	S= !A
OU exclusivo (XOR)	S= A $\oplus$ B	S= A ^ B

Fonte: Elaborado pelo autor.

O circuito da Figura 11.1 mostra as possíveis entradas booleanas nos pinos B1, B2 e B3 e a saída do circuito lógico booleano é expressa de forma real através do LED no pino B7.

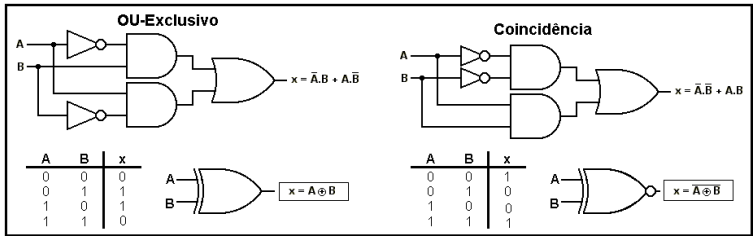
Figura 11. 1: Uso de resistores de *pull down* para aplicar função lógica 0/1.



Fonte: Elaborado pelo autor.

É importante salientar que através das operações básicas E, OU, NÃO e OU-Exclusivo é possível construir outras operações como NAND, NOR e Coincidência, que é o inverso do OU-Exclusivo (ver Figura 11.2). Isto é realizado transformando a expressão booleana literal em uma expressão booleana em C e apresentando o resultado em um LED de saída.

Figura 11. 2: Outras funções lógicas a partir de NOT, OR e AND.



Fonte: Elaborado pelo autor.

Exemplo 1: Elabore um programa para emular uma porta lógica OU-Exclusivo através do microcontrolador.

```
#include “SanUSB1.h” // Emulação de circuitos lógicos booleanos (OU-Exclusivo)
short int A, B, saida, ledpisca;

void main(){
clock_int_4MHz();

while (TRUE)
{
A=input(pin_b1); //entrada com pull-down externo (resistor conectado ao Terra)
B=input(pin_b2); //entrada com pull-down externo (resistor conectado ao Terra)

saida = A^B; //saida é igual ao resultado do OU-Exclusivo obtida pelas entradas dos pinos A e B
output_bit(pin_b7,saida); //O pino_b7 mostra o resultado do circuito lógico booleano alocado em saida

ledpisca=!ledpisca; // ledpisca é igual ao inverso de ledpisca
output_bit(pin_b0,ledpisca); // b0 recebe o valor de ledpisca
tempo_ms(500);
}}
```

Exemplo 2: Elabore um programa e a tabela verdade para emular o circuito lógico booleano da expressão lógica **S=!(!A & B) & C**.

O programa para emular de forma real esse circuito é mostrado abaixo:

```
#include “SanUSB1.h” // Emulação de circuitos lógicos booleanos
short int A, B, C, saida, ledpisca;
void main(){
clock_int_4MHz();
while (TRUE){
A=input(pin_b1); //entrada com pull-down externo (resistor conectado ao Terra)
B=input(pin_b2); //entrada com pull-down externo (resistor conectado ao Terra)
C=input(pin_b3); //entrada com pull-down externo (resistor conectado ao Terra)

saida = !(!A & B) & !C; //saida do circuito booleano obtida pelas entradas de b1, b2 e b3
output_bit(pin_b7,saida); //O pino_b7 mostra o resultado do circuito lógico booleano

ledpisca=!ledpisca; // ledpisca é igual ao inverso de ledpisca
output_bit(pin_b0,ledpisca); // b0 recebe o valor de ledpisca
tempo_ms(500);
}}
```

Note que para emular qualquer outro circuito booleano com três entradas, basta modificar apenas a função de conversão em negrito (saida = !(!A & B) & !C). A tabela verdade desse circuito booleano é mostrada na Tabela 11.3.

Tabela 11. 3: Tabela verdade do exemplo 2.

ENTRADAS			**SAIDA**
A	**B**	**C**	**S**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Fonte: Elaborado pelo autor.

Exemplo 3: Elabore um programa e a tabela verdade para emular o circuito lógico booleano da expressão lógica **S= (A & B) | !C | !(C & D).**

```
#include “SanUSB1.h” // Emulação de circuitos lógicos booleanos
short int A, B, C, D, saida, ledpisca;
void main(){
clock_int_4MHz();
while (TRUE){
A=input(pin_b1); //entrada com pull-down externo (resistor conectado ao Terra)
B=input(pin_b2); //entrada com pull-down externo (resistor conectado ao Terra)
C=input(pin_b3); //entrada com pull-down externo (resistor conectado ao Terra)
D=input(pin_b3); //entrada com pull-down externo (resistor conectado ao Terra)

saida= (A & B) | !C | !(C & D);
output_bit(pin_b7,saida); //O pino_b7 mostra o resultado do circuito lógico booleano
ledpisca=!ledpisca; // ledpisca é igual ao inverso de ledpisca
output_bit(pin_b0,ledpisca); // b0 recebe o valor de ledpisca
tempo_ms(500);}}
```

A tabela verdade deste circuito lógico é mostrada na Tabela 11.4.

Tabela 11. 4: Tabela verdade do exemplo 3.

ENTRADAS				**SAIDA**
A	**B**	**C**	**D**	**S**
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Fonte: Elaborado pelo autor.

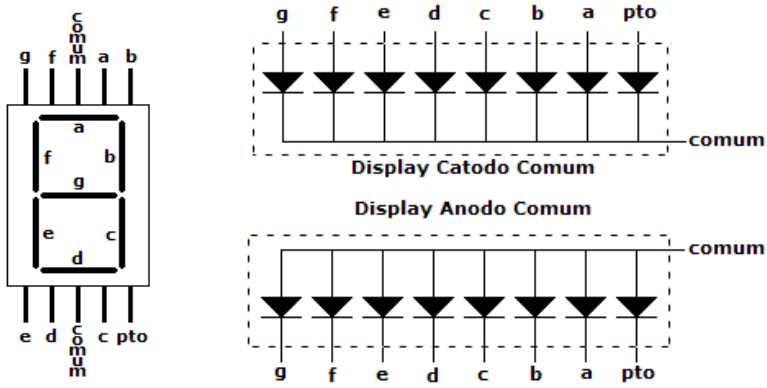
A tabela verdade pode ser facilmente comprovada de forma real montando o circuito proposto.

Capítulo 12
Emulação de decodificador para display de 7 segmentos

Antes de comentar sobre os decodificadores, vamos definir o que é um display de sete segmentos. O display de sete segmentos, é formado por sete leds. Quando necessita-se acender o número "0", liga-se os leds correspondentes ao digito "0", por exemplo, os segmentos a, b, c, d, e, f. Na Figura12.1 é mostrado um display de sete-segmentos e a respectivos pinos. No lado direito, os dois tipos de displays, anodo comum e catodo comum. Não esqueça que no anodo comum o led liga com Gnd e no catodo comum o led liga com Vcc.

Como os segmentos são leds, então é necessário limitar a corrente, para isso devemos usar um resistor em cada segmento (catodo comum), ou apenas um no comum (anodo comum), senão serão queimados. Normalmente se utilizam resistores entre 220 e 560 ohms para uma fonte de 5Volts. Uma dica, se for usar um display, teste antes cada segmentos, para ter certeza que não está usando um display com algum segmento queimado.

Figura 12. 1: Display de 7 segmentos e conexão interna.



Fonte: Elaborado pelo autor.

Os decodificadores, inverso dos codificadores, tem a função de converter um código "desconhecido" de linguagem de máquina, como o binário de entrada mostrado neste exemplo, em um código compreensível, como o código decimal ou um desenho em um display de 7 segmentos. Os decodificadores fixos podem ser construídos com portas lógicas reais ou emulados, ou seja, reproduzidos via software, como é o caso proposto.

Os decodificadores de displays de 7 segmentos, como o 9317 (anodo comum) e o 9307 (catodo comum),recebem 4 bits de entrada para a decodificação do número a ser “desenhado” pelos segmentos dos displays (ver Tabela 12.1).

Tabela 12. 1: Sinais CI 9317.

		Entradas				**Saídas**								**Desenho**
/LT	**/RBI**	**A0**	**A1**	**A2**	**A3**	a	b	c	d	e	f	g	ponto	
L	X	X	X	X	X	L	L	L	L	L	L	L	H	teste
H	L	L	L	L	L	H	H	H	H	H	H	H	L	apaga
H	H	L	L	L	L	L	L	L	L	L	L	H	H	0
H	X	H	L	L	L	H	L	L	H	H	H	H	H	1
H	X	L	H	L	L	L	L	H	L	L	H	L	H	2
H	X	H	H	L	L	L	L	L	L	H	H	L	H	3
H	X	L	L	H	L	H	L	L	H	H	L	L	H	4
H	X	H	L	H	L	L	H	L	L	H	L	L	H	5
H	X	L	H	H	L	L	H	L	L	L	L	L	H	6
H	X	H	H	H	L	L	L	L	H	H	H	H	H	7
H	X	L	L	L	H	L	L	L	L	L	L	L	H	8
H	X	H	L	L	H	L	L	L	L	H	L	L	H	9

Fonte: Elaborado pelo autor.

Para a emulação, ou seja, reprodução via software, de decodificadores de displays de sete segmentos, os pinos do microcontrolador devem apresentar os seguintes valores mostrados naTabela 12.2.

Tabela 12. 2: Comandos em binário e hexa para multiplexação.

Display Anodo comum –segmento acende com zero (0)								
NÚMERO DISPLAY	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	**Porta B Hexadecimal**
	g	**f**	**e**	**d**	**c**	**b**	**a**	
0	1	0	0	0	0	0	0	0x40
1	1	1	1	1	0	0	1	0x79
2	0	1	0	0	1	0	0	0x24
3	0	1	1	0	0	0	0	0x30
4	0	0	1	1	0	0	1	0x19
5	0	0	1	0	0	1	0	0x12
6	0	0	0	0	0	1	0	0x02
7	1	1	1	1	0	0	0	0x78
8	0	0	0	0	0	0	0	0x00
9	0	0	1	0	0	0	0	0x10

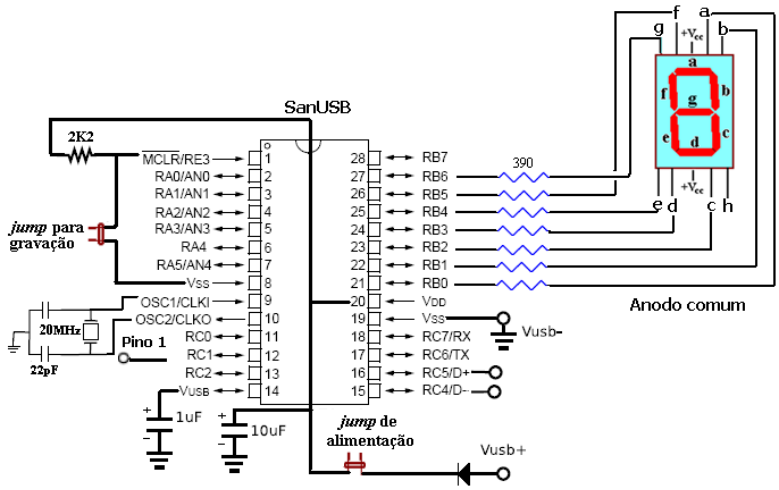
Fonte: Elaborado pelo autor.

Abaixo é mostrado um programa exemplo para contar de 0 a 9 com um display de sete segmentos anodo comum. Dependendo dos display anodo ou catodo comum, como também dos pinos do microcontrolador ligados ao displays, é possível utilizar o mesmo programa abaixo, alterando apenas os valores dos elementos da matriz setseg[10].

```
#include "SanUSB1.h"//Display de 7seg conta de 0 a 9
unsigned char set_seg[10] = {0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10};
int i;
void interrupcao(){}
void main(){
   clock_int_4MHz();
   TRISB = 0b00000000;
   nivel_baixo(pin_c0);
   while(1){
      for(i = 0; i<10; i++){
            PORTB = set_seg[i];
tempo_ms(500);
      }   }}
```

Exemplo: Construa um decodificador emulado atarvés de diagramas de Karnaugh para escrever, letra por letra, a cada segundo, a palavra StoP. É importante salientar que os pinos do microcontrolador e os pinos do display em anodo comum devem ser conectados com um resistor de 220Ω a 1KΩ para não queimar os segmentos do display. O circuito da Fiigura 12.2 mostra a ligação do display de 7 segmentos.

Figura 12. 2: Conexão do display 7 seg na porta B do PIC.



Fonte: Elaborado pelo autor.

Os segmentos acendem com zero no pino do microcontrolador (anodo comum). Note

que como são somente quatro letras só são necessárias duas entradas (Xe Y) para a decodificação (ver Figura 12.3).

Figura 12. 3: Mapa de bits para multiplexar a palavra STOP.

	Entradas		Pin_b0	Pin_b1	Pin_b2	Pin_b3	Pin_b4	Pin_b5	Pin_b6
	X	Y	a	b	c	d	e	f	g
S	0	0	0	1	0	0	1	0	0
t	0	1	1	1	1	0	0	0	0
o	1	0	1	1	0	0	0	1	0
P	1	1	0	0	1	1	0	0	0

Fonte: Elaborado pelo autor.

Após a definição de cada segmento do display para representar os caracteres da palavra, é feita a simplicação de cada segmento através dos diagramas de Karnaugh da Tabela

```
#include "SanUSB1.h" // Emulação de decodificador para display de 7 segmentos - //palavra StoP

#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao(){       }

short int X, Y;  //Entradas
short int a, b, c, d, e, f, g;  //saídas

void decodificador(short int X, short int Y) //Função auxiliar do decodificador fixo para StoP
{
a=X  ^  Y;        saida_pino(pin_b0,a);       //Anodo comum
b=!X | !Y;        saida_pino(pin_b1,b);
c=Y;              saida_pino(pin_b2,c);
d=X & Y;          saida_pino(pin_b3,d);
e=!X & !Y;        saida_pino(pin_b4,e);
f=X & !Y;         saida_pino(pin_b5,f);
g=0 ;             saida_pino(pin_b6,g);
}

void main(){
clock_int_4MHz();
while (1)
{
decodificador(0,0); // Insere as entradas X=0 e Y=0 no decodiicador fixo ? Saída letra S
tempo_ms(1000);

decodificador(0,1); // Saída letra t
tempo_ms(1000);
decodificador(1,0); // Saída letra o
tempo_ms(1000);
decodificador(1,1); // Saída letra P
tempo_ms(1000);
}}
```

Exemplo 2: Construa um decodificador emulado para escrever, letra por letra no mesmo display de 7 segmentos, a cada segundo, a palavra USb2. Como o display é anodo comum (+5V no anodo do display), os segmentos acendem com zero no pino do microcontrolador (ver Figura 12.4).

Figura 12. 4: Mapa de bits para multiplexar a palavra USb2.

	Entradas		Pin_b0	Pin_b1	Pin_b2	Pin_b3	Pin_b4	Pin_b5	Pin_b6
	X	Y	a	b	c	d	e	f	g
U	0	0	1	0	0	0	0	0	1
S	0	1	0	1	0	0	1	0	0
b	1	0	1	1	0	0	0	0	0
2	1	1	0	0	1	0	0	1	0

Fonte: Elaborado pelo autor.

Após a definição de cada segmento do display para representar os caracteres da palavra, é feita a simplicação de cada segmento através dos diagramas de Karnaugh,como exemplificado na Figura 12.5, para a construção da função de emulação do decodificador fixo.

Figura 12. 5: Diagramas de Karnaugh USb2.

a	**/Y**	**Y**	**b**	**/Y**	**Y**	**c**	**/Y**	**Y**	**d**	**/Y**
/X	1	0	**/X**	0	1	**/X**	0	0	/X	0
X	1	0	**X**	1	0	**X**	0	1	X	0
e	**/Y**	**Y**	**f**	**/Y**	**Y**	**g**	**/Y**	**Y**		
/X	0	1	**/X**	0	0	**/X**	1	0		
X	0	0	**X**	0	1	**X**	0	0		

Fonte: Elaborado pelo autor.

O programa abaixo mostra as funções de emulação do decodificador fixo para a palavra USb2 obtidas dos diagramas de Karnaugh.

```
#include "SanUSB1.h" // Emulação de decodificador para display de 7 segmentos - palavra
//Usb2

#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao(){
      }
short int X, Y;  //Entradas
short int a, b, c, d, e, f, g;  //saídas

void decodificador(short int X, short int Y) //Função auxiliar do decodificador fixo para USb2
{
a=!Y;          saida_pino(pin_b0,a);   //Anodo comum
b=X^Y;         saida_pino(pin_b1,b);
c=X&Y;         saida_pino(pin_b2,c);
d=0;           saida_pino(pin_b3,d);
e=!X&Y;        saida_pino(pin_b4,e);
f=X&Y;         saida_pino(pin_b5,f);
g=!X&!Y;       saida_pino(pin_b6,g);
}
void main(){
clock_int_4MHz();
while (1)
{
decodificador(0,0); // Insere as entradas X=0 e Y=0 no decodiicador fixo ? Saída letra S
tempo_ms(1000);
decodificador(0,1); // Saída letra t
tempo_ms(1000);
decodificador(1,0); // Saída letra o
tempo_ms(1000);
decodificador(1,1); // Saída letra P
tempo_ms(1000);
}}
```

Capítulo 13
Multiplexação com displays de 7 segmentos

Como a economia de consumo e de componentes são sempre fatores importantes a serem considerados em projetos de sistemas digitais, uma técnica bastante utilizada é a multiplexação de displays. Esta técnica permite que um só decodificador de displays como o 9317 (anodo comum), o 9307 (catodo comum) ou apenas sete pinos de um microcontrolador, que emulam as saídas do decodificador, possam controlar uma série de displays em paralelo.

Estes são ciclicamente acesos e apagados numa frequência acima de 20Hz de tal forma que, para o olho humano, todos os displays estejam acesos permanentemente. Para isso, são colocados transistores de corte que atuam em sequência sobre cada terminal comum dos displays em paralelo.

O programa abaixo mostra um exemplo para contar de 0 a 99 multiplexando dois displays de sete segmentos anodo comum.

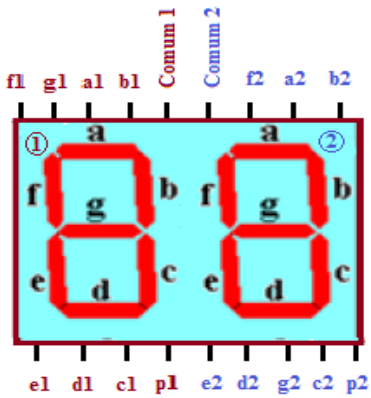
```
#include "SanUSB1.h"

#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao(){             }
#byte port_b = 0xf81;//REGISTRO PORTA B

int setseg[] ={0x40, 0x79, 0x24, 0x30, 0x19, 0x12, 0x02, 0x78, 0x00, 0x10};//numeros de 0 a 9
em hexadecimal para display ANODO COMUM

int i,z,dezena,unidade;

void main(){

clock_int_4MHz();
PORTB= 0b00000000;// Define os pinos da porta B como saìda
TRISB=0x00;   // porta B como saìda.

while (1){
for(i=0;i<99;i++)//CONTA DE 00 A 99
{
for(z=0;z<15;z++){
//REPETE CADA N√MERO POR 15 VEZES POIS O DELAY … CURTO E N√O DARIA TEMPO
VER!
dezena=i/10;//QUEBRA A VARIAVEL i EM 2 PARTES, PRIMEIRO EM DEZENA
//SE O N√MERO FOR 27, DEZENA=2 E UNIDADE=7
unidade=i%10;//DEPOIS EM UNIDADE
nivel_alto(pin_a0);//SELECIONA 1. DISPLAY
nivel_baixo(pin_a1);
```

```
PORTB=setseg[dezena];//MOSTRA O VALOR DE DEZENA
tempo_ms(5);
nivel_alto(pin_a1);//SELECIONA 2. DISPLAY
nivel_baixo(pin_a0);
PORTB=setseg[unidade];//MOSTRA O VALOR DE UNIDADE
tempo_ms(5);
}}
}
}
```

Para iniciar, faça *jumps* entre os pinos abaixo observando a Figura 13.1.

a1 e a2 --- b1 e b2 --- c1 e c2 --- d1 e d2 --- e1 e e2 --- f1 e f2 --- g1 e g2

Figura 13. 1: Pinos do display de 7 segmentos duplo.



Fonte: Elaborado pelo autor.

Complete a ligação do display 7 segmentos ao PIC conectando da seguinte forma (ver Tabela 13.1 e Figuras 13.2 e 13.3).

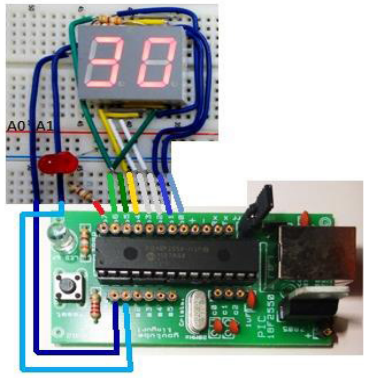
Tabela 13. 1: Conexão dos segmentos ao port B do PIC.

B0	B1	B2	B3	B4	B5	B6	B7
Segmento a	Segmento b	Segmento c	Segmento d	Segmento e	Segmento f	Segmento g	Segmento ponto

Fonte: Elaborado pelo autor.

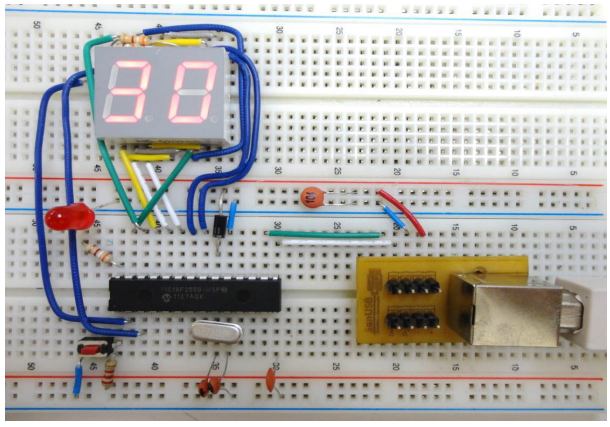
Insira um resistor de 100 R a 1000 R em cada pino comum e conecte: Comum 1 no pino **a0** e Comum 2 no pino **a1** de acordo com a programação.

Figura 13. 2: Esquemático de ligação display 7 segmentos duplo multiplexado.



Fonte: Elaborado pelo autor.

Figura 13. 3: Prática Multiplexação display 7 segmentos, montada em protoboard.



Fonte: Elaborado pelo autor.

Capítulo 14
Comunicação Serial EIA/RS-232

A comunicação serial teve início com a invenção do telégrafo. Depois teve um grande desenvolvimento com a invenção do Teletype (teletipo) pelo Francês Jean Maurice Émile Baudot, em 1871, daí o nome *Baud Rate*. Baudot, além de criar toda a mecânica e elétrica do Teletype, criou também uma tabela de códigos (Código de Baudot) com letras, números, e símbolos para a transferência serial assíncrona digital de informações. Daí surgiu o Padrão de comunicação RS-232, que significa *Padrão Recomendado* versão 232.

Na transmissão dos caracteres através da linha telegráfica, o sinal de Marca era representado pela presença de corrente elétrica, e o Espaço pela ausência desta corrente. Para que o Teletype conseguisse distinguir o início e o final de um caractere, o mesmo era precedido com um sinal Espaço (*start bit*) e finalizado com um sinal de Marca (*stop bit*). Entenda que o estado da linha ociosa (sem transmissão de dados) era o sinal de Marca (presença de corrente elétrica). Foi baseado nesse sistema que o padrão de transmissão RS-232 evoluiu e se tornou um padrão muito utilizado nos computadores e equipamentos digitais.

Algumas interfaces EIA/RS-232 nos computadores atuais fornecem aproximadamente -10v e +10v, mas suportam mínimas de -25v e máximas de +25v.

A Comunicação serial é feita pela transmissão de bits em seqüência. É um modo de comunicação muito recomendado para transmissão de dados a longa distância. Nesse caso, a comunicação serial apresenta um menor custo pelo número reduzido de fios e conseqüentemente menor velocidade em relação à comunicação paralela.

Para a transmissão de dados por distâncias maiores e com pouca interferência pode-se utilizar uma interface com outros padrões como o EIA/RS-232 e o EIA/RS-485. A comunicação serial pode ser síncrona ou assíncrona. Na primeira, além dos bits de dados são enviados também bits de sincronismo, ou seja, o receptor fica em constante sincronismo com o Transmissor. Na comunicação assíncrona, que é o modo mais utilizado de comunicação entre sistemas de controle e automação por não necessitar de sincronismo, existe um bit que indica o início da transmissão, chamado de *start bit (nivel lógico baixo)* e um bit que indica o final da transmissão chamado de *stop bit (nivel lógico alto)*. Nessa transmissão, o Receptor em sincronismo com o Transmissor apenas no início da transmissão de dados. Deve-se considerar que o transmissor e o receptor devem estar na mesma velocidade de transmissão.

Quando o canal serial está em repouso, o sinal correspondente no canal tem um nivel lógico ‘1’. Um pacote de dados sempre começa com um nivel lógico ‘0’ (start bit) para sinalizar ao receptor que um transmissão foi iniciada. O “start bit” inicializa um temporizador interno no receptor avisando que a transmissão. Seguido do start bit, 8 bits de dados de mensagem são enviados com a velocidade de transmissão pré-programada no emissor e no receptor. O pacote é concluído com os bits de paridade e de parada (“stop bit”).

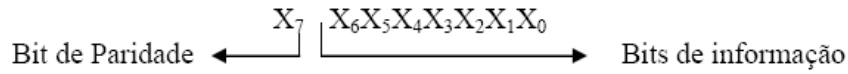
O bit de paridade é usado como nono bit com o propósito de detecção de erro. Nessa convenção, quando o número total de dígitos ‘1’ , o valor do bit de paridade é 1 e quando for ímpar é 0.

A interrupção do canal serial é utilizada quando se espera receber um dado em tempo aleatório enquanto se executa outro programa. Quando o dado chega, o start bit (nivel lógico baixo) aciona a interrupção, previamente habilitada, onde a recepção da comunicação serial é executada. Caso o canal serial seja utilizado somente para *transmissão* de dados, não é necessário habilitar a interrupção serial.

14.1 Código ASCII

Um dos formatos mais utilizados em comunicação serial, como no padrão EIA/RS-232, é o ASCII (American Standard Code for Information Interchange). Este formato utiliza sete bits de cada byte (valor máximo 0x7F) e o oitavo bit de paridade que pode ou não ser utilizado, conforme mostrado na Figura 14.1.

Figura 14. 1: Dado em comunicação serial.



Fonte: Elaborado pelo autor.

Se o número de “1s” for par, o bit de paridade X7 é zero e, se for ímpar, X7 é um verificando-se na tabela ASCII.

Capítulo 15
Interface USART do microcontrolador

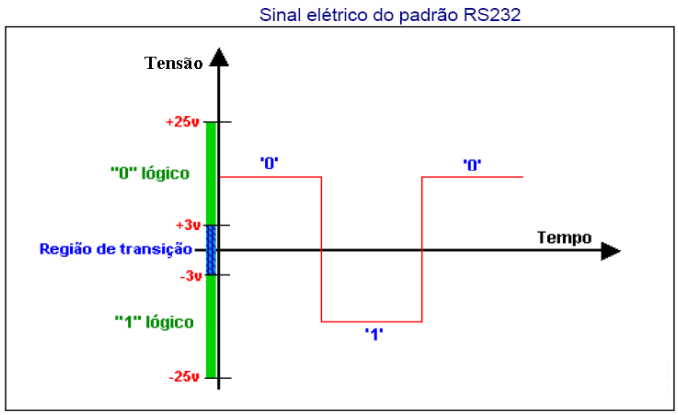
A interface serial USART (transmissor-receptor universal síncrono e assíncrono) dos microcontroladores pode ser síncrona ou assíncrona, sendo esta última a mais utilizada para comunicação com o mundo externo utilizando o padrão EIA/RS-232, onde cada byte serial é precedido por um start-bit de nivel lógico baixo e encerrado por um stop-bit de nivel lógico alto. Os conectores utilizados são o DB9 e o DB25.

Em suma, os pinos utilizados na comunicação serial entre computadores e microcontroladores são o TXD, o RXD e o Terra (GND).

O nivel lógico alto no padrão RS232 está entre **–3 e –25V** e o nivel lógico baixo está entre **+3 e +25V**. Para a comunicação entre um PC e um PIC são utilizados chips que convertem os níveis de tensão TTL/RS232 (ver Figura 15.1).

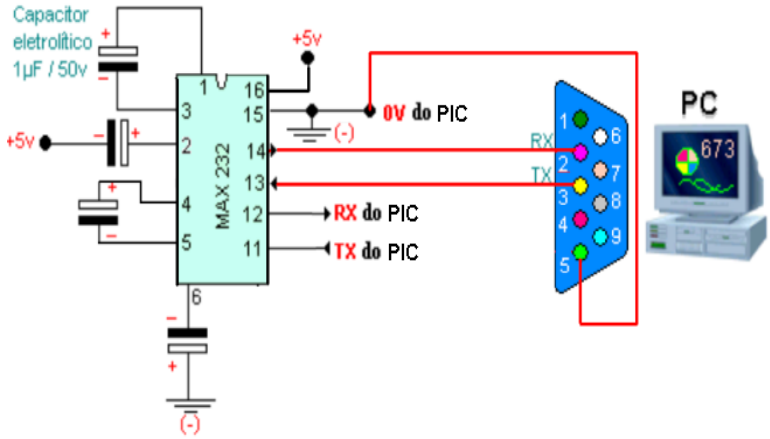
Par converter os padrões TTL/RS232, o chip mais utilizado é o MAX232, o qual utiliza quatro inversores para converter entre –10V (RS232) em +5V (TTL), e entre +10V (RS232) em 0V (TTL). Computadores apresentam cerca de –10V e +10V, mas suportam mínimas de -25v e máximas de +25v. Assim Como o MAX232 existem outros conversores, tipo ICL3232, etc. O esquema de ligação do MAX232 é mostrado na Figura 15.2.

Figura 15. 1: Sinal elétrico RS-232.



Fonte: Elaborado pelo autor.

Figura 15. 2: Circuito de conexão MAX 232.



Fonte: Elaborado pelo autor.

Capítulo 16
Acionamento de motores microcontrolados

Os motores mais utilizados com sistemas microcontrolados são os motores CC, motores de passo e servo-motores.

16.1 Acionamento de motores CC de baixa tensão

Os motores CC são abundantes no mercado em função da ampla gama de utilização, consequentemente, existem em várias dimensões, tensões, pesos, características e são fáceis de encontrar em sucatas como video-cassete, brinquedos, impressoras, etc. e geralmente vêm associados a uma caixa de engrenagem para aumento do torque e redução de velocidade (ver Figura 16.1).

Figura 16. 1: Motores CC.

Fonte: Elaborado pelo autor.

16.2 Motores elétricos utilizados em automóveis

Os motores utilizados em automóveis são todos com tensão nominal a 12 volts (ver Figura 16.2), são robustos e normalmente projetados para condições extremas, tais como poeira, calor, variações de tensão e corrente, entre outros. Algumas opções são ideais para aplicação no robô por serem compactos, fortes, alta rotação e leves, além de serem muito fáceis de conseguir em oficinas e empresas do ramo. Os motores mais usados em projetos são de trava-elétrica das portas, bomba do limpador de para-brisa e de gasolina, bomba de combustível, motor do vidro-elétrico, motor da ventoínha, motor do ventilador interno, limpador de para-brisa dianteiro e traseiro, bomba hidráulica do freio ABS.
Figura 16. 2: Motor CC com caixa de redução.

Fonte: Elaborado pelo autor.

Além do acionamento elétrico, os motores CC de baixa potência utilizados em automação e robótica, apresentam geralmente uma caixa de redução, que é um equipamento composto por engrenagens, com o intuito de reduzir a velocidade de rotação do eixo (ou angular) e aumentar o torque do motor. O torque varia em função da força aplicada (F) e do raio de giro (nas engrenagens é a metade do diâmetro primitivo), segundo a equação T = F x r.Sendo: F = força (em Newtons), r = raio de giro (em metros) e T = torque (em N.m).

Já que o motor imprime uma força constante, a variação do torque entre engrenagens ocorre devido ao raio de giro. Na prática em um sistema de engrenagens, comprovada pelas equações abaixo, quanto maior o diâmetro da engrenagem movida (D_2), maior o torque (T_2) proporcional e menor a velocidade de rotação (n_2). Considerando a engrenagem 1 com motora e a engrenagem 2 como movida, tem-se:

Fconst -> $T_1/r_1 = T_2/r_2$ -> $T_1/D_1 = T_2/D_2$

$\boldsymbol{T_2 . D_1 = T_1 . D_2}$

$\mathbf{n_2 . D_2 = n_1 . d_1}$

16.3 Coroa e o parafuso com rosca sem-fim

A coroa e o parafuso com rosca sem-fim compõem um sistema de transmissão muito utilizado principalmente nos casos em que é necessária elevada redução de velocidade ou um elevado aumento de força, como nos redutores de velocidade.

O número de entradas do parafuso tem influência no sistema de transmissão. Se um parafuso com rosca sem-fim tem apenas uma entrada (mais comum) e está acoplado a uma coroa de 60 dentes, em cada volta dada no parafuso a coroa vai girar apenas um dente. Como a coroa tem 60 dentes, será necessário dar 60 voltas no parafuso para que a coroa gire uma volta. Assim, a rpm da coroa é 60 vezes menor que a do parafuso. Se, por exemplo, o parafuso com rosca sem-fim está girando a 1.800 rpm, a coroa girará a 1.800 rpm, divididas por 60, que resultará em 30 rpm.

Suponhamos, agora, que o parafuso com rosca sem-fim tenha duas entradas e a coroa tenha 60 dentes. Assim, a cada volta dada no parafuso com rosca sem-fim, a coroa girará dois dentes. Portanto, será necessário dar 30 voltas no parafuso para que a coroa gire uma volta.

Assim, a rpm da coroa é 30 vezes menor que a rpm do parafuso com rosca sem-fim. Se, por exemplo, o parafuso com rosca sem-fim está girando a 1.800 rpm, a coroa girará a 1.800 divididas por 30, que resultará em 60 rpm. A rpm da coroa pode ser expressa pela equação:

i = Nc . Zc =Np. Zp

Nc=Np. Zp /Zc

onde:

Nc = número de rotações da coroa (rpm)

Zc = número de dentes da coroa

Np = número de rotações do parafuso com rosca sem-fim (rpm)

Zp= número de entradas do parafuso

As possibilidades mais comuns de controle digital de motores CC são descritas nos tópicos a seguir.

16.4 Chaveamento de motores CC com transistores MOSFET

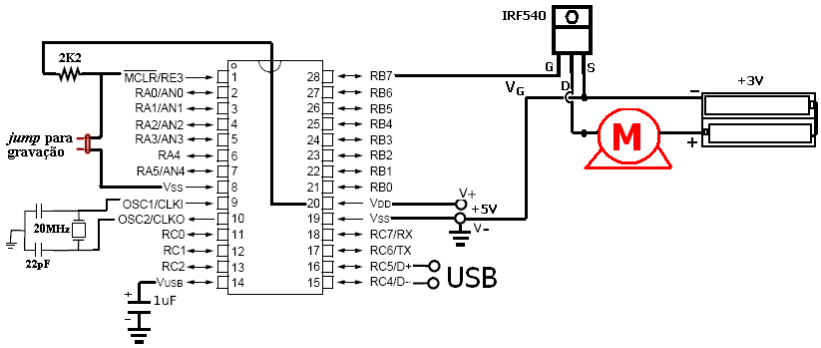
Os transistores de efeito de campo MOSFET executam o chaveamento por tensão na base e podem ser utilizados no lugar dos transistores Darlington para acionamento de dispositivos de média potência, devido principalmente à menor queda de tensão e à menor dissipação em forma de calor.

Os MOSFETs apresentam alta taxa de velocidade no chaveamento e uma resistência interna muito baixa (décimos de ohms). Deesa forma, a queda de tensão nesse transistor é muito baixa, o que não acontece com transistores Darlington. A fgura abaixo mostra a configuração dos pinos de um MOSFET, onde o pino 1 é o *Gate* (base), o pinos 2 é o *Drain* e o pino 3 é o *Source*. Um dos Mosfets mais utilizados é o IRF540.

O modelo IRF540 suporta tensões de chaveamento entre o *drain* e o *source* de 100V e corrente de até 22A. A figura abaixo mostra o circuito com MOSFET para acionamento de quatro motores com MOSFETs. A etapa de potência é composta pelos MOSFETs IRF530 e diodos de roda lvre para proteção de tensão reversa. O funcionamento é simples. Quando o microcontrolador coloca na saída das portas de controle um ‘1’ lógico, os transistores MOSFET entram em condução e uma tensão de 12V é aplicada sobre as cargas. Os resistores e LEDs servem somente para visualização do chaveamento. Note que o pino *Source* do MOSFET é conectado ao Gnd do microcontrolador para gerar a tensão de chaveamento no *gate* (V_G).

A Figura 16.3 mostra o acionamento de um motor CC de 3V utilizado em robótica móvel através de um mosfet IRF540.

Figura 16. 3: Acionamento de motor com Mosfet.

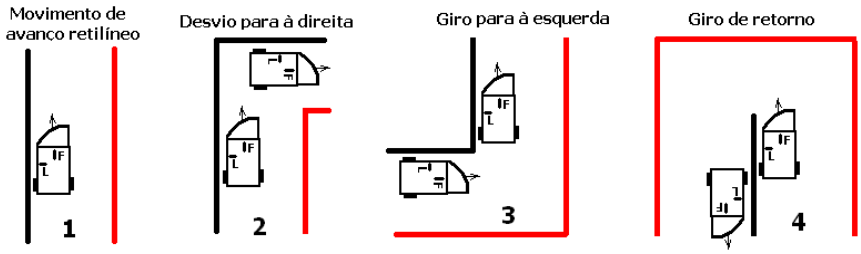


Fonte: Elaborado pelo autor.

16.5 Exemplo: seguidor ótico de labirinto

Neste caso é interessante definir que no princípio seguidor de parede o robô considera o obstáculo como referência a qual ele vai seguir em movimento de avanço normal. Nesse exemplo ele "cola" do lado esquerdo, ou seja, o motor direito deve ser mais rápido que o motor esquerdo, quando os dois estiverem acionados.Seguindo as paredes do labirinto até final encontram-se quatro situações, mostradas na Figuta 16.4.

Figura 16. 4: Situações encontradas em labirintos.



Fonte: Elaborado pelo autor.

É importante salientar que no princípio de seguir o obstáculo, o robô não pode tocar no outro lado, em vermelho, pois ele pode tomá-lo como referência e voltar, o que fará sair por onde entrou.

16.6 Estabilidade do controle de movimento

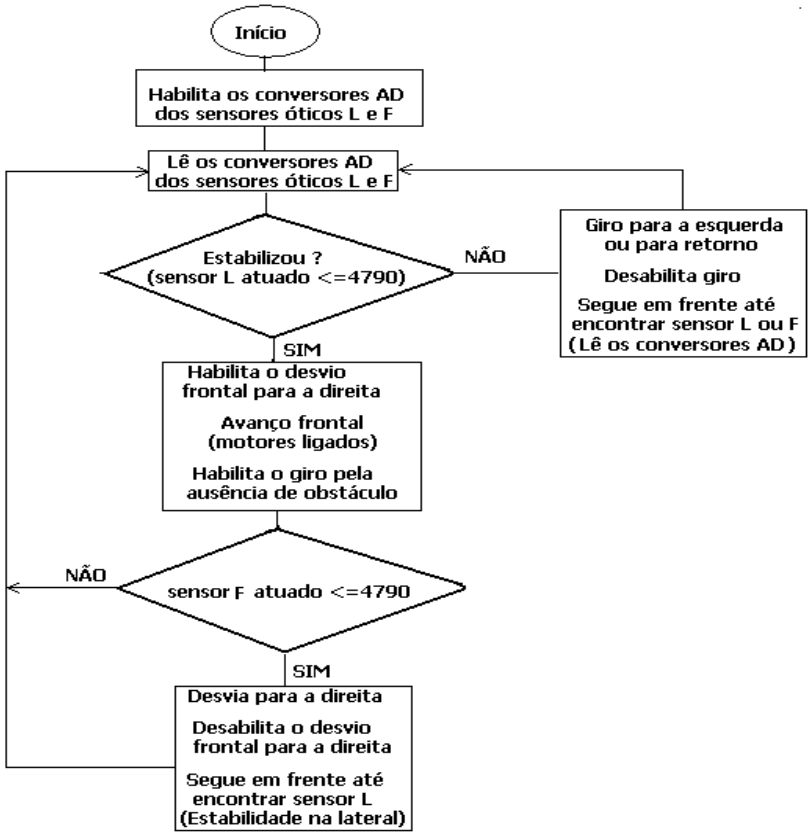
Para garantir estabilidade do controle de movimento, ou seja, garantir que o robô está seguindo corretamente a referência (o obstáculo), o sensor ótico lateral (L), com sinal analógico, deve ser lido frequentemente e estar com valores que garantam a presença do obstáculo.

Caso seja acusada a ausência de obstáculo, o microcontrolador deve parar o motor

esquerdo e acionar o motor direito, um determinado tempo, suficiente para garantir as situações 3 e 4. Note que o tempo de 4 é maior que o de 3, mas com o tempo de 4 na situação 3, o robô vai ser seguro pelo obstáculo até acabar o tempo de desvio e seguir em frente até encontrar obstáculo frontal ou lateral.

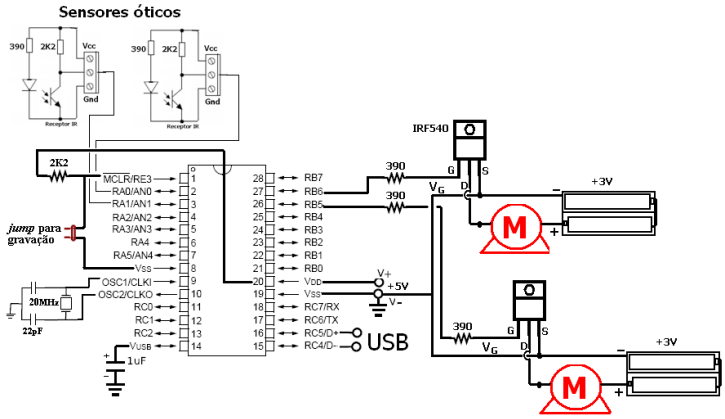
Caso o sensor frontal verifique obstáculo, mostrado na situação 2, o microcontrolador para o motor direito, aciona o motor esquerdo e segue em frente até encontrar obstáculo lateral, o que garante a estabilidade do movimento. Note que se o desvio para a direita for pouco, o guia oval frontal do robô conduzirá o robô à estabilidade ao tocar no obstáculo com o avanço frontal. O fluxograma do firmware do microcontrolador pode ser visto na Figura 16.5 e o circuito é mostrado na Figura 16.6.

Figura 16. 5: Fluxograma de funcionamento de robô móvel com sensores.



Fonte: Elaborado pelo autor.

Figura 16. 6: Uso de sensores óticos para controlar motores.

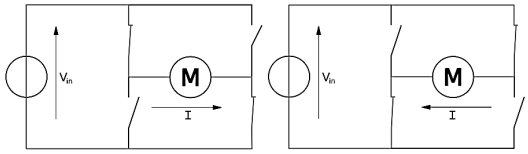


Fonte: Elaborado pelo autor.

16.1 Ponte H

O acionamento da ponte H (ver Figura 16.7) permite o movimento do motor nos dois sentidos. A ponte H pode ser feita com transistores MOSFETs, mais aconselhável devido a baixa queda de tensão, ou de potência Darllington TIP ou BD.

Figura 16. 7: Motor em Ponte H.



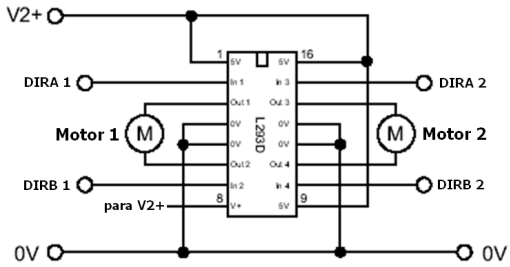
Fonte: Elaborado pelo autor.

16.2 Driver Ponte H L293D

Uma das soluções mais simples e barata em atuação de robôs móveis consiste utilizar um integrado *motor driver* como o L293D (ver Figura 16.8). Este integrado possibilita o controle de dois motores CC, utilizando quatro pinos de saída do microcontrolador. O circuito integrado L293D deve ter duas alimentações. Uma para comando (5V) no pino 16 e outra para potência (por exemplo 9,6 V ou 5V) no pino 8. Os motores percebem uma queda de 0,7V em relação à tensão da fonte externa.

As entradas nos pinos 2 e 7 são para acionar o motor A e entradas nos pinos 10 e 15 são para acionar o motor B. O pino 8 é conectado à fonte de alimentação dos motores que tem o mesmo Gnd do circuito de controle.

Figura 16. 8: CI Ponte H – L293D.



Fonte: Elaborado pelo autor.

A mudança dos sinais nos pinos de entrada tem o efeito de produzir a alteração do sentido da corrente no enrolamento do motor, logo do seu sentido de rotação. ATabela 16.1 permite programar o movimento em qualquer direção (conjugando 2 motores).

Tabela 16. 1: Comandos ponte H com CI L293D.

ENABLE	DIRA	DIRB	FUNÇÃO
H	H	L	Para Frente
H	L	H	Para trás
H	L/H	L/H	Stop Rápido
L	X	X	Stop Lento

Fonte: Elaborado pelo autor.

Se a primeira entrada alta, segunda entrada baixa , então o motor se desloca para frente, se a primeira entrada baixa e a segunda entrada alta , o motor se movimenta para trás. Se ambas as entradas baixas ou altas, o motor pára.

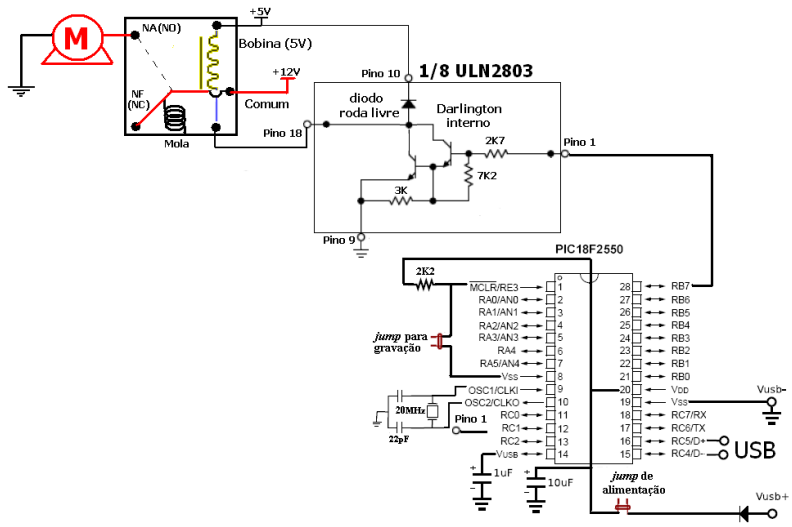
16.3 Solenóides e relés

Uma solenóide consiste num êmbolo de ferro colocado no interior de uma bobina (indutância) elétrica, enrolada em torno de um tubo. Quando se alimenta eletricamente a bobina, cria-se um campo magnético que atrai o êmbolo (núcleo móvel) para dentro da bobina. No caso de um relé, fecha um contato para circulação de outro nivel maior de corrente. Os relés são dispositivos comutadores eletromecânicos. O controle de uma solenóide ou relé pode ser feito pelo chaveamento de um transistor Darlington ou de um MOSFET mostrado abaixo.

O relé é um tipo de interruptor acionado eletricamente que permite o isolamento elétrico de dois circuitos. O relé é formado por um eletroímã (uma bobina enrolada sobre um núcleo de material ferromagnético) que quando acionado, através da atração eletromagnéti-

ca, fecha os contatos de um interruptor. Normalmente o interruptor de um relé tem duas posições, com isso existem dois tipos, os NF(normalmente fechado) e NA (normalmente aberto), como mostra a Figura 16.9. A bobina do relé é acionada por uma tensão contínua que é especificada de acordo com o fabricante, bobinas de 5, 12 e 24 Volts são as mais comuns.

Figura 16. 9: Acionamento de motor 12V com relé bobina 5V.



Fonte: Elaborado pelo autor.

Uma das características do relé é que ele pode ser energizado com correntes muito pequenas em relação à corrente que o circuito controlado exige para funcionar. Isso significa a possibilidade de controlar circuitos de altas correntes como motores, lâmpadas e máquinas industriais, diretamente a partir de microcontroladores.

16.4 Driver de potência ULN2803

Um *driver* de potência é utilizado sempre quando se necessita acionar um *hardware* especifico de maior potência. Esse driver pode ser usado para controlar motores de passos, solenóides, relés, motores CC e vários outros dispositivos. Ele contém internamente 8 transistores Darlington NPN de potência, oito resistores de base de 2K7 e oito diodos de roda livre, para descarregar no Vcc (pino 10) a corrente reversa da força contra-eletromotriz gerada no chaveamento dos transistores, protegendo os mesmos.

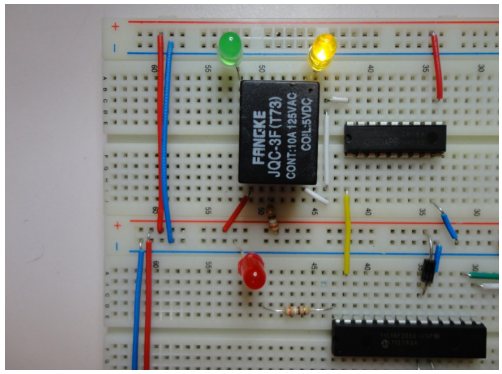
Quando o microcontrolador coloca +5V (nivel lógico 1) no pino 1 do driver ULN2803, ele conecta o pino 18 do outro lado, onde está liga um pólo do motor, ao Gnd (nivel lógico 0). Como o outro lado da bobina (o comum) ou do motor deve estar ligado ao Vcc da fonte de até 30V, esse comando irá energizar a bobina ou o motor.

Nesta prática será possível ligar/desligar cargas de tensões mais elevadas que a do

circuito, que é 5V da USB. Se estiver sendo utilizado um relé 5Vcc/220Vca, é possível ligar lâmpadas e até eletrodomésticos, como ventiladores, televisores, rádios; sendo necessário apenas conectar 220Vca ao comum do relé. É necessário apenas acrescentar um driver de potência, o ULN 2803, que dispõe de 8 transistores internos, ou seja, com ele é possível acionar até 8 relés, consequentemente, 8 cargas.

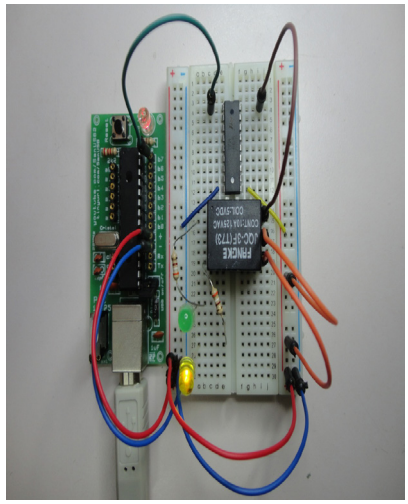
Inicialmente serão utilizados 2 LEDs conectados aos contatos NA e NF do relé apenas para testar seu funcionamento. Enquanto um LED acender, o outro estará apagado. Para alimentar o LED, será utilizada a tensão de 5V da própria USB do circuito SanUSB. A Figura 16.10 mostra detalhes da montagem em protoboard e a Figura 16.11 apresenta a conexão com a placa SanUSB.

Figura 16. 10: Esquemático Prática relés.



Fonte: Elaborado pelo autor.

Figura 16. 11: Prática Microrrelés, montada em protoboard.

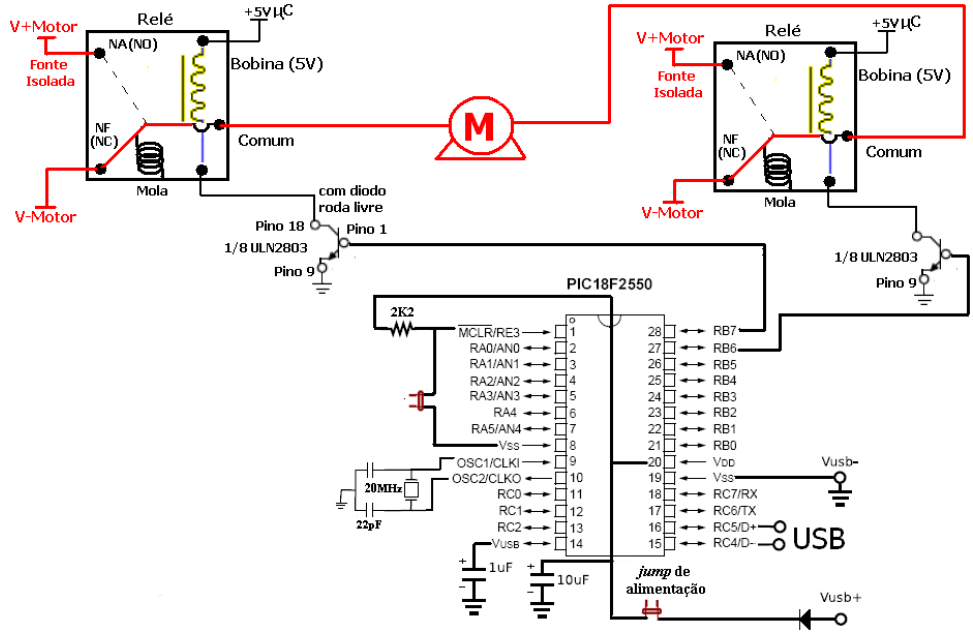


Fonte: Elaborado pelo autor.

16.5 Ponte H com microrelés

Como foi visto, acionamento da ponte H permite o movimento do motor nos dois sentidos. A ponte H pode ser feita também com apenas dois microrelés. Neste caso, pode-se utilizar também o driver ULN2803 para a energização das bobinas, pois já contém internamente oito transistores com resistores de base e oito diodos de roda livre. Esse tipo de ponte H, mostrada na Figura 16.12, não causa queda de tensão na fonte de alimentação do motor, porque as fontes de energização da bobina do microrelé e de alimentação do motor devem ser diferentes, ou seja, isoladas uma da outra, para que seja possível o chaveamento do relé.

Figura 16. 12: Acionamento de motor nos dois sentidos com relés em Ponte H.



Fonte: Elaborado pelo autor.

Note que inicialmente os dois relés estão conectados ao V-Motor. Ao energizar a bobina do relé da esquerda, conectando o V+Motor, a corrente da fonte passa pelo motor no sentido da esquerda para a direita o que determina o sentido de rotação do motor. Ao desligar o relé da esquerda e acionar o relé da direita ocorre o sentido de rotação inverso do motor.

Quando se utiliza motor CC em ponte H para atuadores robóticos, como rodas de veículos ou braços mecânicos, o que determina o torque e a velocidade do atuador é a relação de transmissão da caixa de engrenagens conectada ao motor.

Capítulo 17
Acionamento de motores de passo

Motores de passos são dispositivos mecânicos eletromagnéticos que podem ser controlados digitalmente.

A crescente popularidade dos motores de passo se deve à total adaptação desses dispositivos à lógica digital. São encontrados não só em aparelhos onde a precisão é um fator muito importante como impressoras, plotters, scanners, drivers de disquetes, discos rígidos, mas também, como interface entre CPUs e movimento mecânico, constituindo, em suma, a chave para a Robótica.

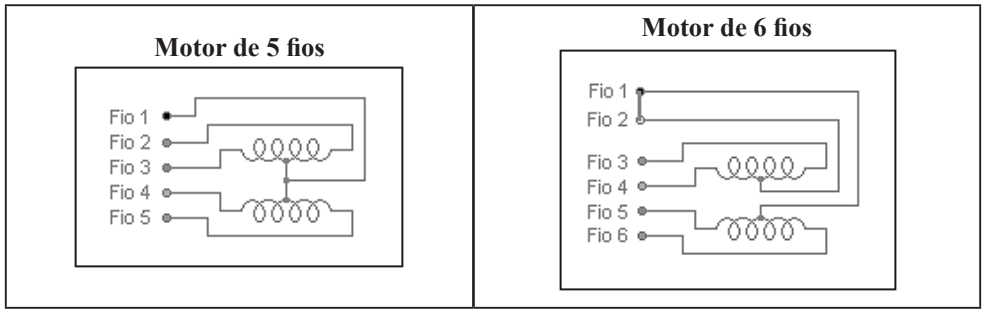
17.1 Motores de passo unipolares

Os motores de passo unipolares são facilmente reconhecidos pela derivação ao centro das bobinas. O motor de passo tem 4 fases porque o número de fases é duas vezes o número de bobinas, uma vez que cada bobina se encontra dividida em duas pela derivação ao centro das bobinas (comum).

Normalmente, a derivação central das bobinas está ligada ao terminal positivo da fonte de alimentação (Vcc) e os terminais de cada bobina são ligados alternadamente à terra através de chaveamento eletrônico produzindo movimento.

As bobinas se localizam no estator e o rotor é um imã permanente com 6 pólos ao longo da circunferência. Para que haja uma maior resolução angular, o rotor deverá conter mais pólos.Os motores de passo unipolares mais encontrados possuem 5 ou 6 fios. Os motores de passo unipolares de 6 fios possuem dois **fios comuns (derivação central)**. Para o acionamento do motor de passo, estes fios comuns devem ser ligados à fonte de alimentação (+5V ou +12V) e os terminais da bobina ligados ao controle de chaveamento do motor de passo. A Figura 17.1 ilustra as bobinas de um motor de passo unipolar.

Figura 17. 1: Motores de passo unipolar – conexão interna.



Fonte: Elaborado pelo autor.

Para descobrir os terminais de um motor de passo, deve-se considerar que:

Para motores de 6 fios, a resistência entre os fios comuns (Fio 1 e Fio 2) é infinita por se tratarem de bobinas diferentes.

A resistência entre o fio comum (Fio 1) e o terminal de uma bobina é a metade da resistência entre dois terminais desta bobina.

Para encontrar a seqüência correta dos fios para chaveamento das bobinas, pode-se ligar manualmente o fio comum ao Vcc, e de forma alternada e seqüencial, o GND (terra) da fonte aos terminais das bobinas, verificando o movimento do motor de passo.

17.2 Modos de operação de um motor de passo unipolar

A Tabela 17.1 ilustra os comandos para acionamento de um motor de passo.

Tabela 17. 1: Características e lógica de acionamento de motor de passo.

PASSO COMPLETO 1 (FULL-STEP)

-***Somente meia bobina é energizada a cada passo a partir do comum***;
-Menor torque;
-Pouco consumo de energia.

Nº do passo	1a	2a	1b	2b	Decimal
1-->	**1**	0	0	0	8
2-->	0	**1**	0	0	4
3-->	0	0	**1**	0	2
4-->	0	0	0	**1**	1

PASSO COMPLETO 2 (FULL-STEP 2)

-***Duas meia-bobinas são energizadas a cada passo***;
-Maior torque;
-Consome mais energia que o Passo completo 1.

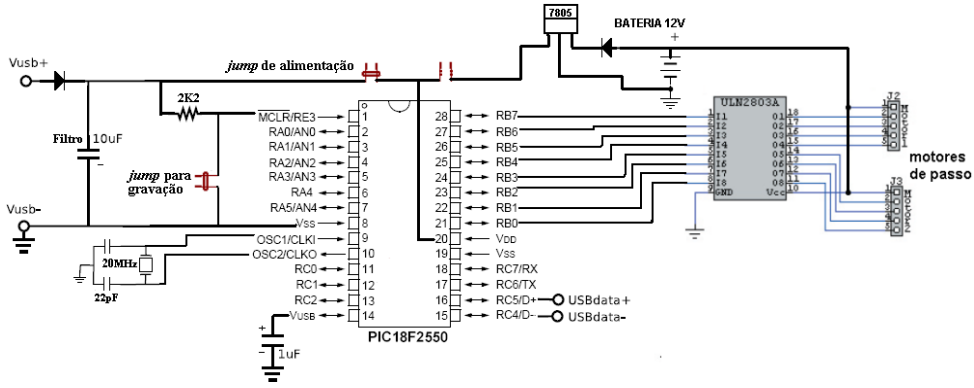
Nº do passo	1a	2a	1b	2b	Decimal
1-->	**1**	**1**	0	0	8
2-->	0	**1**	**1**	0	4
3-->	0	0	**1**	**1**	2
4-->	**1**	0	0	**1**	1

Fonte: Elaborado pelo autor.

17.3 Acionamento bidirecional de dois motores de passo

Como o driver de potência ULN2803 ou ULN2804 possui internamente 8 transistores de potência, este é capaz de manipular dois motores de passo ao mesmo tempo. Ele contém internamente oito diodos de roda livre e oito resistores de base dos transistores, o que possibilita a ligação direta ao microcontrolador e aos motores de passo (ver Figura 17.2).

Figura 17. 2: Conexão do motor de passo no PIC.



Fonte: Elaborado pelo autor.

A bateria para motores de passo deve ter uma corrente suficiente para energizar as bobinas do motor de passo. Dessa forma, é possível associar baterias 9V em paralelo para aumentar a corrente de energização ou utilizar baterias de *No-Breaks*.

O firmware do controle de posição de um motor de passo unipolar utilizando LCD está descrito abaixo.

```
#include "SanUSB1.h"
#include "lcd.h"
unsigned int vpm;
unsigned char buffer[20];
int limpa;
int voltas;
int contador;
int j;
#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao(){ }
void main() {
clock_int_4MHz();
lcd_ini();
lcd_comando(LCD_CLEAR);
lcd_comando(LCD_CURSOR_OFF);
vpm = 3;
contador = 0;
```

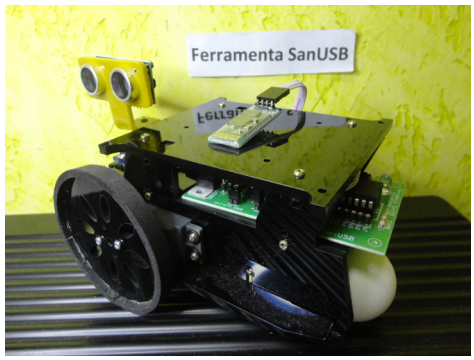
```
voltas = 0;
limpa=1;
nivel_baixo(pin_a3);
nivel_baixo(pin_a2);
nivel_baixo(pin_a1);
nivel_baixo(pin_a0);
while (1) {
   if(voltas)
   {
      lcd_comando(LCD_CLEAR);
      lcd_comando(LCD_CURSOR_OFF);
      lcd_escreve(1, 0, "Contando...     ");
      //sprintf(buffer,"voltas %d / %d",(vpm-voltas),vpm);

 sprintf(buffer,"Max: %d voltas",vpm);
      lcd_escreve2(2, 0, buffer);
            while(voltas)
      {
      nivel_alto(pin_a3);
      tempo_ms(2);
      nivel_alto(pin_a2);
      tempo_ms(2);
      nivel_baixo(pin_a3);
      tempo_ms(2);
      nivel_alto(pin_a1);
      tempo_ms(2);
      nivel_baixo(pin_a2);
      tempo_ms(2);
      nivel_alto(pin_a0);
      tempo_ms(2);
      nivel_baixo(pin_a1);
      tempo_ms(2);
      nivel_alto(pin_a3);
      tempo_ms(2);
      nivel_baixo(pin_a0);
      if(entrada_pin_b6==0) // sensor magnético ( read switch )
      {
      while(entrada_pin_b6==0)
      {
      nivel_alto(pin_a3);    // <---- Alteração aqui adicionado
```

Capítulo 18
Servo-motores

Há dois tipos de servomotores: os de posição, com giro de 180º, e o de rotação, que possui o giro contínuo. O Servo de Posição é utilizado em antenas parabólicas, em braços robóticos, na robótica móvel terrestre com o controle de pernas mecânicas e no controle de câmeras. O Servo de Rotação é prioritariamente escolhido para a locomoção por rodas.

Trata-se de dispositivos muito precisos que giram sempre o mesmo ângulo para um dado sinal. Um servo típico possui três fios de ligação, normalmente preto, vermelho e branco (ou amarelo). O condutor preto é a referência de massa da alimentação (0 volts), o condutor vermelho é a alimentação e o condutor branco (ou amarelo) é o sinal de posicionamento, como é mostrado na figura 18.1 que é um robô para desvio de obstáculos, acionado por dois servo-motores de rotação. O sinal do servo-motor de posição é normalmente um pulso de 1 a 2 milisegundos (ms), repetido depois de um pulso de 10 a 20ms. Com o pulso de aproximadamente 1 ms o servo move-se para um sentido e com o impulso de 2 ms para o sentido oposto. Desse modo, com um impulso de 1,5 ms, o servo roda para a posição central.

Figura 18. 1: Aplicação de servo-motores em robô móvel.



Fonte: Elaborado pelo autor.

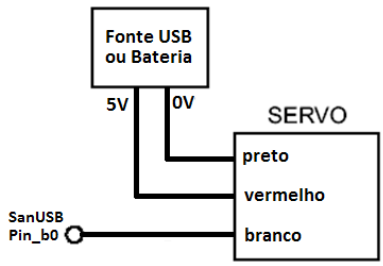
A tensão de alimentação do servomotor é tipicamente de 5V, podendo variar entre 4,5V e 6V. Devido à alta redução do jogo de engrenagens, o torque que se obtém de um servo é bastante alto, considerando o seu tamanho reduzido. Lamentavelmente, os servos consomem correntes elevadas (de 200 mA a 1 A) e introduzem ruído elétrico nos condutores de alimentação, necessitando a aplicação de capacitores de filtro. O programa abaixo move um

servo-motor de rotação para frente e um outro para trás. Note que essa operação é utilizada por robôs móveis que possuem dois motores em anti-paralelo.

```
#include “SanUSB1.h”
#define motor1 pin_b5
#define motor2 pin_b6
#pragma interrupt interrupcao
void interrupcao(){}
int16 frente=50;
short int led;
void main(){
clock_int_4MHz();
while (TRUE)
{
while (frente>0)
{  nivel_alto(motor2);              //Inicializa o pulso do motor 1
nivel_alto(motor1);              //Inicializa o pulso do motor 2
tempo_ms(1);
nivel_baixo(motor1);      //Termina o pulso de 1ms do motor1– sentido horário
tempo_ms(1);
nivel_baixo(motor1);
nivel_baixo(motor2);        //Termina o pulso de 2ms do motor2 – sentido anti-horário
tempo_ms(10);
--frente;
}
frente=50;
led=!led;    //pica led a cada 50 ciclos do servo-motor, ou seja a cada 12ms*50 = 600ms
output_bit(pin_b7,led);
}}
```

A Figura 18.2 ilustra as conexões de um servomotor.

Figura 18. 2: Visualização interna de um servo-motor.



Fonte: Elaborado pelo autor.

O programa abaixo move um servo-motor de posição. Com o pulso de aproximadamente 1 ms, o servo move-se para 0 graus, e com o pulso de 1,2 ms para 90 graus e com o pulso de 2 ms para 180 graus. Note que este motor é utiliz do em antenas parabólicas, em sistemas robóticos e no controle de câmeras.

```
#include "SanUSB1.h" //Servo-motor de parabólica - 3 posições definidas: 0, 90 e 180 graus.

#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao(){                }

Unsigned int  FRENTE=200, TRAS=200; //no servo de teste 200 passos corresponderam a
aprox. 1 volta

void main(){
    clock_int_4MHz();
    while (1){
    while (FRENTE>0){
    FRENTE--;
    nivel_alto(pin_b0);
    tempo_ms(1);        // tempo de busca de 0 graus
    nivel_baixo(pin_b0);
    tempo_ms(10);}
    FRENTE=200;

      while (TRAS>0){
          TRAS--;
          nivel_alto(pin_b0);
          tempo_ms(2);       // tempo de busca de 180 graus
          nivel_baixo(pin_b0);
      tempo_ms(10);}
      TRAS=200;
      inverte_saida(pin_b7);}}
```

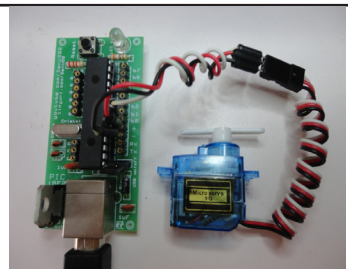
O micro-servomotor da Figura 18.4 suporta 1,6kg e apresenta apenas 3 fios, conforme Tabela 18.1.

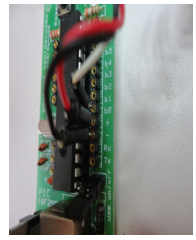
Tabela 18. 1: Pinos do servomotor.

BRANCO	SINAL(PINO B0)
VERMELHO	VCC (5V) +
PRETO	GND (0V) -

Fonte: Elaborado pelo autor.

Figura 18. 3: Micro-servomotor 1,6kg com placa SanUSB.

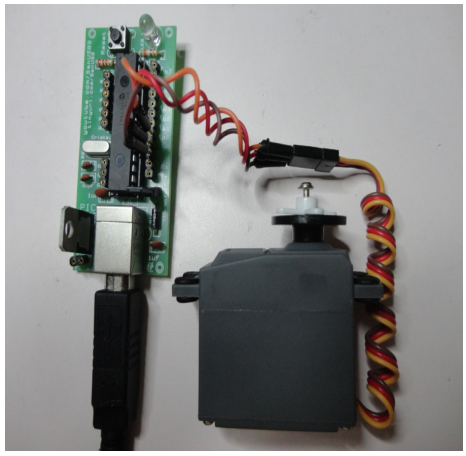




Fonte: Elaborado pelo autor.

Na Figura 18.4, é possível observar a mesma ligação feita para o servo de10kg.

Figura 18. 4: Micro-servomotor 10kg com placa SanUSB.



Fonte: Elaborado pelo autor.

Código em C para MPLABX Micro-servomotor:

```
#include "SanUSB1.h"
int x;
#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao(){                }
void main(){
clock_int_4MHz();
   while(true){
   nivel_alto(pin_b7);
      for(x=0;x<40;x++){
      nivel_alto(pin_b0);
      tempo_ms(2);
      nivel_baixo(pin_b0);
      tempo_ms(10);}
      for(x=0;x<40;x++){
      nivel_alto(pin_b0);
      tempo_ms(1);
      nivel_baixo(pin_b0);
      tempo_ms(11);}
      for(x=0;x<40;x++){
      nivel_alto(pin_b0);
      tempo_ms(0);
      nivel_baixo(pin_b0);
      tempo_ms(12);
      }   }}
```

Código em C para MPLABX Servomotor:

```
#include "SanUSB1.h"
int x;
#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao(){                }
void main(){
clock_int_4MHz();
   while(true){
   nivel_alto(pin_b7);
      for(x=0;x<200;x++){//anti horario
        nivel_alto(pin_b0);
        tempo_ms(2);
        nivel_baixo(pin_b0);
        tempo_ms(8);}
      for(x=0;x<40;x++){//parado
        nivel_alto(pin_b0);
        tempo_ms(0);
        nivel_baixo(pin_b0);
        tempo_ms(10);}
      for(x=0;x<200;x++){//horario
        nivel_alto(pin_b0);
        tempo_ms(1);
        nivel_baixo(pin_b0);
        tempo_ms(9);
      }   }}
```

Capítulo 19
Fotoacopladores e sensores infravermelhos

Fotoacopladores ou optoisoladores proporcionam a isolação de sinais em uma grande variedade de aplicações. Também chamados de acopladores óticos, eles comutam ou transmitem sinais e informações ao mesmo tempo que isolam diferentes partes de um circuito.

Optoisoladores lineares são usados para isolar sinais análogos até 10MHz, enquanto optoisoladores digitais são usados para controle, indicação de estados, isolação de sinais de comando e mudanças de níveis lógicos.

Existem fotoacopladores de diversos tipos e com construções internas diversas, como, por exemplo, acopladores onde a construção interna é baseada em um diodo infravermelho e um fototransistor. Como exemplo podemos citar o 4N25 e o TIL111.

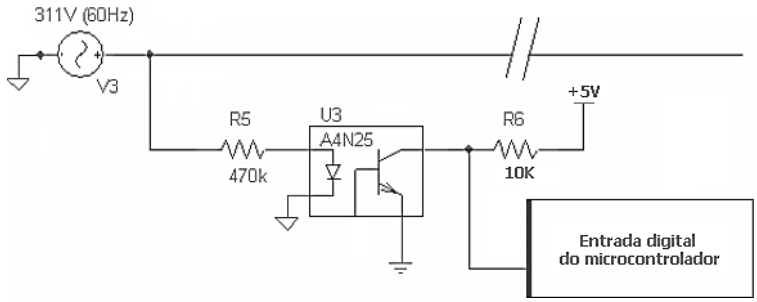
Esse dispositivo pode ser utilizado por um microcontrolador para identificar a presença de tensão 220VAC em um determinado ponto. A potência máxima dissipada por esse componente é de 250mW em 25 graus Celsius. Dessa forma, deve-se dimensionar um resistor em série com o foto-diodo interno para protegê-lo.

Escolhendo resistores são de 333mW, ou seja, a potência máxima que pode ser dissipada em cada um deles. É interessante que exista um certo intervalo de segurança entre a potência máxima do componente e a potência máxima dissipada. Então, a potência máxima escolhida para os resistores é de 200mW. Na Equação abaixo é calculado o resistor que será utilizado no circuito, considerando a tensão de pico. Considere 311V como a tensão de pico.

$P=V^2/R \rightarrow 0,2W = (311)^2/R \rightarrow R=483\ K\Omega$.

O resistor comercial mais próximo desse valor é 470KΩ (ver Figura 19.1).

Figura 19. 1: Aplicação de um fotoacoplador.

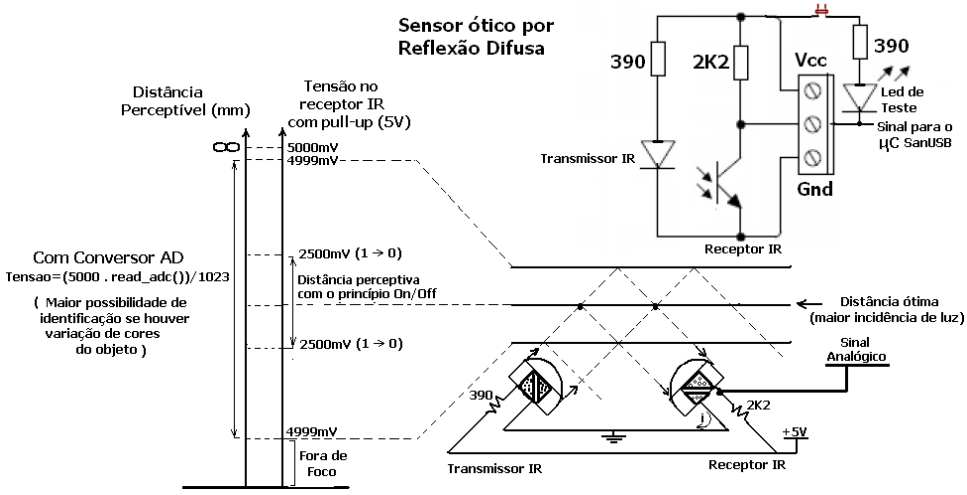


Fonte: Elaborado pelo autor.

19.1 Transmissor e receptor IR

Os transmissores e receptores IR (*infrared* ou Infravermellhos) são muito utilizados como sensor ótico por reflexão difusa para registro de posição. A Figura 19.2 mostra o circuito do sensor e um gráfico do sinal de saída (em mV) do receptor IR em função da distância perceptível pelo receptor IR com resistor de elevação de tensão para 5V (*pull-up* 2K2).

Figura 19. 2: Conexão do par infravermelho: TIL 32 (emissor) e TIL 78 (receptor).



Fonte: Elaborado pelo autor.

Para ter um maior alcance e perceber uma maior variação de posição, com esse sensor, é aconselhável utilizar o conversor AD de 10 bits do microcontrolador para identificar variações de até 5mV no sinal do sensor. A distância ótima é a distância em que incide no receptor a maior parte do feixe de luz emitido pelo transmissor. Nessa distância ocorre a maior variação (queda) do sinal de saída analógico do receptor IR.

Utilizando o princípio *on/off*, só há identificação de posição quando o sinal do receptor for menor ou igual a 2,5V (nivel lógico baixo), o que torna o sensoreamento muito limitado.

Durante os testes desse circuito foi verificado que, devido a resistência de 390Ω em paralelo com 2K2, quando o led é conectado no circuito, há uma diminuição na variação do sinal de saída analógico do receptor IR. O programa abaixo mostra a leitura em mV do sensor ótico via emulação serial.

```
#include "SanUSB1.h" //Leitura de tensão em mV com infrared

#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao(){                }

Long int  tensao;
void main() {
clock_int_4MHz();

setup_adc_ports(AN0);  //Habilita entrada analógica - A0
setup_adc(ADC_CLOCK_INTERNAL);

while(1){                          //ANALÓGICO          DIGITAL(10 bits)
set_adc_channel(0);                // 5000 mV                 1023
tempo_ms(10);                      // tensao                read_adc()
tensao= (5000*(int32)read_adc())/1023;
printf ("\r\nA tensao e' = %lu mV\r\n",tensao); // Imprime pela serial bluetooth

inverte_saida(pin_b7);
tempo_ms(500);}}
```

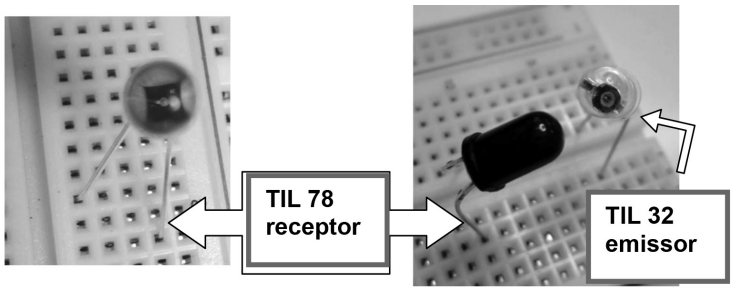
Durante os testes desse circuito foi verificado que, devido a resistência de 390 Ω em paralelo com 2K2 Ω, quando o led é conectado no circuito, há uma diminuição na variação do sinal de saída analógico do receptor IR. O programa a seguir mostra a leitura em mV do sensor ótico via emulação serial.

```
        #include “SanUSB1.h” //Leitura de tensão em mV com variação de um //potenciômetro
        #include <usb_san_cdc.h>// Biblioteca para comunicação serial virtual
        int32 tensao;
#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao(){                }
        main() {
        usb_cdc_init(); // Inicializa o protocolo CDC
        usb_init(); // Inicializa o protocolo USB
        usb_task(); // Une o periférico com a usb do PC
        setup_adc_ports(AN0);  //Habilita entrada analógica - A0
        setup_adc(ADC_CLOCK_INTERNAL);

        while(1){                          //ANALÓGICO          DIGITAL(10 bits)
        set_adc_channel(0);                // 5000 mV                 1023
        delay_ms(10);                      // tensao                read_adc()
        tensao= (5000*(int32)read_adc())/1023;
        printf (usb_cdc_putc,"\r\nA tensao e' = %lu mV\r\n",tensao); // Imprime pela serial //virtual
        inverte_saida(pin_b7);
        delay_ms(500);  }}
```

Deve-se atentar que no receptor infravermelho o catodo e o anodo são invertidos, ou seja, no TIL 78, o terminal que vai para o GND é o Anodo.Se o receptor não for escuro, preto ou azul, mas sim transparente, deve-se diferenciar como mostra a Figura 19.3. O emis-

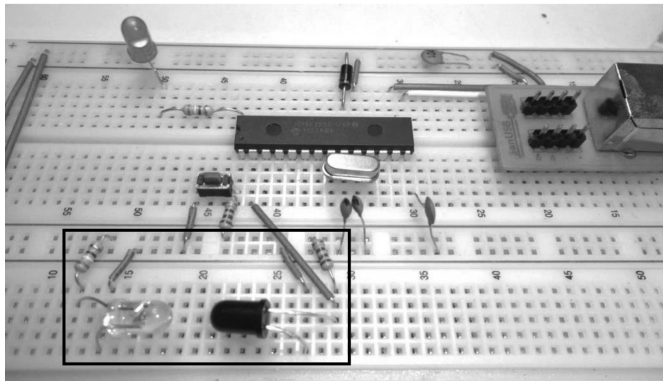
sor, visto frontalmente, apresenta contato circular, enquanto o receptor, quadrado, conforme mostrado na Figura 19.3. A Figura 19.4 mostra o esquemático de conexão.

Figura 19. 3: Par infravermelho.



Fonte: Elaborado pelo autor.

Figura 19. 4: Sensor Infravermelho, montado em protoboard.



Fonte: Elaborado pelo autor.

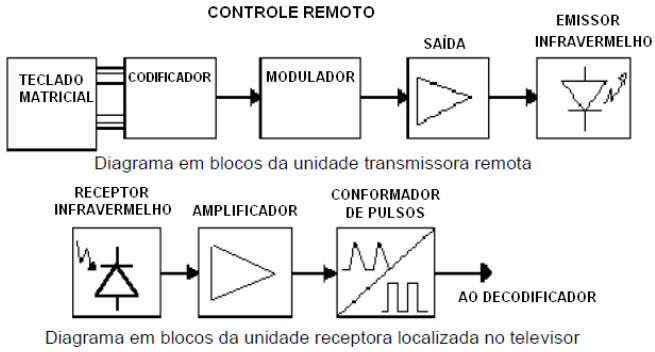
Capítulo 20
Automação e domótica com controle remoto universal

A comunicação entre uma unidade remota e um eletrodoméstico, como uma TV, se dá geralmente por emissão de radiação infravermelha modulada por pulsos (ver ilustração na Figura 20.1).

Para tornar o sistema insensível a interferências e filtrar os ruídos, aceitando apenas as ordens do controle remoto, o código de pulsos do emissor contém um dado binário, que é identificado pelo decodificador do receptor.

As interferências podem se originar de fontes estáticas, isto é, que não pulsam, como o sol, lâmpadas incandescentes, aquecedores, e de fontes dinâmicas que são mais intensas e geram maior interferência, como lâmpadas fluorescentes, a imagem da TV, outros transmissores de infravermelho, entre outros.

Figura 20. 1: Diagrama de blocos de comunicação infravermelha.



Fonte: Elaborado pelo autor.

O receptor, geralmente contido num único invólucro montado no painel frontal do televisor, entrega ao decodificador apenas os pulsos retangulares correspondentes aos códigos de identificação e dados, eliminando a maioria das fontes de interferências, exceto as que tenham a mesma frequência de pulsos, cabendo a rejeição destas ao Decodificador, se não tiverem o mesmo código de pulsos.

Para acionar uma carga à distância basta ter o controle remoto e o receptor infravermelho, pois ao invés de capturar o código em bits emitidos pelo controle remoto para decodificação, é possível identificar apenas o start bit desse código que apresenta nivel lógico baixo (0V) que, conectado ao pino de interrupção externa (B0) do microcontrolador com um resistor de *pull-up* de 2K2, executará uma tarefa desejada como, por exemplo, o chaveamento de um relé para acionamento de uma máquina, conforme Figura 20.2.

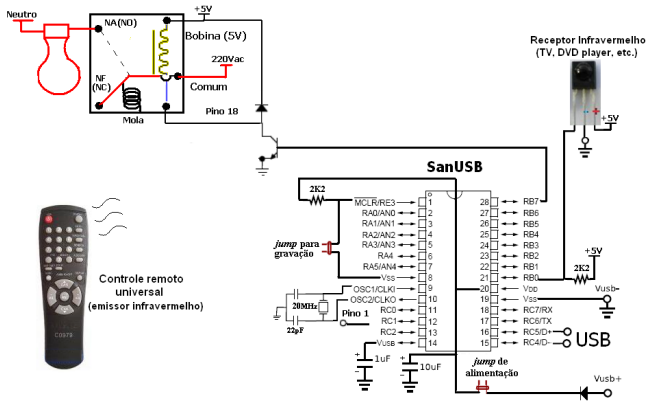
Note que, se nesse caso não houver um sistema de decodificação, o receptor deve ter

um invólucro para proteção contra interferências, pois está sujeito às fontes estáticas e dinâmicas. Abaixo um programa exemplo de acionamento de um relé através de um controle remoto universal.

```
#include “SanUSB1.h”
short int rele;
#pragma interrupt interrupcao //Tem que estar aqui ou dentro do firmware.c
void interrupcao() {
rele=!rele;
output_bit(pin_b5,rele);
tempo_ms(1000); } //Tempo para deixar o receptor cego por 1 seg após a 1ª atuação da
interrupção
void main() {
    clock_int_4MHz();
    enable_interrupts (global); // Possibilita todas interrupcoes
    enable interrupts(int_ext);//Habilita int. ext. 0 no pino B0 onde está o receptor infravermelho
_
      while (TRUE) {
        inverte_saida(pin_B7);
        tempo_ms(500);
}}
```

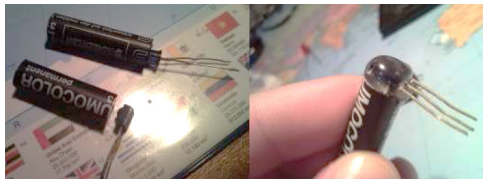
Para filtrar as interferências dinâmicas é necessário colocar o receptor em uma caixa preta com um pequeno orifício ou em um tubo feito de caneta com cola quente, como mostra a Figura 20.3, para receber somente a luz IR direcional.

Figura 20. 2: Conexão de receptor infravermelho de TV no PIC.



Fonte: Elaborado pelo autor.

Figura 20. 3: Exemplo de proteção do receptor contra emissões externas de raios IR.



Fonte: Elaborado pelo autor.

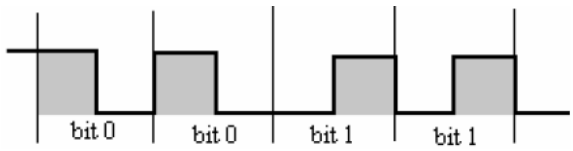
Capítulo 21

Codificação de receptor infravermelho utilizando a norma RC5

RC5 é uma norma universal desenvolvida pela Phillips para comandos a distância por infravermelho utilizada principalmente em equipamentos de áudio, televisores, videocassetes e outros aparelhos domésticos, com uma área de alcance de aproximadamente 10m.

A norma RC5 usa modulação bifásica, ou seja, código Manchester. Cada bit é separado por dois semi-ciclos; a metade esquerda e direita têm níveis opostos. Se o bit a ser transmitido for zero (0), o seu lado esquerdo (primeiro semi-ciclo) é um e o seu lado direito é zero (segundo semi-ciclo). Se o bit a ser transmitido for um (1), o seu lado esquerdo é zero quando o seu lado direito é um, conforme Figura 21.1.

Figura 21. 1: Modulação Bifásica.



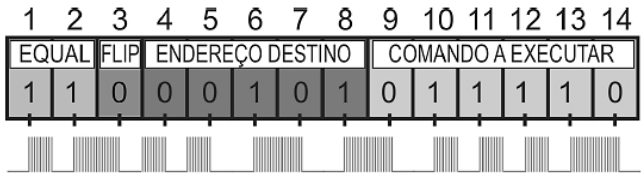
Fonte: Elaborado pelo autor.

O código transmitido consiste de uma palavra de 14 bits, sendo eles :

- 2 bits para ajuste do nivel (2 start bits). O primeiro e o segundo corresponde a 1;
- 1 bit para controle (toggle bit ou flip) que muda de estado lógico cada vez que um botão é pressionado na unidade de comando a distância. Isto serve para indicar se o botão foi pressionado uma vez ou se continua sendo pressionado;
- 5 bits de endereço do sistema para seleção de 1 dos 32 sistemas possíveis. Isso define o tipo de aparelho que se pretende controlar;
- 6 bits de comando representando 1 dos 128 comandos possíveis. Isso define a ação que se pretende executar em um determinado aparelho (sistema) selecionado.

Na Figura 21.2 é possível visualizar os bits de 1 a 14 e sua identificação: em azul claro os start bits 1 e 2 de equalização; o bit 3, FLIP, em amarelo; bits de 4 a 8 em azul indicando o endereçamento; e os bits de 9 a 14, indicando o comando a executar.

Figura 21. 2: Sinal transmitido.



Fonte: Elaborado pelo autor.

Na norma RC5, os dados são modulados numa frequência portadora de 30 a 40KHz, indicados pelos dois últimos números do receptor TSOP1740, TSOP4836, TSOP4830. Considerando uma modulação de 36 kHz, o período de cada bit corresponde a 64 pulsos de 1/36 kHz, ou seja, 64 vezes 27,7 us (micro segundos), o em que corresponde a aproximadamente 1772us. O programa abaixo identifica o o endereço do sistema e o comando das teclas pressionadas por um controle remoto RC5.

```
#include “SanUSB1.h” //Programa com protocolo RC5 da Phillips de controle remoto

char chegoupc, comando, sistema, bit_rx;
int32 i;

void identifica_bit(void){
bit_rx=0;
if (input(pin_b0)==1) {bit_rx=1;} //identifica 0 ou 1
tempo_us(860); // //pula para o próximo período para outra leitura
}
#pragma interrupt interrupcao
void interrupcao(){
inverte_saida(pin_b6);
sistema=0;
comando=0;
//*************************************************
tempo_us(750); //Tempo do start bit 1 com a perda do tempo do primeiro semi-ciclo alto pela interrupção
tempo_us(1720); //Tempo do start bit 2
tempo_us(1720); //Tempo do toogle bit

tempo_us(860); //pula o primeiro semi-ciclo
identifica_bit();
if (bit_rx) sistema|=16; //Bit 5 do byte sistema 0b00010000

tempo_us(860); //pula o primeiro semi-ciclo
identifica_bit();
if (bit_rx) sistema|=8; //Bit 4 do byte sistema 0b00001000
tempo_us(860); //pula o primeiro semi-ciclo
identifica_bit();
if (bit_rx) sistema|=4; //Bit 3 do byte sistema 0b00000100

tempo_us(860); //pula o primeiro semi-ciclo
identifica_bit();
```

```
if (bit_rx) sistema|=2; //Bit 2 do byte sistema 0b00000010

tempo_us(860); //pula o primeiro semi-ciclo
identifica_bit();
if (bit_rx) sistema|=1; //Bit 1 do byte sistema 0b00000001

//*************************************************

    tempo_us(860); //pula o primeiro semi-ciclo
    identifica_bit();
    if (bit_rx) comando|=32; //Bit 4 do byte comando

    tempo_us(860); //pula o primeiro semi-ciclo
    identifica_bit();
    if (bit_rx) comando|=16; //Bit 5 do byte comando

    tempo_us(860); //pula o primeiro semi-ciclo
    identifica_bit();
    if (bit_rx) comando|=8; //Bit 4 do byte comando

    tempo_us(860); //pula o primeiro semi-ciclo
    identifica_bit();
    if (bit_rx) comando|=4; //Bit 3 do byte comando
```

O resultado dos comandos gerados corresponde às teclas pressionadas de um controle remoto Phillips RC5.

```
    tempo_us(860); //pula o primeiro semi-ciclo
    identifica_bit();
    if (bit_rx) comando|=2; //Bit 2 do byte comando

    tempo_us(860); //pula o primeiro semi-ciclo
    identifica_bit();
    if (bit_rx) comando|=1; //Bit 1 do byte comando
//*************************************************
escreve_eeprom(0x10,sistema); escreve_eeprom(0x11,comando);    ////guarda as variáveis
decodificadas
while(le_eeprom(0xfd));
printf ("comando: %x\r\n",le_eeprom(0x11));
tempo_ms(500); //Tempo para deixar o receptor cego por um segundo após a primeira atuação
da interrupção
}

void main() {
clock_int_4MHz();

habilita_interruptcao(int0);

while(1){
      if (kbhit(1)) //avisa se chegou dados do PC
      {
      chegoupc=getc(); //se chegou, retém o caractere e compara com 'L' ou 'D'
      if (chegoupc=='L') {inverte_saida(pin_b6); printf("\r\nTesta led!\r\n");}
      }
      ++i; if (i>=10000) {i=0; inverte_saida(pin_b7);} //Led de visualização
      }}
```

Capítulo 22

LCD (display de cristal líquido)

O LCD, ou seja, display de cristal líquido, é um dos periféricos mais utilizados como dispositivo de saída em sistemas eletrônicos. Ele contém um microprocessador de controle, uma RAM interna que mantém *escritos* no display (DDRAM) os dados enviados pelo microcontrolador e uma RAM de construção de caracteres especiais (CGRAM). Os LCDs são encontrados nas configurações previstas na Tabela 22.1.

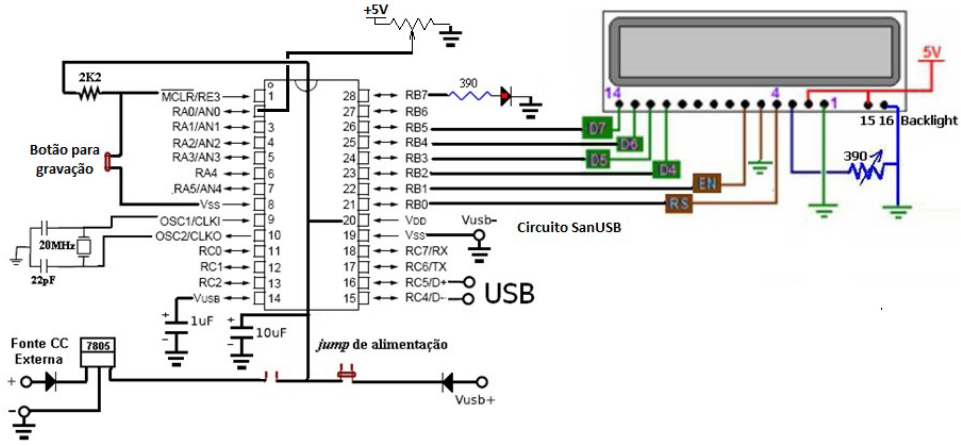
Tabela 22. 1: Configurações do LCD.

Número de Colunas	Número de Linhas	Quantidade de pinos
8	2	14
12	2	14/15
16	1	14/16
16	2	14/16
16	4	14/16
20	1	14/16
20	2	14/16
20	4	14/16
24	2	14/16
24	4	14/16
40	2	16
40	4	16

Fonte: Elaborado pelo autor.

Os displays mais comuns apresentam 16 colunas e duas linhas. Eles têm normalmente 14 pinos ou 16 pinos. Destes, oito pinos são destinados para dados ou instrução, seis são para controle e alimentação do periférico e dois para *backlight*. O *LED backlight* (iluminação de fundo) serve para facilitar as leituras durante a noite. Neste caso, a alimentação deste led faz-se normalmente pelos pinos 15 e 16, sendo o pino 15 para ligação ao anodo e o pino 16 para o catodo. A ferramenta SanUSB tem uma biblioteca em C para MPLABX C18 que suporta LCDs **16x2** e **16x4** e utiliza **somente o *nibble* superior do barramento de dados (D7, D6, D5 e D4)**, como é o caso da biblioteca LCD.h. A Figura 22.1 mostra a conexão do LCD à ferramenta SanUSB.

Figura 22. 1: Conexão do LCD no PIC.



Fonte: Elaborado pelo autor.

A Tabela 22.2 mostra um resumo das instruções mais usadas na comunicação com os módulos LCD.

Tabela 22. 2: Instruções mais comuns de LCD.

DESCRIÇÃO	MODO	RS	R/W	Código (Hex)
Display	Liga (sem cursor)	0	0	**0C**
	Desliga	0	0	**0A/ 08**
Limpa Display com Home cursor		0	0	**01**
Controle do Cursor	Liga	0	0	**0E**
	Desliga	0	0	**0C**
	Desloca para Esquerda	0	0	**10**
	Desloca para Direita	0	0	**14**
	Cursor Home	0	0	**02**
	Cursor Piscante	0	0	**0D**
	Cursor com Alternância	0	0	**0F**
Sentido de deslocamento	Para a esquerda	0	0	**04**
cursor ao entrar com caractere	Para a direita	0	0	**06**
Deslocamento da mensagem	Para a esquerda	0	0	**07**
ao entrar com caractere	Para a direita	0	0	**05**
Deslocamento da mensagem	**Para a esquerda**	0	0	**18**
sem entrada de caractere	**Para a direita**	0	0	**1C**
End. da primeira posição	primeira linha	0	0	**80**
	segunda linha	0	0	**C0**

Fonte: Elaborado pelo autor.

Para deslocar o conteúdo do LCD um caractere para a direita, utilize o comando **lcd_comando(instrução),** por exemplo, **lcd_comando (0x1C**) e para rolar o conteúdo do LCD um caractere para a esquerda, utilize o comando **lcd_comando (0x18).** A Tabela 22.3 mostra algumas instruções de comando da biblioteca LCD.h e o respectivo valor em decimal da instrução configurada em**lcd_comando()**:

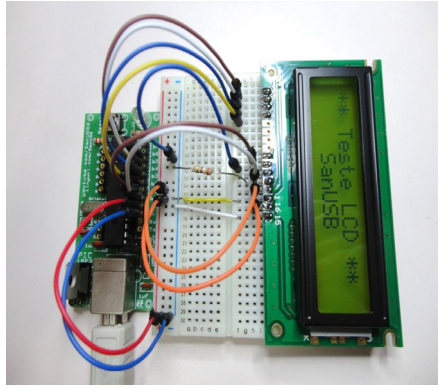
Tabela 22. 3: Instruções mais comuns de LCD.

LCD_FIRST_ROW	128
LCD_SECOND_ROW	192
LCD_THIRD_ROW	148
LCD_FOURTH_ROW	212
LCD_CLEAR	1
LCD_RETURN_HOME	12
LCD_UNDERLINE_ON	14
LCD_MOVE_CURSOR_LEFT	16
LCD_MOVE_CURSOR_RIGHT	20
LCD_TURN_OFF	0
LCD_TURN_ON	8
LCD_BLINK_CURSOR_ON	15
LCD_SHIFT_LEFT	24
LCD_SHIFT_RIGHT	28

Fonte: Elaborado pelo autor.

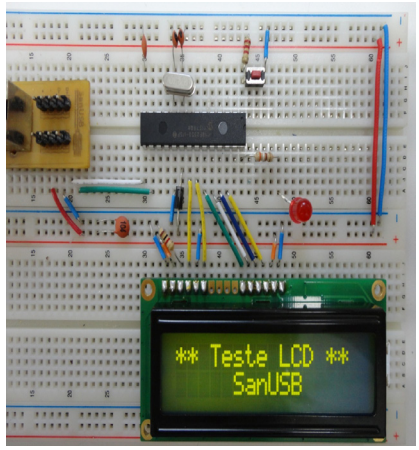
As Figuras 22.2 e 22.3 mostram a conexão do LCD às placa SanUSB.

Figura 22. 2: Conexão do LCD à placa SanUSB.



Fonte: Elaborado pelo autor.

Figura 22. 3: Prática Display LCD, montada em protoboard.



Fonte: Elaborado pelo autor.

A seguinte seqüência de comandos, gera o efeito de uma mensagem rolando no display. Para isso, será necessário declarar uma variável do tipo INT x.

Código em C para MPLABX: Ler conversor AD e Escrever em LCD:

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

unsigned int i;
unsigned char buffer1[20];

#pragma interrupt interrupcao
void interrupcao(){}

void main(void) {
clock_int_4MHz();
habilita_canal_AD(AN0);

lcd_ini();
lcd_comando(LCD_CLEAR);
lcd_comando(LCD_CURSOR_OFF);
tempo_ms(100);

lcd_escreve(1, 1, " **  Teste LCD  **");
tempo_ms(600);
lcd_escreve(2, 1, "SanUSB");
tempo_ms(500);
   while(1)
   {
   i= le_AD10bits(0);
sprintf(buffer1,"%d  ",i); //Imprime valor do potenciômetro de 0 a 1023
   lcd_escreve2(2, 12, buffer1); //com buffer
   tempo_ms(100);
}}
```

A posicição o cursor no LCDé configurada dentro da função **lcd_escreve(x, y, "nome");**, onde x e y são, respectivamente, a linha e a coluna onde o cursor deve ser reposicionado.

Desta forma, caso deseje escrever, por exemplo, a frase **Teste LCD** na primeira linha do display, sem apagar a segunda linha, basta inserir o comando **lcd_escreve(1, 1, " Teste LCD"); .** Isto irá posicionar o cursor na primeira linha, e primeira coluna.

STRING : É o trecho de caracteres delimitado por aspas duplas, que irá definir como será a seqüência de caracteres a ser gerada. Dentro das aspas, podem ser inseridos caracteres de texto, caracteres especiais e especificadores de formato.

No caso dos **caracteres especiais**, por não possuírem uma representação impressa, são compostos por uma barra invertida seguida de um símbolo, geralmente uma letra.

Exemplo de caracteres especiais : **\f** (limpar display), **\n** (nova linha), **\b** (voltar um caractere), **\r** (retorno de carro), **\g** (beep), etc...

Obs: alguns caracteres especiais somente resultarão efeito em terminais seriais.

Já os **especificadores de formato** são os locais, em meio ao texto, onde serão inseridas as variáveis que aparecerão após a STRING. Desta forma, estes especificadores devem obedecer algumas regras, de acordo com o tipo da variável a ser impressa (ver Tabela 22.4).

Tabela 22. 4: Especificadores de formato.

Tipo de variável	**Especificador de formato e exemplos de uso**	
int	**%u**	valor decimal (ex: 30)
	%x	valor em hexadecimal (ex: 1D)
	%3u	valor decimal alinhado com três dígitos (ex: _30)
	%03u	valor decimal alinhado 3 dígitos c/ zero (ex: 030)
signed int	**%i**	valor decimal com sinal. (ex: -2)
	%02i	decimal com sinal, 2 casas e zeros a esq. (ex: -02)
long **int32**	**%lu**	valor decimal (ex: 32345675);
	%05lu	valor decimal 5 casas c/ zeros a esquerda. (ex: 01000)
signed long **signed int32**	**%li**	valor decimal c/ sinal (ex: -500)
	%4li	valor decimal c/ sinal alinhado a esquerda (ex: -_500)
float	**%f**	valor real. Ex: (23.313451)
	%2.3f	valor real c/ 2 casas inteiras, 3 decimais. Ex: (23.313)
char	**%c**	caractere. Ex: (A)

Fonte: Elaborado pelo autor.

22.1 Exemplo: controle de tensão de uma solda capacitiva com LCD

O princípio de uma solda capacitiva acontece através da descarga instantânea de capacitores previamente carregados por dois terminais de solda em um ponto específico.

Este projeto consiste em realizar o controle de tensão de uma solda capacitiva em

baixo custo, através de um sistema microcontrolado utilizando o PIC18F2550. Para a leitura da tensão CC nos terminais da solda capacitiva, na ordem de 300V, é necessário inicialmente utilizar um divisor de tensão para adequação à tensão máxima do conversor AD do microcontrolador de 5V. Esta relação do divisor é compensada via software, multiplicando o valor de tensão lido pela mesma relação de divisão. Os valores de tensão real e tensão de referência na ordem de 270V, que pode ser incrementada ou decrementada por dois botões de ajuste, são mostrados em um display LCD. A última tensão de referência ajustada é guardada na memória. Dessa forma, quando o sistema é reiniciado a tensão de referência assume o último valor ajustado.

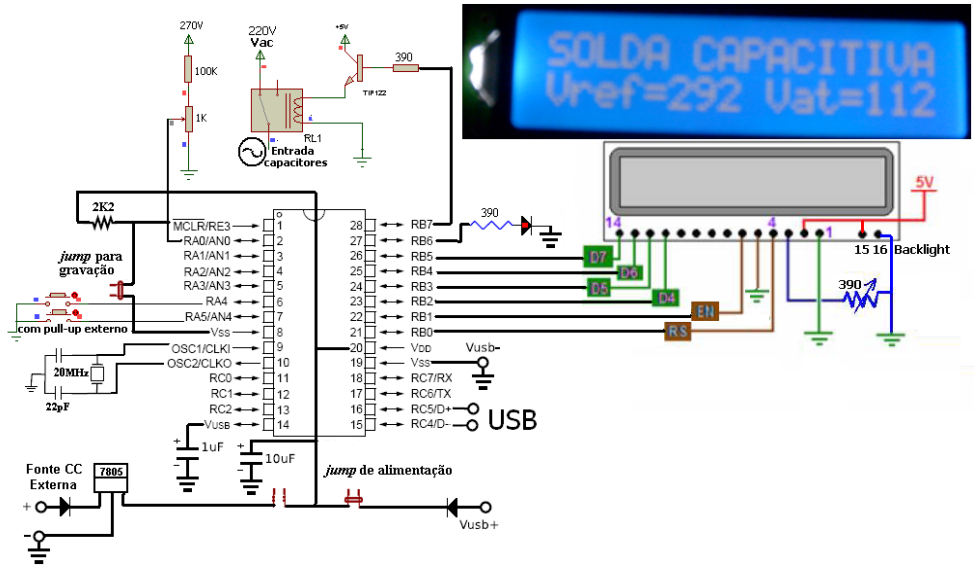
Quando a tensão real e a de referência são iguais, a alimentação de 220V do circuito de potência é cortada pela abertura de um relé NF (normalmente fechado) e um LED de atuação ascende indicando que a tensão de referência foi atingida. O LED de atuação indica a presença ou não de tensão residual nos capacitores de carga e apaga somente após a descarga de tensão nos terminais de solda, o que contribui para evitar descargas de tensão nos operadores durante o manuseio da solda.

Para regular esse sistema embarcado é necessário medir a tensão nos terminais da solda capacitiva com o multímetro e igualar com o valor atual indicado no LCD através do potenciômetro de ajuste do divisor de tensão. O circuito do sistema de controle de tensão e a foto do LCD após a montagem em *protoboard* indicando a tensão de referência para desligamento (Vref) e a tensão atual (Vat) podem ser vistos na Figura 22.4. O código exemplo está a seguir:

```
#include "SanUSB1.h"
#include "lcd.h"        //RB0-RS, RB1-EN, RB2-D4, RB3-D5, RB4-D6, RB5-D7
#pragma interrupt interrupcao
void interrupcao(){}
#define botaoinc pin_a4
#define botaodec pin_a5
#define rele pin_b7
#define ledrele pin_b6
unsigned int vref=270, guardavref, constante=100;
unsigned long int vatual, valorAD;//Deve ser de 32 bits devido ao cálculo do AD que esoura 65536
unsigned char baixovref, altovref; // Como vref> 256 guardar o valor em 2 bytes, posições 10 e 11  //da
EEPROM interna
short int flag1, flag2;
    void main() {
        clock_int_4MHz();
        lcd_ini(); // Configuração inicial do LCD
        nivel_baixo(rele);
        nivel_baixo(ledrele);
        guardavref=(256*le_eeprom(10))+le_eeprom(11)+1; //+1 para compensar um bug de //decremento no
reinício
        if (guardavref>=100 && guardavref<=500) {vref=guardavref;} // Resgata o último valor de //referência
adotado

  setup_ADC_ports (AN0); //(Selecao_dos_pinos_analogicos)
  setup_adc(ADC_CLOCK_INTERNAL ); //(Modo_de_funcionamento)
  set_adc_channel(0); //(Qual_canal_vai_converter)
  tempo_ms(10);
  printf(lcd_escreve,"SOLDA CAPACITIVA");
    while (1) {
```

```
//*********************************BOTÕES*************************************
if (!input(botaoinc)) {flag1=1;}
if (flag1==1 && input(botaoinc) ) {flag1=0;++vref;
//se o botão foi pressionado (flag1==1) e se o botão já foi solto (input(botao)) incremente vref
altovref=vref/256; baixovref=vref%256;
write_eeprom(10,altovref); write_eeprom(11,baixovref);    }
// Como Vref>256, guarde o valor de vref nas posicões 10  e 11 da eeprom interna
if (!input(botaodec)) {flag2=1;}
if (flag2==1 && input(botaodec) ) {flag2=0;--vref;
//se o botão foi pressionado (flag2==1) e se o botão já foi solto (input(botao)) decremente vref
altovref=vref/256; baixovref=vref%256;
write_eeprom(10,altovref); write_eeprom(11,baixovref);    }
// guarde o valor na de vref nas posicões 10  e 11 da eeprom interna
//*****************************************************************************
if (vatual>=vref) {nivel_alto(rele); nivel_alto(ledrele); } //Abre o relé, avisa com led
if (vatual<=20) {nivel_baixo(rele); nivel_baixo(ledrele);} //Só desliga depois da descarga

//*****************************************************************************
valorAD = read_adc(); // efetua a conversão A/D
vatual=((constante*5*valorAD)/1023);     //Regra de três:          5        -------  1023
//                Tensão real (mV) --------  ValorAD
lcd_pos_xy(1,2);
printf(lcd_escreve,"Vref=%lu Vat=%lu   ",vref, vatual);
tempo_ms(300);                                                    }}
```

Figura 22. 4: Exemplo de aplicação do LCD.



Fonte: Elaborado pelo autor.

Capítulo 23
LDR

LDR significa *LightDependent Resistor,* ou seja, Resistor Variável Conforme Incidência de Luz. Esse resistor varia sua resistência conforme a intensidade de radiação eletromagnética do espectro visível que incide sobre ele.

Um LDR é um transdutor de entrada (sensor) que converte a (luz) em valores de resistência. É feito de sulfeto de cádmio (CdS) ou seleneto de cádmio (CdSe). Sua resistência diminui quando a luz é intensa, e quando a luz é baixa, a resistência no LDR aumenta.

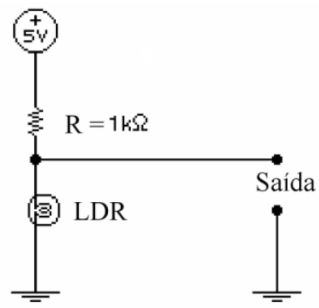
Um multímetro pode ser usado para encontrar a resistência na escuridão (geralmente acima de 1MΩ) ou na presença de luz intensa (aproximadamente 100Ω).

O LDR é muito frequentemente utilizado nas chamadas fotocélulas que controlam o acendimento de poste de iluminação e luzes em residências. Também é utilizado em sensores foto-elétricos.

23.1 Exemplo: Modelagem de um Luxímetro Microcontrolado com LDR

Este luxímetro tem em seu circuito sensor um LDR, um resistor divisor de tensão e uma fonte de tens*ão estabilizada, como mostra a F*igura 23.1.

Figura 23. 1: Circuito sensor com LDR.



Fonte: Elaborado pelo autor.

Medições da tensão de saída foram feitas e colocadas em uma tabela juntamente com as iluminâncias medidas por um luxímetro comercial faixa de 1 a 50000 Lux. Os valores encontrados são vistos na tabela abaixo. Os valores em negrito foram considerados como limite de cada equação da reta. A correspondência entre a tensão da saída e a iluminância podem ser vistos na Tabela 23.1.

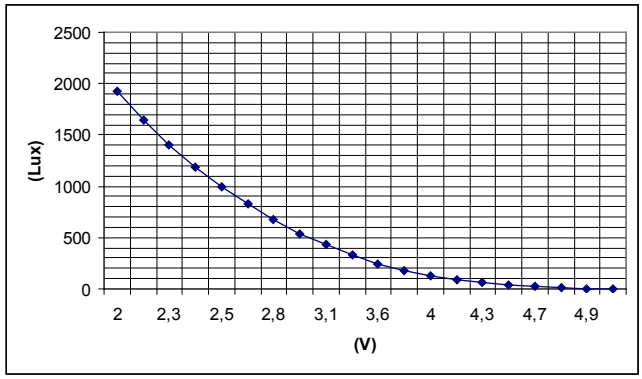
Tabela 23. 1: Relação entre lux e tensão.

Lux	2	5	12	20	36	60	94	130	**180**
Volt	4,9	4,9	4,8	4,7	4,5	4,3	4,1	4	**3,8**
Lux	338	530	**674**	827	1000	1183	1404	1651	1923
Volt	3,3	3	**2,8**	2,7	2,5	2,4	2,3	2,1	2

Fonte: Elaborado pelo autor.

Com base na Tabela 23.1, foi construído o gráfico da Figura 23.2.

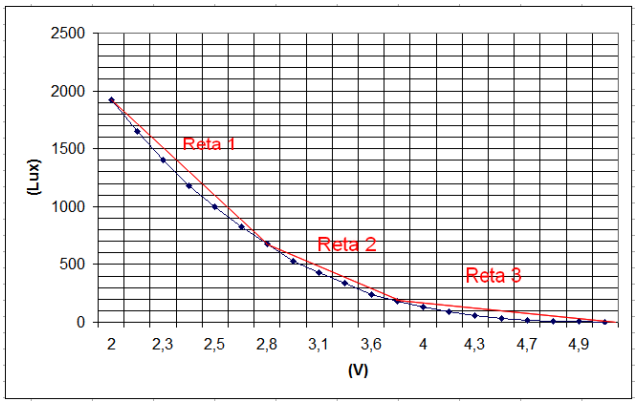
Figura 23. 2: Gráfico Lux x Volt.



Fonte: Elaborado pelo autor.

Para simplificar o programa do PIC, foi modelado a curva do gráfico acima, dividindo-a em três retas como mostra a Figura 23.3.

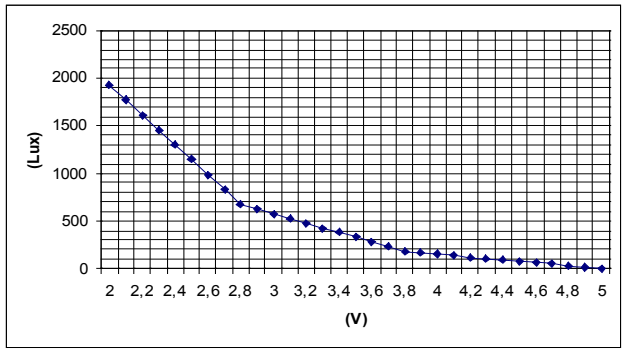
Figura 23. 3: Modelagem matemática dos valores obtidos.



Fonte: Elaborado pelo autor.

O programa funciona da seguinte maneira: lê o conversor A/D e multiplica esse valor por sua resolução (no caso de um converso AD de 10 bits, a resolução é de aproximadamente 5 mV), encontrando então a tensão (V), depois são feitas 3 comparações (IF) para saber qual das três equações acima deve ser utilizada para calcular a iluminância (Lux). A figura abaixo mostra o novo gráfico lux versus volts. O gráfico da equação linear geral de cada reta é mostradona Figura 23.4.

Figura 23. 4: Modelagem matemática dos valores obtidos.



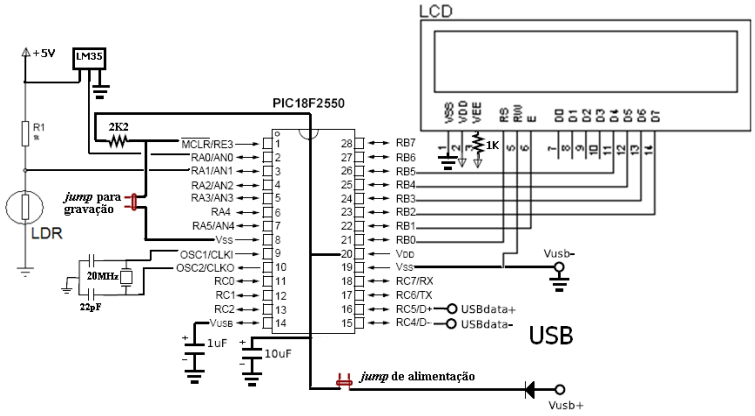
Fonte: Elaborado pelo autor.

23.2 Supervisório

Esta interface foi desenvolvida utilizando ambiente de programação gráfica e emulação via USB de um canal serial COM virtual.

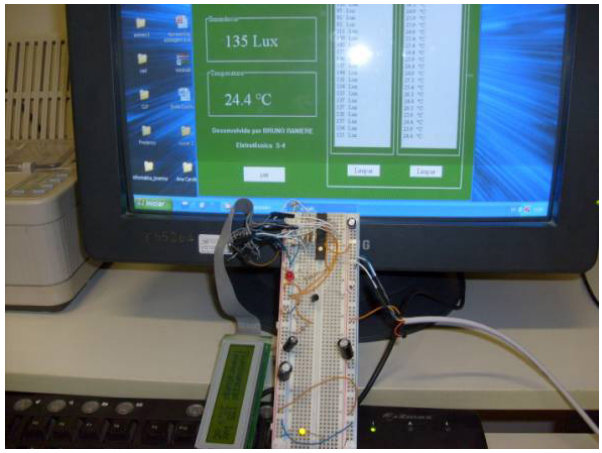
É visto na Figura 23.5, o esquema circuito eletrônico montado e a na Figura 23.6, a foto do circuito montado em operação. No final do trabalho é mostrado o programa completo para ler a iluminância no canal AD 1 e a temperatura do ambiente com um LM35 no canal AD 0.

Figura 23. 5: Esquema eletrônico do circuito luxímetro.



Fonte: Elaborado pelo autor.

Figura 23. 6: Foto do circuito montado.



Fonte: Elaborado pelo autor.

O luxímetro mostrado neste trabalho apresenta como uma solução de baixo custo para aplicações onde não é necessário haver uma grande precisão nas medições. O método do modelagem de curva pode ser aplicado em várias ocasiões onde não se sabe a equação que gerou o gráfico proposto. Isto ratifica a versatilidade de sistemas microcontrolados.

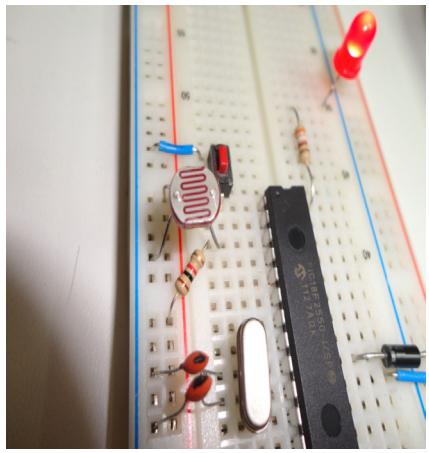
```
// Programa Luxímetro digital + termômetro digital c/ comunicação via USB//
 ------------------------------------------------------------------------------------------------------------
#include “SanUSB1.h”
float tens,lux,temp;

#pragma interrupt interrupcao
void interrupcao(){}

void main()
{
    clock_int_4MHz();
    lcd_ini();
    usb_cdc_init(); // Inicializa o protocolo CDC
    usb_init(); // Inicializa o protocolo USB
    usb_task(); // Une o periférico com a usb do PC
    //while(!usb_cdc_connected()) {} // espera o protocolo CDC se conectar com o driver CDC
    //usb_wait_for_enumeration(); //espera até que a USB do Pic seja reconhecida pelo PC

    setup_adc_ports(AN0_TO_AN1);
    setup_adc(ADC_CLOCK_INTERNAL);
    nivel_baixo(pin_b6);
    printf (lcd_escreve," \f ");

      while(1)
        {
          set_adc_channel(1);
          tempo_ms(20);
```

```
tens=5*(float)read_adc()/1023;
if (tens>2 && tens<2.8)  {     lux=(3936.4-(1249*tens))/0.8;  }

if (tens>=2.8 && tens<=3.8)  {     lux=2057.2-494*tens;  }

if (tens>3.8)  {     lux=(900-180*tens)/1.2;  }

if (tens>2)  {  //Leitura válida
   lcd_pos_xy(1,1);
   printf ("%.0f",lux);
   tempo_ms(50);
   printf ("L");
   printf (lcd_escreve,"Iluminancia: %.0f lux     ",lux );
   lcd_envia_byte(0,0x0C); //Apaga o cursor
}

if (tens<=2)       //Leitura não válida
{
   lcd_pos_xy(1,1);
   printf ("Erro");
   tempo_ms(50);
   printf ("L");
   printf (lcd_escreve,"valor fora da faixa!  ");
   lcd_envia_byte(0,0x0C); //Apaga o cursor
}
    tempo_ms(30);
    set_adc_channel(0);
    tempo_ms(20);
    temp=500*(float)read_adc()/1023;
    lcd_pos_xy(1,2);
    printf ("%.1f",temp);
```

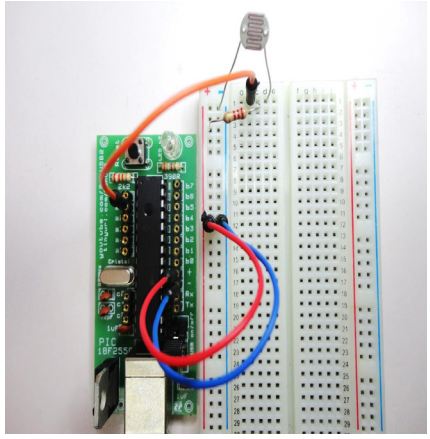
```
tempo_ms(50);
    printf ("T");
    printf (lcd_escreve,"Temperatura: %.1f oC  ",temp);
    lcd_envia_byte(0,0x0C); //Apaga o cursor
     tempo_ms(800);
    nivel_alto(pin_b6);
    tempo_ms(200);
    nivel_baixo(pin_b6);         }
}
```

As Figura 23.7, 23.8 mostram a ligação do LDR à ferramenta SanUSB.

Figura 23. 7: Esquemático Prática Sensor luminosidade LDR em protoboard.



Fonte: Elaborado pelo autor.
Figura 23. 8: Sensor de luminosidade LDR com placa SanUSB.



Fonte: Elaborado pelo autor.

Capítulo 24
Interface I2C

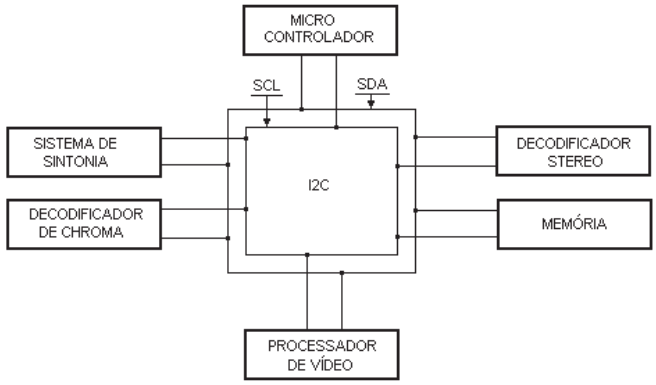
I²C significa Inter-IC (*Integrated Circuit*). Este barramento serial foi desenvolvido pela Philips como o objetivo de conectar CIs e periféricos de diferentes fabricantes em um mesmo circuito, como microcontroladores, memórias externas e relógio em tempo real, usando o menor número de pinos possível. Este protocolo serial necessita somente de duas linhas: uma linha serial de dados (SDA) e uma de clock (SCL). Quando o baramento não está em uso, as duas linhas ficam em nivel lógico alto forçadas pelos resistores de *pull-up*.

O barramento serial, com transferência de 8 bits por vez, possibilita comunicação bidirecional com velocidade de 100 Kbps no modo Padrão, 400 Kbps no modo Fast, ou até 3,4 Mbits/s no modo High-speed.

Esta interface apresenta a filosofia *multi-master* onde todo CI da rede pode transmitir ou receber um dado, e o transmissor gera o seu próprio clock de transmissão. O número máximo de CIs que podem ser conectados é limitado apenas pela capacitância máxima do barramento de 400pF.

Um exemplo típico de configuração I²C em TVs é mostrado na Figura 24.1.

Figura 24. 1: Configuração I²C em TVs.

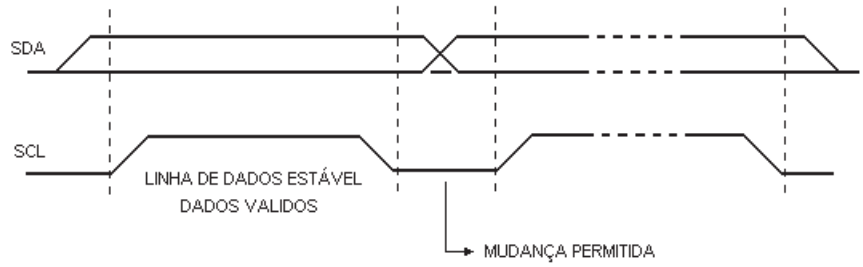


Fonte: Elaborado pelo autor.

24.1 Regras para transferência de dados

Cada bit da linha de dados só é lido quando o nivel da linha de clock está em nivel alto (ver figura 24.2).

Figura 24. 2: Leitura de dados em comunicação.

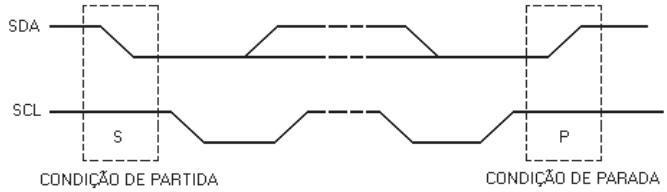


Fonte: Elaborado pelo autor.

As condições de partida e parada de transmissão são sempre geradas pelo MASTER. O barramento é considerado como ocupado após a condição de partida, e livre um certo período de tempo após a condição de parada.

Uma transição de ***H para L da linha SDA*** (*start bit*) durante o tempo em que a linha SCL permanece em H, ou seja, um dado válido, é definido como **condição de partida** e uma transição de ***L para H da linha SDA***(*stop bit*) durante o período H da linha SCL, define uma **condição de parada**(ver Figura 24.3).

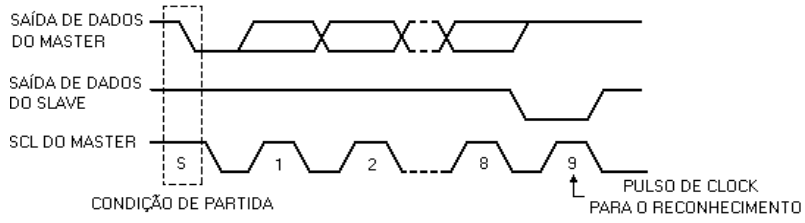
Figura 24. 3: Comandos de início e fim de comunicação.



Fonte: Elaborado pelo autor.

Cada byte é acompanhado de um bit de reconhecimento obrigatório. O reconhecimento é gerado pelo MASTER no ***décimo bit*** liberando a linha SDA (nivel alto) durante a ocorrência do pulso de clock de reconhecimento. Por sua vez, o CI receptor (SLAVE) é obrigado a levar a linha SDA a nivel baixo durante o período H do clock de reconhecimento, conforme Figura 24.4.

Figura 24. 4: Reconhecimento do byte.



Fonte: Elaborado pelo autor.

Se o SLAVE reconhecer o endereço, mas depois de algum tempo na transferência não receber mais nenhum byte de dados, o MASTER deverá abortar a transferência. Esta condição é indicada pelo SLAVE, devido à não geração do reconhecimento logo após a recepção do primeiro byte de dados. O SLAVE deixa a linha de dados em nivel H e o MASTER gera a condição de parada.

Caso haja uma interrupção interna no SLAVE durante a transmissão, ele deverá levar também a linha de clock SCL a nivel L, forçando o MASTER a entrar em um modo de espera.

Para escrever um dado nos escravos é necessário enviar um byte de endereço do escravo, onde os **4 bits mais significativos** identificam o tipo de escravo (por exemplo, memórias EEPROM é **1010 ou 0xa0** e RTC é **1101 ou 0xd0** (com exceção do RTC PCF8583 cujo endereço também é **0xa0**). Os **3 bits intermediários** especificam de um até 8 dispositivos, que são discriminados nos pinos de endereço de cada escravo, e o **bit menos significativo R/W** indica se a operação é de leitura (1) ou escrita (0). Após isso, deve-se enviar uma palavra de 8 ou16 bits de endereço onde se quer escrever e depois o dado. No final do pacote uma condição de parada (*i²c_stop*).

Função da biblioteca I2C que descreve essa operação de escrita em memória EEPROM:

```
void escreve_eeprom(byte dispositivo, long endereco, byte dado)
// Escreve um dado em um endereço do dispositivo
// dispositivo - é o endereço do dispositivo escravo (0 - 7)
// endereco - é o endereço da memória a ser escrito
// dado - é a informação a ser armazenada
{
        if (dispositivo>7) dispositivo = 7;
i2c_start();
i2c_escreve_byte(0xa0 | (dispositivo << 1)); // endereça o dispositivo livrando o LSB que é o R\W
i2c_le_ack();                       // Lê reconhecimento do escravo
i2c_escreve_byte(endereco >> 8);    // parte alta do endereço de 16 bits
i2c_le_ack();
i2c_escreve_byte(endereco);   // parte baixa do endereço de 16 bits
i2c_le_ack();
i2c_escreve_byte(dado);             // dado a ser escrito
i2c_le_ack();
i2c_stop();
tempo_ms(10); // aguarda a programação da memória
```

Para a operação de leitura de um escravo é necessário um *start* repetido e no final do pacote um sinal de não-reconhecimento (*nack*) e uma condição de parada (*i2c_stop*).

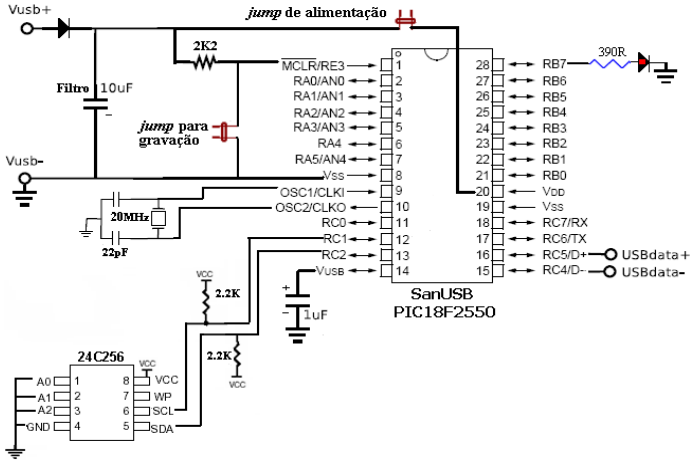
A Função da biblioteca I2C que descreve este protocolo de operação de leitura de memória EEPROM é a seguinte:

```
byte le_eeprom(byte dispositivo, long int endereco)
// Lê um dado de um endereço especificado no dispositivo
// dispositivo - é o endereço do dispositivo escravo (0 - 7)
// endereco - é o endereço da memória a ser escrito
{
byte dado;
if (dispositivo>7) dispositivo = 7;
i2c_start();
i2c_escreve_byte(0xa0 | (dispositivo << 1)); // endereça o dispositivo
i2c_le_ack();
i2c_escreve_byte((endereco >> 8));       // envia a parte alta do endereço de 16 bits
i2c_le_ack();
i2c_escreve_byte(endereco);     // envia a parte baixa do endereço de 16 bits
i2c_le_ack();
i2c_start();                              //repetido start
// envia comando para o escravo enviar o dado
i2c_escreve_byte(0xa1 | (dispositivo << 1)); endereça o dispositivo e colocando em leitura
0xa1
i2c_le_ack();
dado = i2c_le_byte()                    // lê o dado
i2c_nack();
i2c_stop();
return dado;
```

Capítulo 25
Memória EEPROM externa I²C

Para sistemas embarcados em que são necessários a aquisição de dados de mais de 256 bytes (capacidade da EEPROM interna dos microcontroladores), é necessária a utilização de memórias EEPROM externals. Os modelos mais comuns são o 24LC e 24C256 (256 Kbits que corresponde a 32Kbytes). Estas memórias possuem oito pinos e apresentam, entre outras características, interface de comunicação I2C. A Figura 25.1 mostra o circuito simples de uma EEPROM I2C ligada nn ferramenta SanUSB.

Figura 25. 1: Uso de memória EEPROM externa via I²C.



Fonte: Elaborado pelo autor.

O programa abaixo mostra o armazenamento de valores digital de tensão de 0 a 5000mV de um potenciômetro, a cada segundo, em um buffer (região de memória circular) de 150 registros de 16 bits na memória EEPROM externa, ou seja, 300 bytes, que é mostrado via comuncação somente quando o botão da placa SanUSB é pressionado.

```
#include "SanUSB48.h" // Firmware para configuração e leitura por hardware de EEPROM i2c
#include"i2c_usb.h"//Biblioteca de funcoes I2C com a placa SanUSB, onde RB0(SDA) e RB1(SCL)

unsigned char valor,valorbcd, endereco, numquant=0, temp=0;
unsigned char  comando[6], n=0, m=0;
short int flagA4=0, flagA5=0;
```

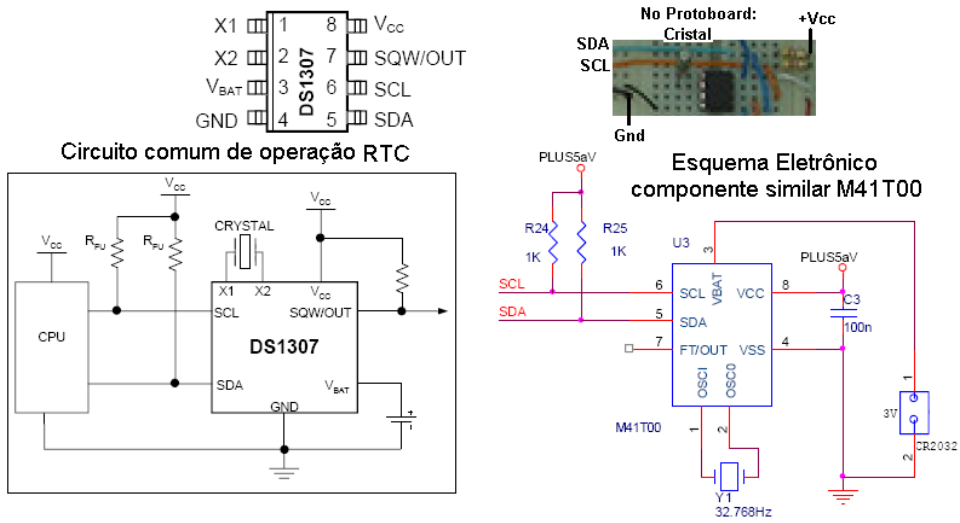
```
unsigned long int resultado, tensao_lida16;
unsigned int i,j,endereco16=0, posicao=0, valorgravado;
unsigned char byte1,byte2; // 2 Partes do valor da tensao_lida16

unsigned int conv_dec_2bytes(unsigned int valor16)
{//Função auxiliar para alocar um valor de 16 bits (até 65535) em 2 bytes
   byte1= valor16%256; byte2 = valor16/256; //o que for resto (%) é menos significativo
   return(byte2,byte1);
}
#pragma interrupt interrupcao
void interrupcao(){
if (serial_interrompeu)  {
    serial_interrompeu=0;
    comando[n] = le_serial();
    if (comando[n]==79) {flagb=1;}
       ++n; if(n>=5){n=0;}
}
}
void main(){

   clock_int_48MHz();
habilita_interrupcao(recep_serial);
   habilita_canal_AD(AN0);
   taxa_serial(9600);
   i2c_ini();

   while(1){
      resultado = le_AD10bits(0);
      tensao_lida16 = (resultado * 5000)/1023; //Valor até 16 bits (2 bytes)
      sendnum(tensao_lida16); swputc(' ');

      //*************************************************************************
      conv_dec_2bytes(tensao_lida16);
      posicao=2*endereco16; //endereço é o ponteiro de 16 bits (byte 1 e byte 2)
      escreve_ieeprom( posicao, byte2); //Byte mais significativo do int16
      escreve_ieeprom( posicao+1, byte1 ); //byte menos significativo do int16
      ++endereco16; if (endereco16>=150){endereco16=0;} //Buffer de 300 bytes posicao<300
      //*************************************************************************/
      if(entrada_pin_e3==0){
          send_hex(le_eeprom(5)); swputc(' ');

//**************LEITURA DO BUFFER DA EEPROM EXTERNA I2C*******************
             for(i=0; i<10; ++i) { //150 Valores de 16 bits ou 300 de 8 bits.
                for(j=0; j<15; ++j) {
                   valorgravado= 256*le_ieeprom((i*30)+2*j) + le_ieeprom((i*30)+2*j+1);
                   sendnum(valorgravado); swputc(' ');
                            }
                   sendrw((rom char *)"\n\r");
                            }
                   sendrw((rom char *)"\n\r");
   //*************************************************************************/
```

```
                        }
      ++i; if(i>255) {i=0;}
      escreve_eeprom(5, i); //Escreve na posicao 1000 para depurar a leitura da eeprom
      inverte_saida(pin_b7);
      tempo_ms(1000);
         }
}
```

Capítulo 26
RTC (relógio em tempo real)

O *Real Time Clock I²C* DS1307 (ver Figura 26.1) é um relógio/calendário serial de baixo custo controlado por um cristal externo de 32.768 Hz. A comunicação com o DS1307 é através de interface serial I²C (SCL e SDA). Oito bytes de RAM do RTC são usados para função relógio/calendário e são configurados na forma Binary Coded Decimal – BCD. É possível a retenção dos dados na falta de energia utilizando **uma bateria de lítio de 3V - 500mA/h** conectada ao pino 3.

Figura 26. 1: RTC DS1307 e similar.



Fonte: Elaborado pelo autor.

Para representar números decimais em formato binário, o relógio DS1307, bem como calculadoras e computadores utilizam o código BCD, que incrementa a parte alta do byte hexadecimal quando o número da parte baixa é maior que 9. Isto é possível somando 6 (0110b) ao resultado maior que 9. Este código facilita a transmissão de dados e a compreensão do tempo, tendo em vista que em formato hexadecimal, apresenta o valor em decimal.

Para transformar decimal em BCD, é possível dividir o número binário (byte) por 10 e colocar o resultado isolado das dezenas no nibble alto do byte BCD e o resto, ou seja, as

unidades, no nibble baixo do byte BCD.

Para iniciar o relógio DS1307, após o *power-on*, é necessário incrementar os segundos quando estiverem todos os registros da RAM em zero. A bateria GP 3.6V garante o funcionamento do relógio e também o processamento do PIC. Testes indicaram que a bateria suportou o processamento e incremento automático do relógio por cerca de sete horas sem alimentação externa.

O firmware para configuração e leitura por hardware do relógio RTC DS1307 (BCD) e gravação na EEPROM externa i2c está descrito abaixo.

```
#include "SanUSB48.h" // Firmware para configuração e leitura por hardware de EEPROM i2c e de relógio DS1307 (BCD)
#include"i2c_usb.h"//Biblioteca de funcoes I2C com a placa SanUSB, onde RB0(SDA) e RB1(SCL)

unsigned char valor,valorbcd, endereco, numquant=0, temp=0;
unsigned char  comando[6], n=0, m=0;
short int flagA4=0, flagA5=0;
unsigned long int resultado, tensao_lida16;

unsigned int i,j,endereco16=0, posicao=0, valorgravado;
unsigned char byte1,byte2,xis; // 2 Partes do valor da tensao_lida16

unsigned int conv_dec_2bytes(unsigned int valor16)
{//Função auxiliar para alocar um valor de 16 bits (até 65535) em 2 bytes
   byte1= valor16%256; byte2 = valor16/256; //o que for resto (%) é menos significativo
   return(byte2,byte1);
}

#pragma interrupt interrupcao
void interrupcao(){
if (serial_interrompeu)  {
    serial_interrompeu=0;
    comando[n] = le_serial();
    if (comando[n]==79) {flagb=1;}
    //////////FUNCAO 4: CONFIGURA RELÓGIO////////////////////Ex: Digitar A4H09 (Hora = 09) , A4D15 (Dia = 15).
    if (comando[n]=='A'){n=0;comando[0] = 'A';} //UTILIZAR VALORES DECIMAIS EM DOIS DIGITOS. ex:06, 23, 15, etc.

       if ( comando[1]== '4' && comando[2]== 'H' && n==2) { endereco=2;} //Escreve o endereco das horas
       if ( comando[1]== '4' && comando[2]== 'M' && n==2) { endereco=1;} //Escreve o endereco dos minutos
       if ( comando[1]== '4' && comando[2]== 'S' && n==2) { endereco=0;} //Escreve o endereco dos segundos
       if ( comando[1]== '4' && comando[2]== 'D' && n==2) { endereco=4;} //Escreve o endereco do dia
       if ( comando[1]== '4' && comando[2]== 'N' && n==2) { endereco=5;} //Escreve o endereco do mes
       if ( comando[1]== '4' && comando[2]== 'Y' && n==2) { endereco=6;} //Escreve o endereco do ano

       if ( comando[1]== '4' && comando[3]>='0'&&comando[3]<='9'&& n==3)
```

```
 {numquant=(comando[3]-0x30);}
       if ( comando[1]== '4' && comando[4]>='0'&&comando[4]<='9'&& n==4)
{numquant=numquant*10+(comando[4]-0x30);
                                                    flagA4=1;
                                                    }

    //////////////*////////FUNCAO 5: LÊ RELÓGIO//////////////////////////Ex: A5- Lê o relógio e o
calendário
       if (comando[1]== '5' && n==1){flagA5=1;}

++n; if(n>=5){n=0;}
  }
}
void main(){

clock_int_48MHz();
   habilita_interrupcao(recep_serial);
   habilita_canal_AD(AN0);
   taxa_serial(9600);
   i2c_ini();
   while(1){

      if (flagA4){ flagA4=0; //Comandos A4 para Configurar o RTC
            escreve_rtc(endereco, dec_para_bcd(numquant)); //Escrever em BCD no RTC
            send_hex(le_rtc(hora)); swputc (':');      //Envia resultado via serial por bluetooth ou
qualquer outro modem.
            send_hex(le_rtc(min)); swputc (':');     //Exemplo de resposta: 18:49:37 19/04/14
send_hex(le_rtc(seg)); swputc (' ');
send_hex(le_rtc(dia)); swputc ('/');
send_hex(le_rtc(mes)); swputc ('/');
send_hex(le_rtc(ano)); swputc(' ');
               }
      if (flagA5){ flagA5=0; //BCD em hexadecimal representa o decimal
            send_hex(le_rtc(hora)); swputc (':');
send_hex(le_rtc(min)); swputc (':');
            send_hex(le_rtc(seg)); swputc (' ');
send_hex(le_rtc(dia)); swputc ('/');
send_hex(le_rtc(mes)); swputc ('/');
            send_hex(le_rtc(ano)); swputc(' ');
               }

      resultado = le_AD10bits(0);
tensao_lida16 = (resultado * 5000)/1023; //Valor até 16 bits (2 bytes)
      sendnum(tensao_lida16); swputc(' ');

      //*************************************************************************
      conv_dec_2bytes(tensao_lida16);
      posicao=2*endereco16; //endereço é o ponteiro de 16 bits (byte 1 e byte 2)
      escreve_ieeprom( posicao, byte2); //Byte mais significativo do int16
      escreve_ieeprom( posicao+1, byte1 ); //byte menos significativo do int16
      ++endereco16; if (endereco16>=150){endereco16=0;} //Buffer de 300 bytes posicao<300
      //*************************************************************************/

      if(entrada_pin_e3==0){
     sendnum(le_eeprom(5)); swputc(' ');

       //**************LEITURA DO BUFFER DA EEPROM EXTERNA I2C*******************
            for(i=0; i<10; ++i) { //150 Valores de 16 bits ou 300 de 8 bits.
               for(j=0; j<15; ++j) {
```

```
                valorgravado= 256*le_ieeprom((i*30)+2*j) + le_ieeprom((i*30)+2*j+1);
                sendnum(valorgravado); swputc(' ');
                    }
                sendrw((rom char *)"\n\r");
                }
                sendrw((rom char *)"\n\r");
//*****************************************************************************/
                    }
  //  ++i; if(i>255) {i=0;}
  //   escreve_eeprom(5, i);
     inverte_saida(pin_b7);
     tempo_ms(1000);
        }}
```

O protocolo de comunicação com o RTC foi idealizado para ser simples, de fácil implementação e para possibilitar baixa probabilidade de erros. Após o endereço do sistema (A), da função desejada e da posição de memória, o usuário, ou o software de monitoramento, deve inserir os dígitos X (0 a 9) necessários para as funções mostradas na Tabela 26.1.

Tabela 26. 1: Funções para manipulação e verificação serial dos dispositivos do sistema de aquisição de dados.

Endereço da placa	Função	Posição memória	Valor	Resultados EEPROM externa e RTC
A	4	S (Segundo) M (Minuto) H (Hora) D (Dia) N (Mês) Y (Ano)	XX	Escrita na variável do relógio RTC com o valor XX
A	5	-	-	Leitura das variáveis do relógio RTC

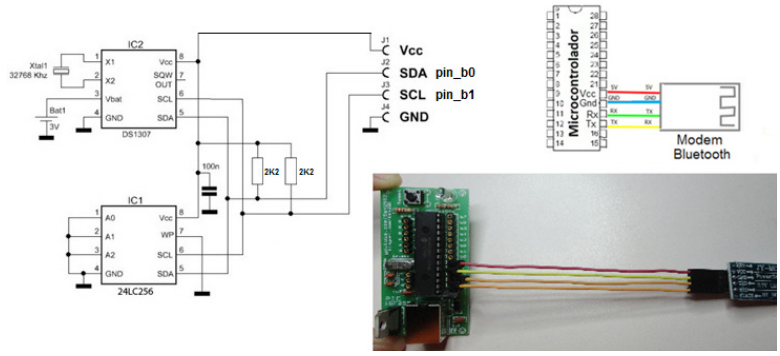
Fonte: Elaborado pelo autor.

Capítulo 27
Protótipo de sistema básico de aquisição de dados

Em muitos sistemas de aquisição de dados e controle é necessária a medida de algumas grandezas físicas, como exemplo, temperatura, pressão e velocidade, entre outras. Tais grandezas são inerentes a alguns fenômenos físicos e, em geral, sua natureza é analógica. Isto é, trata-se de variáveis que assumem valores contínuos e reais, diferentes de sinais digitais que são descontínuos e expressados segundo representação binária. Comumente quando as saídas analógicas dos sensores são processadas por sistemas digitais, há a necessidade do condicionamento do sinal para que os sinais provenientes dos sensores sejam adequados às características de um conversor AD. Assim, com o uso de um microcontrolador dotado de um conversor interno AD para aquisição de dados, o valor analógico convertido para digital é processado pelo software de controle de acordo com decisões lógicas baseadas em comparações ou em operações matemáticas.

A bateria em paralelo com a fonte de alimentação tem uma grande relevância neste projeto de aquisição de dados. Além de evitar *reset* por queda de tensão, ela permite a mudança da fonte de alimentação da USB para a fonte externa sem desconexão do sistema. Ver exemplo na Figura 27.1.

Figura 27. 1: Sistema de aquisição de dados.



Fonte: Elaborado pelo autor.

Este sistema de aquisição de dados USB é *Dual Clock*, ou seja, utiliza duas fontes de clock, uma para o canal USB de 48MHz, proveniente do oscilador externo de 20MHz, e

outra para o processador na execução do protocolo i2c, proveniente do oscilador RC interno de 4 MHz. (#byte OSCCON=0XFD3 //Aponta o registro do oscilador interno para configuração de 4MHz na função Main -> OSCCON=0B01100110;).

```
////////////////////////////////////////////////////////////////////
//// Este programa utiliza duas fontes de clock, uma para o canal USB////
//// de 48MHz proveniente do oscilador externo de 20MHz e outra para ////
//// o processador na execução do protocolo i2c, proveniente do      ////
//// oscilador interno 4 de MHz/////////////////////////////////////////
//// O Watch Dog Timer (WDT) protege contra travamento do programa  /////
/////////////////////////Cabeçalho Padrão//////////////////////////////

#include “SanUSB1.h”
//#device ADC=8
#include ".\include\usb_san_cdc.h"// Biblioteca para comunicação serial
#include <i2c_dll16sanc.c>
```

```
char escravo,funcao,sensor,endrtc,
valorrtc1,valorrtc2,posmeme1,posmeme2,posmeme3,posmeml1,posmeml2,posmeml3,posqua
nt1,posquant2;
unsigned int  ender, endereco, val, valor,valorbcd;
unsigned int mult=2,end=0, reg, numquant;
unsigned                                                                    int16
hora,horadec,minuto,minutodec,segundo,segundodec,dia,diadec,mes,mesdec,ano,anodec;
unsigned int16 i, j,numpose, numposl,num16,endpromext,k,puloext,bufferdia;
int8 regi[2];
boolean led,ledint,flagwrite;

/*****************************************************************
* Conversão BCD P/ DECIMAL
*****************************************************************/
int bcd_to_dec(int valorb)
{
int temp;
temp = (valorb & 0b00001111);
temp = (temp) + ((valorb >> 4) * 10);
return(temp);
}
/*****************************************************************
* Conversão DECIMAL p/ BCD
*****************************************************************/
int dec_para_bcd(unsigned int valord)
{

return((0x10*(valord/10))+(valord%10));//Coloca a parte alta da divisão por 10 no nibble mais
significativo
}
////////////////////////////////////////////////////////////////////////////
#int_timer1
```

```
void trata_t1 ()
{--mult;
if (!mult)
{mult=2; // 2 *(48MHz/4MHz) - 4 seg
hora=le_rtc(2);
minuto=le_rtc(1);
segundo=le_rtc(0);
dia=le_rtc(4);
mes=le_rtc(5);
ano=le_rtc(6);
ledint = !ledint; // inverte o led de teste - pisca a cada 2 *12 interrupcoes = 1 seg.
output_bit (pin_b0,ledint);
reg= read_adc(); //Tensão e corrente

//escreve_eeprom(0,end,reg); não funciona a escrita i2c dentro da interrupção do timer
write_eeprom( end, reg );
++end; if(end>=127){end=0;}
segundodec=bcd_to_dec(segundo);minutodec=bcd_to_dec(minuto);horadec=bcd_to_dec(hor
a);
diadec=bcd_to_dec(dia);mesdec=bcd_to_dec(mes);anodec=bcd_to_dec(ano);
if (segundodec==05 &&
(minutodec==00||minutodec==10||minutodec==20||minutodec==30||minutodec==40||minutode
c==50))
//if
((segundodec==00||segundodec==10||segundodec==20||segundodec==30||segundodec==40||
segundodec==50))
{flagwrite=1;}
```

```
//endpromext=(minutodec/10)+(horadec*6)+((diadec-1)*24*6*2)+24*6*k;
//endpromext=(segundodec/10)+(minutodec*6); }//Não aceita  DE JEITO NENHUM escrever
na eeprom ext por interrupção do timer via i2c
//printf("\n\rEndpromext = %lu e reg = %u \n\r, segundodec =
%lu\n\r",endpromext,reg,segundodec); //Aceita imprimir via USB

set_timer1(3036 + get_timer1());   }} // Conta 62.500 x 8 = 0,5s

////////////////////////////////////////////////////////////////////////////
void main() {
usb_cdc_init(); // Inicializa o protocolo CDC
usb_init();    // Inicializa o protocolo USB
usb_task();  // Une o periférico com a usb do PC

OSCCON=0B01100110; //Clock interno do processador de 4MHZ

setup_adc_ports(AN0_TO_AN1);  //Habilita entradas analógicas - A0 A1
setup_adc(ADC_CLOCK_INTERNAL);    //Configuração do clock do conversor AD

enable_interrupts (global); // Possibilita todas interrupcoes
enable_interrupts (int_timer1); // Habilita interrupcao do timer 1
setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8);// inicia o timer 1 em 8 x 62500 = 0,5s
set_timer1(3036);

setup_wdt(WDT_ON); //Habilita o temporizador cão de guarda - resseta se travar o programa
principal ou ficar em algum getc();
```

```
while (1) {
//*************************************************************************
if (flagwrite==1) {flagwrite=0; //Flag de gravação setada na interrupção do timer quando chega a hora de gravar
k=0;
for(k=0;k<2;k++)
{
set_adc_channel(k);
tempo_ms(20);
regi[k]= read_adc(); //Tensão M1[0], correnteM1[1]
endpromext=(minutodec/10)+(horadec*6)+((diadec-1)*24*6*2)+24*6*k;
//endpromext=(segundodec/10)+(minutodec*6)+((diadec-1)*60*6*2)+60*6*k;  //Para teste 60 em vez de 24
escreve_eeprom(0,endpromext, regi[k]);
printf("\r\nPosicao = %lu -> Sensor[%lu] = %u\r\n",endpromext,k,regi[k]);
}
}
//*************************************************************************
led = !led; // inverte o led de teste
output_bit (pin_b7,led);

restart_wdt(); // Limpa a flag do WDT para que não haja reset
tempo_ms(500);
//*************************************************************************
if (kbhit(1)) { //verifica se acabou de chegar um novo dado no buffer de recepção, //depois o kbhit é zerado para próximo dado

escravo=getc(); //comando é o Byte recebido pela serial,
if (escravo=='A')
{   funcao=getc();

switch (funcao) //UTILIZAR VALORES DECIMAIS EM DOIS DIGITOS. ex:06 ou 23 ou 15
{
```

```
//*************************************************************************
case '4':
{
endrtc=getc();
valorrtc1=getc();
valorrtc2=getc();              //Ex: A4M43 - Altera os minutos para 43
if (endrtc=='H') { endereco=2;}  //Escreve o endereco das horas
if (endrtc=='M') { endereco=1;}  //Escreve o endereco dos minutos
if (endrtc=='S') { endereco=0;}  //Escreve o endereco dos segundos
if (endrtc=='D') { endereco=4;}  //Escreve o endereco do dia
if (endrtc=='N') { endereco=5;}  //Escreve o endereco do mes
if (endrtc=='Y') { endereco=6;}  //Escreve o endereco do ano

if (valorrtc1>='0'&&valorrtc1<='9') {numquant=(valorrtc1-0x30);}
if (valorrtc2>='0'&&valorrtc2<='9') {numquant=numquant*10+(valorrtc2-0x30);

valor=numquant;

if (endereco==0) { if(valor>59) {valor=0;}}
```

```
if (endereco==1) { if(valor>59) {valor=0;}}
if (endereco==2) { if(valor>23) {valor=0;}}
if (endereco==4) { if(valor>31) {valor=1;}}
if (endereco==5) { if(valor>12) {valor=1;}}
if (endereco==6) { if(valor>99) {valor=0;}}

//---------Converte byte hexadecimal para byte BCD decimal ----------
valorbcd=dec_para_bcd(valor);
//-------------------------------------------------------------------

escreve_rtc(endereco,valorbcd); //Valor1 é byte BCD (decimal).
//printf("\r\nVALOR ESCRITO = %2x\r\n",valorbcd);
//printf(r\nPOSICAO = %2x\r\n",endereco);

hora=le_rtc(2);minuto=le_rtc(1);segundo=le_rtc(0);

printf(r\nA4%2x:%2x:%2x",hora, minuto,segundo);
printf(%2x%2x%2x\r\n",le_rtc(4), le_rtc(5), le_rtc(6));
}
}
break;

//////////////////////FUNCAO 5: LÊ RELÓGIO//////////////////////Ex: A5- Lê o relógio e o calendário
case '5':
printf(usb_cdc_putc,"\r\nA5 %2x:%2x:%2x",le_rtc(2), le_rtc(1),le_rtc(0));
printf(usb_cdc_putc," %2x%2x%2x\r\n",le_rtc(4), le_rtc(5), le_rtc(6));
break;
//////////////////////FUNCAO 6: LÊ BUFFER EEPROM//////////////////////Ex: A6 09(DIA) 0(SENSOR)
case '6':{
posmeme1=getc();
posmeme2=getc();
sensor=getc();
if (posmeme1>='0' && posmeme1<='9') {bufferdia=(posmeme1-0x30);}
if (posmeme2>='0' && posmeme2<='9') {bufferdia=bufferdia*10+(posmeme2-0x30);}
if (sensor>='0' && sensor<='1')  {k=(sensor-0x30);}
printf(usb_cdc_putc,"Buffer Sensor %lu - Dia %lu\r\n",k,bufferdia);
tempo_ms(10);
//puloext=((bufferdia-1)*60*6*2)+60*6*k;// Seleciona buffer de teste de tensao
puloext=((bufferdia-1)*24*6*2)+24*6*k;// Seleciona buffer
for(i=0; i<6; ++i)
```

```
{
//for(j=0; j<60; ++j) {printf(usb_cdc_putc,"%2u ", le_eeprom(0,puloext+(i*60+j)) );}
//"%2u\n\r" para gerar gráfico no excell
for(j=0; j<24; ++j){printf(usb_cdc_putc,"%2u ", le_eeprom(0,puloext+(i*24+j)) );}
tempo_ms(15);
}
printf(usb_cdc_putc,"\r\n"); //posiciona próxima linha
}
break;

}}}
}}
```

Capítulo 28
Transmissão de dados via GSM

A sigla GSM significa Global Standard Mobile ou Global System for Mobile Communications que quer dizer Sistema Global para Comunicações Móveis. O GSM é um sistema de celular digital baseado em divisão de tempo, como o TDMA, e é considerado a evolução deste sistema, pois permite, entre outras coisas, a troca dos dados do usuário entre telefones através do SIM Card e acesso mais rápido a serviços WAP e Internet, através do sistema GPRS.

A transmissão de dados GSM pode ser feita por:

- GPRS (*General Package Radio Service*): É uma conexão em uma rede de pacote de dados. Uma vez conectado nessa rede, o sistema estará sempre on line, podendo transferir dados imediatamente. O GPRS é compatível com o protocolo de rede TCP/IP e as operadoras de GSM disponibilizam um gateway para a Internet, possibilitando conectar e controlar equipamentos *wireless* através da Internet. Como o GPRS é baseado no protocolo IP, ele necessita de autenticação de um servidor da internet.

- SMS (*Short Message Service*): É o serviço de envio/recebimento de pequenas mensagens de texto do tipo datagrama, sem autenticação de um servidor de internet.

Os Modems GSM são controlados através de comandos AT. Esses comandos são normalizados pelas normas GSM 07.07 e GSM 07.05.

A manipulação do modem pode ser realizada em algum emulador de comunicação serial como o Hyperterminal, nele deve-se ajustar para 9600 bps,e não esquecendo de instalar o SIM Card no modem.

28.1 Comandos AT para enviar mensagens SMS de um computador para um celular ou modem GSM

A Tabela 28.1 mostra a lista dos commandos AT para escrever e enviar mensagens SMS.

Tabela 28. 1: Comandos AT (1).

Comando AT	Significado
+CMGS	Envia mensagem
+CMSS	Envia uma mensagem armazenada
+CMGW	Escreve uma mensagem na memória
+CMGD	Apaga mensagem
+CMGC	Envia comando
+CMMS	Envia mais mensagens

Fonte: Elaborado pelo autor.

Exemplo feito com um computador:
1-**AT**
2-OK
3- **AT+CMGF=1**
4- OK
5-**AT+CMGS="+558588888888"**//<ctrl + z (minúsculo)> digita-se o texto após >
6-**> Teste de mensagem**
7- OK
+CMGS: 170 OK

Abaixo está o significado de cada linha:

1- Testa conexão com o modem.
2- Modem conectado.
3- Coloca o celular no modo texto.
4- Modo texto confirmado.
5- Número do telefone que irá receber a mensagem.
6- O modem retorna o caractere "**>**" solicitando a mensagem a ser enviada (ao final: "ctrl z").
7- Mensagem enviada.

28.2 Comandos AT para receber mensagens SMS em um computador enviadas por um celular ou modem GSM

A Tabela 28.2 mostra a lista dos commandos AT para receber e enviar mensagens SMS.

Tabela 28. 2: Comandos AT (2).

Comando AT	Significado
+CNMI	New message indications
+CMGL	Lista mensagens
+CMGR	Lê menssagens
+CNMA	Reconhecimento de nova menssagem

Fonte: Elaborado pelo autor.

Exemplo feito com um computador:
AT
OK
AT+CMGF=1
OK
AT+CMGL="ALL"
+CMGL: 1,"REC READ","+85291234567",,"06/11/11,00:30:29+32"
Hello, welcome to our SMS tutorial.
+CMGL: 2,"REC READ","+85291234567",,"06/11/11,00:32:20+32"
A simple demo of SMS text messaging.

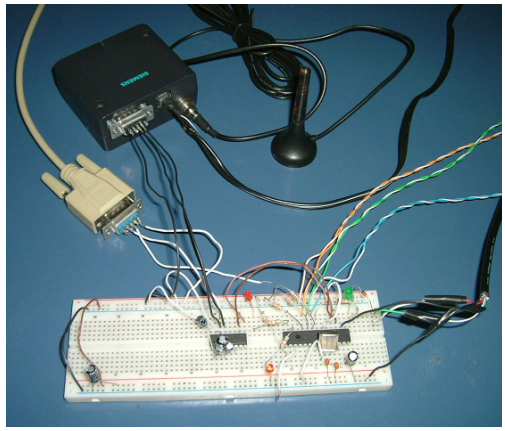
Adiante é apresentado um exemplo de como enviar uma mensagem SMS do modem GSM para um celular com uso do PC. Os comandos enviados ao modem estão em negrito para diferenciar de suas respostas.

1-**AT**
2-OK
3- **AT+CMGF=1**
4- OK
5-**AT+CMGS=”+558588888888”**
6-**>Intrusão**
7- OK

As Figuras28.1 e 28.2apresentam a foto em *protoboard* e o circuito esquemático para transmissão GPRS/GSM. A conexão USB observado no esquema, foi utilizada pela ferramenta SanUSB para a alimentação do circuito e gravação do programa no PIC através do PC. O LED verde foi usado por esta ferramenta para sinalizar o momento em que o sistema estava no modo de gravação. O vermelho simulou o acionamento do alarme, como descrito anteriormente. As chaves conectadas aos pinos 23, 24 e 25, representam as chaves sinalizadoras dos três sensores utilizados. A figura abaixo mostra também o dispositivo MAX232 usado na interface RS/EIA-232 entre o microcontrolador e o modem. Este, representado na figura apenas pelo conector DB9, possui o pino 2 para transmissão de dados e o 3 para recepção, já que se trata de um equipamento do tipo DCE (*Data Comunication Equipment*).

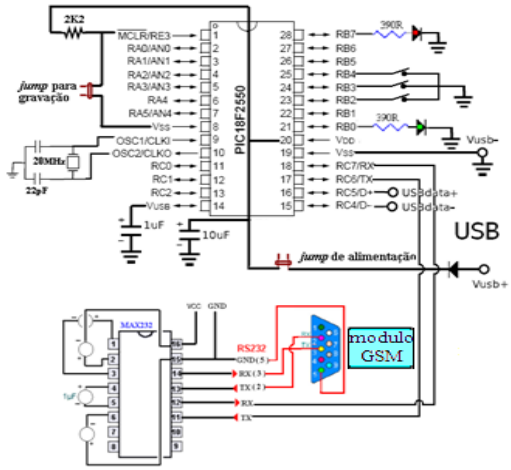
O conversor TTL/EIA-232 Max232 é utilizado para conexão do módulo GSM/GPRS ao sistema, cujos comandos AT são descritos no próximo tópico.

Figura 28. 1: Módulo GSM/GPRS conectado à Ferramenta SanUSB.



Fonte: Elaborado pelo autor.

Figura 28. 2: Esquemático e foto do datalogger conectado ao PIC.



Fonte: Elaborado pelo autor.

```
#include “SanUSB1.h”
short int  ledpisca;

#USE RS232 (BAUD=9600,XMIT=PIN_C6,RCV=PIN_C7,stream=PC)

void main(){
clock_int_4MHz();

printf("AT+CMGF=1\r"); //configura modo texto para o modem

ledpisca=!ledpisca; // ledpisca é igual ao inverso de ledpisca
output_bit(pin_b7,ledpisca); // b7 recebe o valor de ledpisca
tempo_ms(2000);

printf("AT+CMGS=\"+558588888888\"\r"); //5- Envia para o numero de destino
ledpisca=!ledpisca;
output_bit(pin_b7,ledpisca);
tempo_ms(2000);

printf("Alarme atuado\r"); //6 - escreve a mensagem para o modem GSM
putc(0x1A); // control z

ledpisca=!ledpisca;
output_bit(pin_b7,ledpisca);
tempo_ms(2000);

putc(0x0D);

while(TRUE){    nivel_alto(pin_B7);
                tempo_ms(500);
                nivel_baixo(pin_B7);
                tempo_ms(500);
}}
```

Capítulo 29

O protocolo MODBUS embarcado

O protocolo Modbus foi desenvolvido pela Modicon Industrial Automation Systems, hoje Schneider, para comunicar um dispositivo mestre com outros dispositivos escravos. Embora seja utilizado normalmente sobre conexões seriais padrão EIA/RS-232 e EIA/RS-485, ele também pode ser usado como um protocolo da camada de aplicação de redes industriais tais como TCP/IP sobre Ethernet.

Este é talvez o protocolo de mais utilizado em automação industrial, pela sua simplicidade e facilidade de implementação.

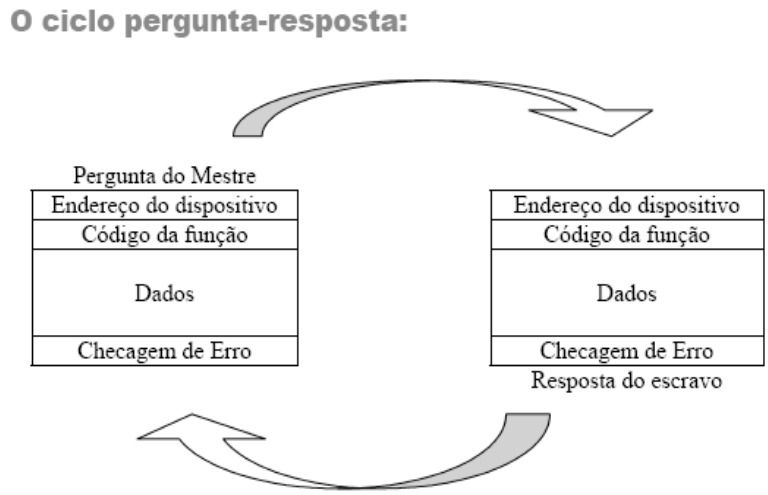
A motivação para embarcar um microcontrolador em uma rede MODBUS pode ser por:

- Baixo custo;
- Tamanho reduzido;
- Alta velocidade de processameto (1 a 12 MIPs);
- Possuir 10 canais ADs internos com resolução de 10 bits;
- Ferramentas de desenvolvimento gratuitas e possibilidade de programação da memória de programa sem necessidade de hardware adicional, bastando uma porta USB;
- Estudo das características construtivas de hardware e de software de uma rede MODBUS.

29.1 Modelo de comunicação

O protocolo Modbus é baseado em um modelo de comunicação mestre-escravo, onde um único dispositivo, o mestre, pode iniciar transações denominadas queries. O demais dispositivos da rede (escravos) respondem, suprindo os dados requisitados pelo mestre ou executando uma ação por ele comandada. Geralmente o mestre é um sistema supervisório e os escravos são controladores lógico-programáveis. Os papéis de mestre e escravo são fixos, quando se utiliza comunicação serial, mas em outros tipos de rede, um dispositivo pode assumir ambos os papéis, embora não simultaneamente, conforme Figura 29.1.

Figura 29. 1: Checagem de dados.



Fonte: Elaborado pelo autor.

29.2 Detecção de erros

Há dois mecanismos para detecção de erros no protocolo Modbus serial: bits de paridade em cada caractere e o frame check sequence ao final da mensagem. O modo RTU utiliza como *frame check sequence* um valor de 16 bits para o CRC (ciclic *redundance check*), utilizando como polinômio, $P(x) = x^{16} + x^{15} + x^2 + 1$. O registro de cálculo do CRC deve ser inicializado com o valor 0xffff.

29.3 Modos de transmissão

Existem dois modos de transmissão: ASCII (American Code for Information Interchange) e RTU (Remote Terminal Unit), que são selecionados durante a configuração dos parâmetros de comunicação.

Como a comunicação geralmente utilizada em automação industrial é em modo RTU (ver Figura 29.2), o projeto proposto foi desenvolvido nesta forma de comunicação.

Figura 29. 2: Modo RTU.

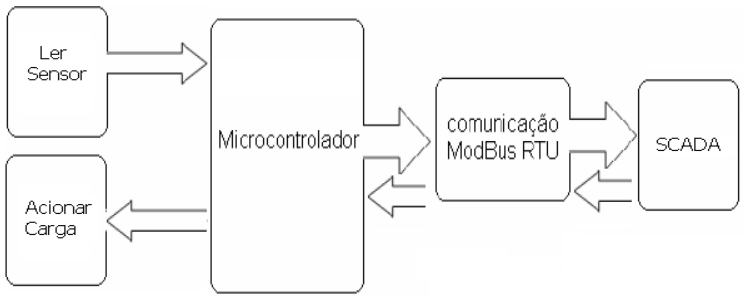
Modo RTU

Start	Endereço	Função	Dados	CRC	END
Silêncio 3..5 chars	← 8 bits→	← 8 bits→	←N x 8 bits→	←16 bits→	Silêncio 3..5 chars

Fonte: Elaborado pelo autor.

A implementação prática de um projeto com o modbus embarcado é mostrada no diagrama de blocos da Figura 29.3.

Figura 29. 3: Diagrama de blocos comunicação ModBus.



Fonte: Elaborado pelo autor.

Para testar a comunicação com escravo Modbus (microcontrolador) em protocolo RTU, através da porta serial emulada pela USB do PC, é possível utilizar o Modbus Tester que é um software livre de Mestre MODBUS escrito em C++. Ele pode ser utilizado para testar se o desenvolvimento das rotinas de processamento do protocolo Modbus contidas no microcontrolador. Durante o teste é possível utilizar também um sistema supervisório real como o Elipse SCADA.

Os sistemas supervisórios são softwares que permitem que sejam monitoradas e rastreadas informações de um processo produtivo ou instalação física. Tais informações são coletadas através de equipamentos de aquisição de dados e, em seguida, manipuladas, analisadas, armazenadas e posteriormente apresentadas ao usuário. Estes sistemas também são chamados de SCADA (Supervisory Control and Data Aquisition).

Para comunicar com o supervisório Elipse é necessário instalar um driver dedicado a comunicação ModBus RTU, que é fornecido gratuitamente pela própria Elipse.

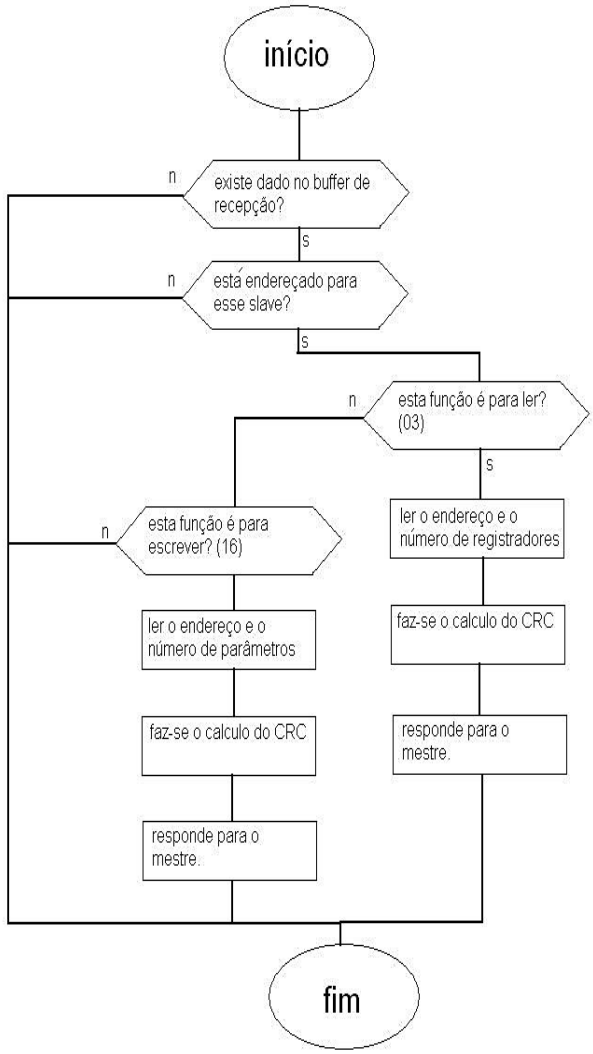
O modo Modbus RTU com microcontrolador PIC desse projeto, suporta funções de leitura (3) e escrita (16).

```
#include “SanUSB1.h”
#include <usb_san_cdc.h>// Biblioteca para comunicação serial
long int checksum = 0xffff;
unsigned int x,i,y,z;
unsigned char lowCRC;
unsigned char highCRC;
int tamanhodata;
int32 buffer[100];
void CRC16 (void) //Modo RTU
{
for (x=0; x<tamanhodata; x++)
{
checksum = checksum^(unsigned int)buffer[x];
for(i=8;i>0;i--)
{
if((checksum)&0x0001)
                checksum = (checksum>>1)^0xa001;
                else
                checksum>>=1;
}
}
highCRC = checksum>>8;
checksum<<=8;
lowCRC = checksum>>8;
buffer[tamanhodata] = lowCRC;
buffer[tamanhodata+1] = highCRC;
checksum = 0xffff;
}
void ler (void)
{
buffer[2]=getc();
buffer[3]=getc();
buffer[4]=getc();
buffer[5]=getc();
buffer[6]=getc();
buffer[7]=getc();
tempo_ms(3);
buffer[2]=0x02;
buffer[3]=0x00;
buffer[4]=port_a;  //o buffer[4] leva o valor de entrada da porta A do microcontrolador para o
//SCADA
tamanhodata = 5;
CRC16();
}
void rxler (void)  //Leu a porta a no buffer[4] e escreve o CRC no buffer[5] e [6], pois
tamanhodata = 5
{
printf(usb_cdc_putc,"%c%c%c%c%c%c%c",buffer[0],buffer[1],buffer[2],buffer[3],buffer[4],buffer[
5],buffer[6]); // 6 bytes
}

void escrever (void)
{
buffer[2]=getc();
buffer[3]=getc();
buffer[4]=getc();
buffer[5]=getc();
buffer[6]=getc();
buffer[7]=getc();
```

```
buffer[8]=getc();
buffer[9]=getc();
buffer[10]=getc();
tempo_ms(3);
tamanhodata = 6;
CRC16();
PORTB = buffer[8]; //A porta B do microcontrolador recebe o valor enviado pelo SCADA
}

void rxescrever (void)
{
printf(usb_cdc_putc,"%c%c%c%c%c%c%c%c",buffer[0],buffer[1],buffer[2],buffer[3],buffer[4],buff
er[5],buffer[6],buffer[7]); //7 bytes
}

void main()
{
clock_int_4MHz();

PORTB= 0b00000000;
while(1)
{
if (kbhit(1))
{
//verifica se acabou de chegar um novo dado no buffer USB, depois o kbhit é zerado para
//próximo dado

buffer[0]=getc();
z = buffer[0];
if (z==1) //verifica se o endereco do slave e igual a 1
{
buffer[1]=getc(); //verifica a função contida no segundo byte buffer[1]

y = buffer[1];
}
if (y==3) //verifica se a função é para leitura e encaminha para leitura de variável do
//microcontrolador
{
ler();
rxler();
}
if (y==16) //verifica se a função é para escrita no microcontrolador, ou seja, comando de
//atuação do uC
{
escrever();
rxescrever();
}}}}
```

O fluxograma desse firmware é mostrado na Figura 29.4.

Figura 29. 4: Fluxograma do sistema de comunicação ModBus.



Fonte: Elaborado pelo autor.

Note que a comunicação serial desse projeto foi emulada via USB, para aplicação em um processo real é necessário utilizar um transceptor ou TTL/EIA-232 (MAX232) ou transceptor ou TTL/EIA-485 (MAX485). Com o MODBUS embarcado é possível integrar um microcontrolador, preservando as limitações de funções, em um processo de automação industrial que utiliza esse protocolo.

Capítulo 30
Introdução à Multitasking e Sistemas Operacionais em tempo real (RTOS)

O uso de um sistema operacional em tempo real (RTOS) para processamento de multitarefas é uma realidade cada vez mais presente nos projetos de sistemas embarcados. A ferramenta computacional SanUSB implementa um RTOS livre através dos compiladores MPLABX C18 e CCS, chamado de RTOSB e baseado em interrupção de temporizadores.

Uma das principais características de um RTOS é a capacidade de processar tarefas concorrentes, ou seja, tarefas paralelas. Dessa forma, o RTOS torna a programação de projetos reais mais simples, pois basta descrever cada tarefa em uma função task do firmware, que o RTOS se encarrega do gerenciamento do processo. Dessa forma, O RTOS é baseado na ideia de multitarefas (*multithread*), onde cada tarefa é uma função em C do firmware em laço infinito.

Em um sistema multitarefa, inúmeras tarefas exigem tempo da CPU, e uma vez que existe apenas uma CPU, é necessária alguma forma de organização e coordenação pelo RTOS para que cada tarefa tenha o tempo que necessita. Na prática, cada tarefa tem um intervalo de tempo muito curto, assim parece que as tarefas são executadas de forma paralela e simultânea.

As duas práticas descritas abaixo de RTOS foram desenvolvidas com a placa SanUSB, que pode ser construída seguindo os tutoriais disponíveis nos arquivos do grupo SanUSB. A Figura 30.1 ilustra a conexão para a prática com RTOS.

• **Prática 1**: Nesta prática o RTOS executa em paralelo três tarefas concorrentes e paralelas em loop infinito para acionar 3 leds, conectados nos pinos B7, B6 e B5, de forma independente.

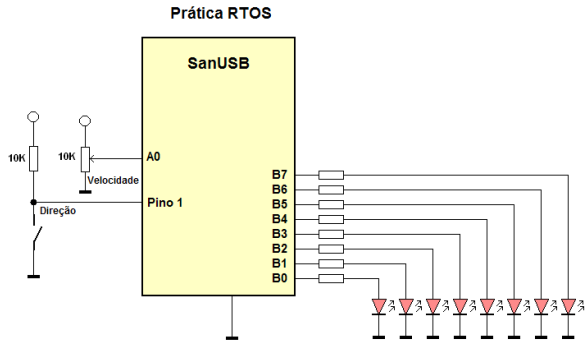
• **Prática 2**: Nesta prática o RTOS executa em paralelo:

1- Tarefa de leitura do AD com potenciômetro para modificar a velocidade de brilho dos leds;

2- Tarefa de rotação de oito leds na porta B; e

3- Tarefa que inverte o sentido de rotação dos leds por botão no pino 1 (PIN_E3) utilizando display de sete segmentos como leds.

Figura 30. 1: RTOS.



Fonte: Elaborado pelo autor.

OBS: Dentro de cada projeto, necessário inserir o cabeçalho OSAcfg.h, como descrito abaixo, que deve indicar a quantidade de tarefas e as características do projeto como a prioridade das tarefas.

#ifndef _OSACFG_H
#define _OSACFG_H
#define OS_TASKS 3
#define OS_DISABLE_PRIORITY
#define OS_ENABLE_TTIMERS

O firmware as prática 1 que executa três tarefas concorrentes está descrito abaixo.

```
#include "SanUSB48.h" //Exemplo com 4 tarefas (task) concorrentes (paralelas)
#include <osa.h> //Vídeo-aula: https://www.youtube.com/watch?v=s6BG8ZN0aDk
//Evitar uso de outros laços dentro das tasks (como for, do - while, etc.!)

#pragma interrupt interrupcao
void interrupcao(){

if (PIR1bits.TMR1IF)
   {
      PIR1bits.TMR1IF = 0;
TMR1H          = 0xD8;
      TMR1L          = 0xF0;
      OS_Timer();
}
}

void PIC_Init(void)
{
   LATB          = 0x00;
TRISB          = 0x00;

   T1CON          = 0x80;    // modo 16 bits
   TMR1H          = 0xD8;    // 1ms
   TMR1L          = 0xF0;
```

```
INTCON = 0;
   INTCONbits.PEIE = 1;
   PIR1bits.TMR1IF = 0;        // Flag interrupcao Timer1
   PIE1bits.TMR1IE = 1;        // Habilita interrupcao Timer1
   T1CONbits.TMR1ON= 1;        // Liga Timer1
}

void Task_1(void)
{
   while(1)
   {
      inverte_saida(pin_b7);
      OS_Delay(1000);
   }
}

void Task_2(void)
{
   while(1)
   {
      OS_Delay(200);
      LATBbits.LATB6^=1;
   }
}

void Task_3(void)
{
   while(1)
   {
      OS_Delay(300);
LATBbits.LATB5^=1;
   }
}
/* // OSAcf.h configurado com 4 tasks
void Task_4(void)
{
   while(1)
   {
      OS_Delay(400);
      LATBbits.LATB4^=1;
   }
} */
void main(void)
{
   clock_int_48MHz();
   PIC_Init();                  // Configurações gerais do PIC

   OS_Init();

   OS_Task_Create(1,Task_1);      // Criando uma tarefa,  prioridade 1
   OS_Task_Create(2,Task_2);      // Criando uma tarefa,  prioridade 2
   OS_Task_Create(3,Task_3);      // Criando uma tarefa,  prioridade 3
   //OS_Task_Create(4,Task_4);      // Criando uma tarefa,  prioridade 4

   OS_EI();                     // Habilita interrupcoes
   OS_Run();                      // Executa o RTOS
}
```

Em um sistema multitarefa, inúmeras tarefas exigem tempo da CPU, e uma vez que existe apenas uma CPU, é necessária alguma forma de organização e coordenação para cada tarefa tenha o tempo que necessita. Na prática, cada tarefa tem um intervalo de tempo muito curto, assim parece que as tarefas são executadas de forma paralela e simultânea.

Quase todos os sistemas baseados em microcontroladores executam mais de uma atividade e trabalham em tempo real. Por exemplo, um sistema de monitoramento da temperatura é composto de três tarefas que, normalmente, que se repete após um pequeno intervalo de tempo, a saber:

- Tarefa 1 lê a temperatura;
- Tarefa 2 Formata o valor da temperatura;
- Tarefa 3 exibe a temperatura;

30.1 Máquinas de estado

As máquinas de estado são simples construções usadas para executar diversas atividades, geralmente em uma seqüência. Muitos sistemas da vida real que se enquadram nesta categoria. Por exemplo, o funcionamento de uma máquina de lavar roupa ou máquina de lavar louça é facilmente descrito com uma máquina de estado de construção. Talvez o método mais simples de implementar uma máquina de estado em C é usar um switch-case. Por exemplo, nosso sistema de monitoramento de temperatura tem três tarefas, nomeado Tarefa 1, Tarefa 2, Tarefa 3 e, como mostrado na Figura 30.2.

Figura 30. 2: Implementação de máquina de estado.

Implementação de máquina de estado

Fonte: Elaborado pelo autor.

A máquina de estado executa as três tarefas usando declarações switch-case. O estado inicial é 1, é incrementado a cada tarefa do Estado para selecionar o próximo estado a ser executado. O último estado seleciona o estado 1, e há um atraso no final do switch-case. A máquina de estado é executada continuamente no interior de um laço infinito.

Em muitas aplicações, os estados não precisam ser executados em seqüência. Pelo contrário, o próximo estado é selecionado direto pelo estado atual ou baseado em alguma condição.

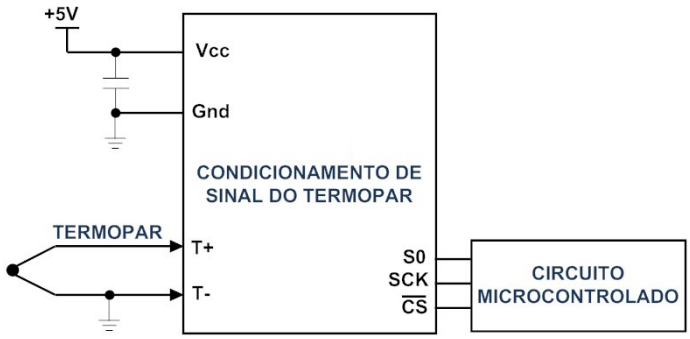
O RTOS também é o responsável por decidir o agendamento (*scheduling*) da sequência das tarefas a serem executadas considerando os níveis de prioridade e o tempo máximo de execução de cada tarefa. Mais detalhes sobre RTOS podem ser vistos em http://sanusb.blogspot.com.br/2012/12/rtos-sanusb.html.

Capítulo 31
SPI

A SPI (*Serial Peripheral Interface*) é um protocolo de dados seriais síncronos utilizado em microcontroladores para comunicação entre o microcontrolador e um ou mais periféricos. A comunicação SPI pode ser realizada entre o circuito de condicionamento do termopar e o microcontrolador por hardware, pois o microcontrolador possui um circuito periférico interno SPI, ou por software, onde a transmissão serial síncrona dos bits é programada no firmware do microcontrolador. O presente projeto utiliza a transmissão dos dados do circuito de condicionamento do termopar por software.

A Força Eletromotriz gerada entre os terminais de um sensor de temperatura termopar é muito pequena para que possa ser interpretada diretamente por um processador ou dispositivo computacional, existindo então a necessidade de um circuito de condicionamento que permita um ganho de sinal de leitura, para que possa ser interpretada pelo circuito microcontrolado. Diante do exposto, decidiu-se utilizar na presente pesquisa o CI de condicionamento MAX31855(MAXIM, 2012), de acordo com o esquemático apresentado na Figura 31.1.

Figura 31. 1: Circuito de condicionamento de sinal do termopar.



Fonte: Elaborado pelo autor.

O CI de condicionamento de sinal do termopar, MAX31855, realiza uma compensação de junção fria e digitaliza o sinal a partir de um termopar tipo K. Os dados de saída tem resolução de 12 bits, compatível para leitura via SPI, do inglês *Serial Peripheral Interface*.

Este conversor pode operar em temperaturas a partir de 0,25ºC até temperaturas elevadas como 1.024ºC.

O CI MAX31855, com diagrama de blocos e indicação dos pinos, processa a conversão AD do valor de tensão lida na saída do termopar (T+ e T-) e transmite posteriormente os resultados para o microcontrolador através de uma interface serial periférica síncrona SPI.

É importante salientar que o CI MAX31855 utiliza um circuito de condicionamento composto por um filtro LC (indutivo-capacitivo) na entrada e um *buffer* para conversão de tensão de 5 V (tensão de alimentação do microcontrolador) para 3,3 V (tensão de alimentação do CI).

Os resultados da conversão AD são lidos no pino S0 (*serial data output*), colocando o pino CS (*chip select*) em nível lógico baixo e aplicando um sinal de *clock* em SCK (*serial clock input*).

Quando o pino CS está em nível lógico baixo, o circuito de condicionamento assume o estado de transmissão. Dessa forma, qualquer processo de conversão é desabilitado. Para iniciar uma nova conversão AD do sinal analógico de saída do termopar em quatro bytes é necessário que o pino CS esteja em nível lógico alto.

A leitura completa da interface serial requer 32 ciclos de *clock*. Os 32 bits de saída são lidos na borda de descida do *clock*, fornecido pelo microcontrolador. O primeiro bit, D31, é um bit que identifica o sinal do valor de temperatura entre positivo e negativo. Os bits D30-D18 contêm a temperatura convertida na ordem do MSB (bit mais significativo) para LSB (bit menos significativo). Desta forma, cada bit transmitido da conversão AD corresponde a 0,25ºC. Um exemplo de aplicação de leitura de um termopar, com função SPI e postagem de valores na nuvem, pode ser conferido em https://github.com/SanUSB.

A comunicação SPI pode ser realizada entre o circuito de condicionamento do termopar e o microcontrolador por hardware, pois o microcontrolador possui um circuito periférico interno SPI, ou por software, onde a transmissão serial síncrona dos bits é programada no firmware do microcontrolador. O presente projeto utiliza a transmissão dos dados do circuito de condicionamento do termopar por software.

Capítulo 32

APÊNDICE I: Gravação online de microcontroladores PIC via Sistema Embarcado Linux

Este apêndice descreve a gravação de microcontroladores pela nuvem, via servidor com interface entre o usuário e um sistema embarcado Linux baseado em Raspberry Pi. O Raspberry Pi é um computador com o tamanho de um cartão de crédito e possui conexões USB para teclado, mouse ou WiFi dongle, entrada HDMI, saída de áudio, entrada para cartão micro SD, onde é gravado o sistema operacional. Uma das vantagens é o baixo custo do hardware, comparado a PCs convencionais, além do custo zero do software Linux. A principal vantagem é a possibilidade de gravação via nuvem de um PIC conectado ao sistema embarcado Linux via USB, através de um programa instalado neste, que utiliza o protocolo HID (*Human Interface Device*). Dessa forma, é possível realizar atualizações no firmware do microcontrolador e configurar o sistema embarcado Linux via Internet visando aplicações com a filosofia de IoT (Internet das coisas). O sistema apresentado possibilita o uso para o aprendizado de linguagens de programação, sistemas embarcados e a integração com os microcontroladores de forma didática para aplicação em projetos reais nas áreas de Informática, Eletrônica, Robótica e Sistemas Embarcados.

32.1 S. O. no Raspberry Pi

Dentre os sistemas operacionais com Rpi, o mais utilizado é o Raspbian (necessário cartão SD de no mínimo 4 GB). Para projetos de eletrônica, robótica, servidores e outros já que deixa livres no Rpi o máximo de recursos possível como memória, processador e consumo de corrente, está disponível também o Minibian (MINImal raspBIAN) que cabe em um cartão microSD de apenas 1 GB.

32.2 Raspbian

O **Raspbian** é um dos sistemas operacionais oficiais do Rpiderivado do Debian e distribuído pela *Raspberry Pi Foundation*. Esta é a distribuição ideal para quem tem menos conhecimentos dos sistemas Linux ou simplesmente necessite de um sistema operacional pronto. O **Raspbian** é um sistema quase completo, já que vem com diversas aplicações pré-instaladas, os *drivers* mais usuais, ferramentas para facilitar algumas configurações necessárias, entre outros. Muitas aplicações e módulos dedicados à programação já vêm

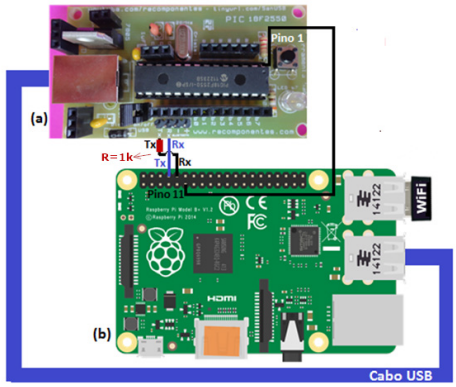
incluídos na imagem do **Raspbian,** bastando iniciar o sistema para acessá-las. **Para iniciantes com o** Rpi**, que desejam experimentar as potencialidades ou começar a programar e desenvolver projetos de sistemas embarcados, o**Raspbian é o mais recomendado. Minibian (MINImal raspBIAN)

O **Minibian** é uma versão minimizada do **Raspbian**. Destina-se àqueles que precisam de um sistema o mais leve possível e sem ferramentas e aplicações desnecessárias. O **Minibian** é excelente, por exemplo, para projetos de eletrônica, robótica, servidores e outros já que deixa livre, no **Raspberry Pi**, o máximo de recursos possível como memória, processador, consumo de corrente, entre outros.Esta distribuição não traz sequer o *GUI* (ambiente de janelas) e cabe num cartão de apenas 1 GB que hoje em dia já não é muito utilizado e às vezes tem-se em desuso na gaveta.

32.3 Descrição da conexão entre a placa SanUSB e o Raspberry Pi

Neste tópico, são descritas as duas formas de comunicação utilizadas neste trabalho, sendo elas a comunicação serial e a interface USB. Para realizar a comunicação serial entre um microcontrolador PIC e Raspberry Pi, é preciso utilizar os pinos GPIO 14 (TX) e o GPIO 15 (RX). Neste projeto, o computador utilizado para realizar a gravação do microcontrolador da placa PIC SanUSB é o sistema embarcado Linux Raspberry Pi, cujos pinos de conexão para gravação via USB e comunicação serial são ilustrados na Figura 32.1.
O pino físico 11 do Rpi deve ser ligado por um fio ao pino 1 (pin_e3) do microcontrolador.

Figura 32. 1: Conexão serial entre a placa SanUSB (a) e o Raspberry Pi (b).



Fonte: Elaborado pelo autor.

A Figura 32.2 mostra a interface gráfica desenvolvida para gravação direta de microcontroladores. A transferência de programas para os microcontroladores é normalmente efetuada através de um hardware de gravação específico e via cabo. Através desta ferramenta, é possível efetuar a descarga de programas para o microcontrolador de forma *online* através da porta USB do sistema embarcado Rpi.

Figura 32. 2: Interface de gravação e mensagem de programa gravado.

Fonte: Elaborado pelo autor.

32.4 Processo de gravação e acionamento *online*

Para iniciar a aplicação proposta, segue abaixo uma lista do material utilizado.

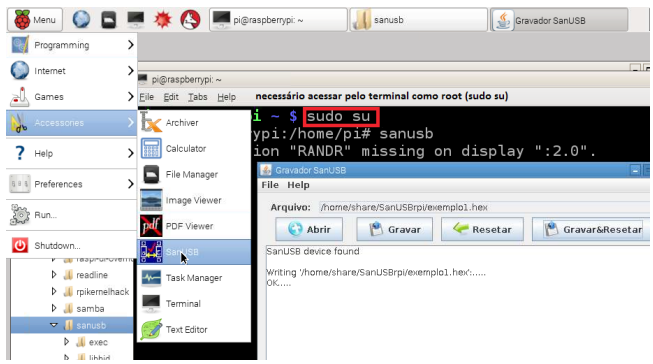
- Placa Raspberry Pi;
- Placa SanUSB com PIC18F2550;
- Cartão Micro SD de 4GB mínimo;
- Cabo USB e três fios para conexão entre as placas.
- Cabo Ethernet ou Modem WiFi USB

É possível realizar a gravação online de microcontroladores PIC através do Rpide duas maneiras: via terminal serial do LINUX e via interface gráfica (SanUSB, 2015).É possível utilizar o protocolo SSH, default no Raspbian, para acessar o Rpi de um computador remoto sendo necessário um programa SSH, como o Putty, conectado na porta 22.

Vale salientar que para abrir a interface gráfica, basta digitar “sanusb” no LXTerminal, como ilustrado na Figura 32.3. Mais detalhes sobre gravação de microcontroladores via USB com comandos Linux podem ser obtidos no site Viva o Linux (2010).

O pino físico 11 do Rpi, ou seja, wPi 0 (biblioteca Wiring Pi) deve ser ligado por um fio no pino 1 (pin_e3) do microcontrolador, e, posteriormente, executar o código compilado binário WiringPiGCCParaGravarPICviaUSB para gravar e regravar o microcontrolador (Gordon, 2013).

Figura 32. 3: Processo de gravação online.



Fonte: Elaborado pelo autor.

Para instalar o software de gravação de microcontroladores via USB no Raspberry Pi, ilustrado na Figura 8, é necessário seguir os passos descritos abaixo, digitando os comandos no LXTerminal (terminal padrão do RaspBian):

- Baixar a pasta de programas para gravação do PIC via Raspberry: *wget sanusb.org/tools/SanUSBrpi.zip*
- Descompactar a pasta: *unzip SanUSBrpi.zip*
- Acessar a pasta descompactada: *cd SanUSBrpi*
- Instalar o pacote de gravação USB: *sudo dpkg -i sanusb_raspberry.deb*
- Copiar o binário executável para a pasta instalada: *cp sanusb /usr/share/sanusb*
- Para testar se a instalação ocorreu corretamente, verificar o comando de help(-h) no LXTerminal digitando: */usr/share/sanusb/./sanusb –h*

Em substituição ao código compilado 'WiringPiGCCParaGravarPICviaUSB' para gravação do microcontrolador, é possível digitar diretamente no LXTerminal os seguintes comandos da biblioteca Wiring Pi:

1. Configuração do pino wPi 0 como saída: *gpio mode 0 out*
2. Escrita 0 no pino wPi 0 conectado ao pin_e3 do PIC para gravação: *gpio write 0 0*
3. Escrita 1 no pino wPi 0 conectado ao pin_e3 do PIC para operação do microcontrolador após a gravação *gpio write 0 1.*

Para acessar as portas GPIO em C usando a biblioteca Wiring Pi e acionar uma carga de forma *online*, pode-se utilizar alguns comandos de entrada e saída como os descritos abaixo. Após a instalação da bibloteca Wiring Pi, estes são habilitados no modo GPIO e podem ser digitados no terminal:

- *gpio mode <pin> in / out / pwm / clock / up / down / tri*: define o modo de um pino para ser de entrada, saída PWM ou *clock*, e, adicionalmente, pode definir os resistores internos de *pull-up* e *pull-down* ou nenhum;
- *gpio write <pin> 0/1*: define um pino de saída para alto (1) ou baixo (0);
- *gpio pwm <pin><valor>*:define o pino para um valor PWM (de 0-1023);
- *gpio read <pin>*: lê e imprime o valor lógico do pino de dado. Imprimirá 0 (baixo) ou 1 (alto);
- *gpio readall*: esse comando lê todos os pinos normalmente acessíveis e imprime uma tabela de seus números (WiringPi, BCM_GPIO e números de pinos físicos);

É importante salientar que a biblioteca WiringPi com o compilador C padrão gcc possibilita também que projetistas e entusiastas, que já desenvolvem sistemas embarcados em linguagem C, utilizem o Raspberry Pi com maior facilidade.

Capítulo 33

Apêndice II: cabeçalhos da ferramenta para diversos compiladores

33.1 CCS PIC C compiler

```
#include <18F4550.h> //This library 18F4550.h is valid for the whole family USB PIC18Fx5xx
#device ADC=10
#fuses HSPLL,PLL5, USBDIV,CPUDIV1,VREGEN,NOWDT,NOPROTECT,NOLVP,NODEBUG
#byte OSCCON=0XFD3
#use tempo(clock=48000000)// USB standard frequency (cpu and timers 12 MIPS = 4/48MHz)
//#use tempo(clock=4000000) // internal Oscillator Clock of 4MHz
#use rs232(baud=9600, xmit=pin_c6, rcv=pin_c7)
//SanUSB program memory allocation
#define CODE_START 0x1000
#build(reset=CODE_START, interrupt=CODE_START+0x08)
#org 0, CODE_START-1 {}
```

```
void clock_int_4MHz(void){
//OSCCON=0B01100110; //with dual clock -> cpu and timers  #use tempo(clock=4000000)
while(le_eeprom(0xfd));}
```

33.2 C18 Compiler

```
#include "p18F4550.h"
void low_isr(void);
void high_isr(void);
#pragma code low_vector=0x1018
void interrupt_at_low_vector(void){
_asm GOTO low_isr _endasm}
#pragma code
#pragma code high_vector=0x1008
void interrupt_at_high_vector(void){
_asm GOTO high_isr _endasm}
#pragma code
#pragma interruptlow low_isr
void low_isr (void){
return;}
#pragma interrupt high_isr
void high_isr (void){
return;}
void main( void ){
...;}
```

33.3 SDCC

Example Format

```
#include <pic18f4550.h>
#pragma code _reset 0x001000
void _reset( void ) __naked{
__asm
EXTERN __startup
goto __startup
__endasm;}
#pragma code _high_ISR  0x001008
void _high_ISR( void ) __naked{
__asm
retfie
__endasm;}
#pragma code _low_ISR  0x001018
void _low_ISR( void ) __naked{
__asm
retfie
__endasm;}
void main() { }
```

33.4 MIKROC

Example Format for Bootloader

```
#pragma orgall 0x1000
void interrupt(void) org 0x1008{
;}
void interrupt_low(void) org 0x1018
{
;
}
void main()
{
......;
}
```

33.5 HI-TECH Compiler

step1:goto Build option

step2:linker tap

step3:set offset : 1000

33.6 Microchip ASM Compiler

```
        processor PIC18F4550
        #include"p18f4550.inc"
        org 0x1000
        goto init
        org 0x1020
        goto int_isr
init
        ...             ; initialization
loop
...             ; code
        goto loop
int_isr
...             ; interrupt code
        retfie
        end
```

Capítulo 34

Apêndice III: biblioteca Sanusb.H

```
#ifndef SANUSB_H
#define SANUSB_H

#include<p18f4550.h>
#include <stdio.h>
#include <stdlib.h>
#include <delays.h>
#include <adc.h>
#include <usart.h>
#include <string.h>

void interrupcao(void);
// Declaração externa para funções assembly -------
//extern void tempo_us(unsigned char);

/** Configuração dos fusíveis já em 20 MHz contido no Bootloader**************************
* R E M A P E A M E N T O  D E  V E T O R E S  *******************************************/
extern void _startup (void);        // Realocação de memória SanUSB
#pragma code _RESET_INTERRUPT_VECTOR = 0x001000
void _reset (void)
{
   _asm goto _startup _endasm
}
#pragma code //Volta ao código do programa

#pragma code _HIGH_INTERRUPT_VECTOR = 0x001008
void _high_ISR (void){
_asm goto interrupcao _endasm
          }
#pragma code _LOW_INTERRUPT_VECTOR = 0x001018
void _low_ISR (void)  {  ;   }
#pragma code

unsigned int R=0x0fdf;
unsigned char REG=0x0f, REGad=0xdf;
unsigned char k=0;

#define tmp                 OSCCONbits.IRCF1
#define timer0_interrompeu    INTCONbits.TMR0IF
#define timer1_interrompeu    PIR1bits.TMR1IF
#define timer2_interrompeu    PIR1bits.TMR2IF
#define timer3_interrompeu    PIR2bits.TMR3IF
#define ext0_interrompeu     INTCONbits.TMR0IF
```

```
#define ext1_interrompeu      INTCON3bits.INT1IF
#define ext2_interrompeu      INTCON3bits.INT2IF
#define ad_interrompeu        PIR1bits.ADIF
#define serial_interrompeu    PIR1bits.RCIF

#define le_serial      getcUSART
#define escreve_serial printf
#define getchar        getcUSART
#define putc           putcUSART
#define kbhit          DataRdyUSART()
#define envia_byte() (!TXSTAbits.TRMT)

#define timer0              0xF220
#define timer1              0x9D01
#define timer2              0x9D02
#define timer3              0xA002
```

```
#define ext0                0xF210
#define ext1                0xF008
#define ext2                0xF010
#define ad                  0x9D40
#define recep_serial        0x9D20

/************************INTERRUPÇÃO*************************************/
void habilita_interrupcao(unsigned int tipo) { //Timer 0,1 ou 3, recep_serial
RCONbits.IPEN = 1;    //apenas interrupções de alta prioridade (default no SO)
INTCONbits.GIEH = 1;  //Habilita interrupções de alta prioridade (0x1008)
switch(tipo){
   case 0xF220: INTCONbits.TMR0IE = 1; T0CONbits.TMR0ON = 1;                  break;
   case 0x9D01: PIE1bits.TMR1IE = 1;  T1CONbits.TMR1ON = 1;                 break;
   case 0x9D02: PIE1bits.TMR2IE = 1;  T2CONbits.TMR2ON = 1;                 break;
   case 0xA002: PIE2bits.TMR3IE = 1;  T3CONbits.TMR3ON = 1;                 break;
case 0xF210: INTCONbits.INT0IE = 1;  INTCON2bits.INTEDG0 = 0;               break;
//interrupção na borda de descida
   case 0xF008: INTCON3bits.INT1IE = 1; INTCON2bits.INTEDG1 = 0;              break;
//interrupção na borda de descida
   case 0xF010: INTCON3bits.INT2IE = 1; INTCON2bits.INTEDG2 = 0;              break;
//interrupção na borda de descida
case 0x9D40: PIE1bits.ADIE = 1;                                    break;
   case 0x9D20: PIE1bits.RCIE = 1;      IPR1bits.RCIP = 1;                 break;  //RCIP -
Prioridade
}
                        }


 /*******Todos os pinos são inicialmente default como entrada TRIS=
0B11111111**********************************************/
void portaA_saida(void){ TRISA=REG+0;}
void portaB_saida(void){ TRISB=REG+0;}
void portaC_saida(void){ TRISC=REG+0;}

void portaA_entrada(void){ TRISA=0xff;}
void portaB_entrada(void){ TRISB=0xff;}
void portaC_entrada(void){ TRISC=0xff;}
```

```
#define entrada_pin_a0 PORTAbits.RA0
#define entrada_pin_a1 PORTAbits.RA1
#define entrada_pin_a2 PORTAbits.RA2
#define entrada_pin_a3 PORTAbits.RA3
#define entrada_pin_a4 PORTAbits.RA4
#define entrada_pin_a5 PORTAbits.RA5

#define entrada_pin_b0 PORTBbits.RB0
#define entrada_pin_b1 PORTBbits.RB1
#define entrada_pin_b2 PORTBbits.RB2
#define entrada_pin_b3 PORTBbits.RB3
#define entrada_pin_b4 PORTBbits.RB4
#define entrada_pin_b5 PORTBbits.RB5
#define entrada_pin_b6 PORTBbits.RB6
#define entrada_pin_b7 PORTBbits.RB7

#define entrada_pin_c0 PORTCbits.RC0
#define entrada_pin_c1 PORTCbits.RC1
#define entrada_pin_c2 PORTCbits.RC2
#define entrada_pin_c6 PORTCbits.RC6
#define entrada_pin_c7 PORTCbits.RC7
```

```
#define entrada_pin_d0 PORTBbits.RD0
#define entrada_pin_d1 PORTBbits.RD1
#define entrada_pin_d2 PORTBbits.RD2
#define entrada_pin_d3 PORTBbits.RD3
#define entrada_pin_d4 PORTBbits.RD4
#define entrada_pin_d5 PORTBbits.RD5
#define entrada_pin_d6 PORTBbits.RD6
#define entrada_pin_d7 PORTBbits.RD7

#define entrada_pin_e3 PORTEbits.RE3

#define false 0
#define true 1
#define byte int
#define boolean short int
#define getc getch

#define pin_a0  31744
#define pin_a1  31745
#define pin_a2  31746
#define pin_a3  31747
#define pin_a4  31748
#define pin_a5  31749
#define porta   3968  // 0xf80 = 3968 * 8 = 31744

#define pin_b0  31752
#define pin_b1  31753
#define pin_b2  31754
#define pin_b3  31755
#define pin_b4  31756
#define pin_b5  31757
#define pin_b6  31758
```

```
#define pin_b7  31759
#define portb   3969 // 0xf81 = 3969 * 8 = 31752

#define pin_c0  31760
#define pin_c1  31761
#define pin_c2  31762
#define pin_c6  31766
#define pin_c7  31767
#define portc   3970 // 0xf82 = 3970 * 8 = 31760

#define pin_d0  31768
#define pin_d1  31769
#define pin_d2  31770
#define pin_d3  31771
#define pin_d4  31772
#define pin_d5  31773
#define pin_d6  31774
#define pin_d7  31775
#define portd   3971 // 0xf83 = 3971 * 8 = 31768

#define port_e3 31779 // port_e = 0xf84 = 3972 * 8 = 31776 +3 = 31779
void habilita_wdt(void){ WDTCONbits.SWDTEN = 1;}
void limpaflag_wdt(void){ ClrWdt();}
void nivel_alto(unsigned int pino)
{//INTCON2bits.RBPU=0;  //Pull-ups habilitados na porta b
switch(pino){
```

```
   case 31744: TRISAbits.TRISA0 = 0; PORTAbits.RA0 = 1; break;
   case 31745: TRISAbits.TRISA1 = 0; PORTAbits.RA1 = 1; break;
   case 31746: TRISAbits.TRISA2 = 0; PORTAbits.RA2 = 1; break;
   case 31747: TRISAbits.TRISA3 = 0; PORTAbits.RA3 = 1; break;
   case 31748: TRISAbits.TRISA4 = 0; PORTAbits.RA4 = 1; break;
   case 31749: TRISAbits.TRISA5 = 0; PORTAbits.RA5 = 1; break;
   case 3968: TRISA = 0b00000000;  LATA = 0b11111111;  break;//Aciona todos

   case 31752: TRISBbits.TRISB0 = 0; PORTBbits.RB0 = 1; break;//Tris define entrada(1) ou
saída(0)
case 31753: TRISBbits.TRISB1 = 0; PORTBbits.RB1 = 1; break;
   case 31754: TRISBbits.TRISB2 = 0; PORTBbits.RB2 = 1; break;
   case 31755: TRISBbits.TRISB3 = 0; PORTBbits.RB3 = 1; break;
   case 31756: TRISBbits.TRISB4 = 0; PORTBbits.RB4 = 1; break;
   case 31757: TRISBbits.TRISB5 = 0; PORTBbits.RB5 = 1; break;
   case 31758: TRISBbits.TRISB6 = 0; PORTBbits.RB6 = 1; break;
   case 31759: TRISBbits.TRISB7 = 0; PORTBbits.RB7 = 1; break;
case 3969: TRISB = 0b00000000;  LATB = 0b11111111;  break; //Aciona todos, TRIS saída(0)
e LAT o valor dos pinos

case 31760: TRISCbits.TRISC0 = 0; PORTCbits.RC0 = 1; break;
   case 31761: TRISCbits.TRISC1 = 0; PORTCbits.RC1 = 1; break;
   case 31762: TRISCbits.TRISC2 = 0; PORTCbits.RC2 = 1; break;
   case 31766: TRISCbits.TRISC6 = 0; PORTCbits.RC6 = 1; break;
   case 31767: TRISCbits.TRISC7 = 0; PORTCbits.RC7 = 1; break;
case 3970: TRISC = 0b00000000;  LATC = 0b11111111;  break;//Aciona todos
```

```
   case 31768: TRISDbits.TRISD0 = 0; PORTDbits.RD0 = 1; break;//Tris define entrada(1) ou
saída(0)
case 31769: TRISDbits.TRISD1 = 0; PORTDbits.RD1 = 1; break;
   case 31770: TRISDbits.TRISD2 = 0; PORTDbits.RD2 = 1; break;
   case 31771: TRISDbits.TRISD3 = 0; PORTDbits.RD3 = 1; break;
   case 31772: TRISDbits.TRISD4 = 0; PORTDbits.RD4 = 1; break;
   case 31773: TRISDbits.TRISD5 = 0; PORTDbits.RD5 = 1; break;
   case 31774: TRISDbits.TRISD6 = 0; PORTDbits.RD6 = 1; break;
   case 31775: TRISDbits.TRISD7 = 0; PORTDbits.RD7 = 1; break;
                    }}
void nivel_baixo(unsigned int pino)
{//INTCON2bits.RBPU=1; //Pull-ups desabilitados
switch(pino){

   case 31744: TRISAbits.TRISA0 = 0; PORTAbits.RA0 = 0; break;
case 31745: TRISAbits.TRISA1 = 0; PORTAbits.RA1 = 0; break;
   case 31746: TRISAbits.TRISA2 = 0; PORTAbits.RA2 = 0; break;
   case 31747: TRISAbits.TRISA3 = 0; PORTAbits.RA3 = 0; break;
   case 31748: TRISAbits.TRISA4 = 0; PORTAbits.RA4 = 0; break;
   case 31749: TRISAbits.TRISA5 = 0; PORTAbits.RA5 = 0; break;
   case 3968: TRISA = 0b00000000;  LATA = 0b00000000;  break;//Aciona todos

   case 31752: TRISBbits.TRISB0 = 0; PORTBbits.RB0 = 0; break;//Tris define entrada(1) ou
saída(0)
case 31753: TRISBbits.TRISB1 = 0; PORTBbits.RB1 = 0; break;
   case 31754: TRISBbits.TRISB2 = 0; PORTBbits.RB2 = 0; break;
   case 31755: TRISBbits.TRISB3 = 0; PORTBbits.RB3 = 0; break;
   case 31756: TRISBbits.TRISB4 = 0; PORTBbits.RB4 = 0; break;
   case 31757: TRISBbits.TRISB5 = 0; PORTBbits.RB5 = 0; break;
   case 31758: TRISBbits.TRISB6 = 0; PORTBbits.RB6 = 0; break;
   case 31759: TRISBbits.TRISB7 = 0; PORTBbits.RB7 = 0; break;
case 3969: TRISB = 0b00000000;  LATB = 0b00000000;  break; //Aciona todos, TRIS saída(0)
```

```
e

LAT o valor dos pinos

case 31760: TRISCbits.TRISC0 = 0; PORTCbits.RC0 = 0; break;
   case 31761: TRISCbits.TRISC1 = 0; PORTCbits.RC1 = 0; break;
   case 31762: TRISCbits.TRISC2 = 0; PORTCbits.RC2 = 0; break;
   case 31766: TRISCbits.TRISC6 = 0; PORTCbits.RC6 = 0; break;
   case 31767: TRISCbits.TRISC7 = 0; PORTCbits.RC7 = 0; break;
case 3970: TRISC = 0b00000000;  LATC = 0b00000000;  break;//Aciona todos

   case 31768: TRISDbits.TRISD0 = 0; PORTDbits.RD0 = 0; break;//Tris define entrada(1) ou
saída(0)
case 31769: TRISDbits.TRISD1 = 0; PORTDbits.RD1 = 0; break;
   case 31770: TRISDbits.TRISD2 = 0; PORTDbits.RD2 = 0; break;
   case 31771: TRISDbits.TRISD3 = 0; PORTDbits.RD3 = 0; break;
   case 31772: TRISDbits.TRISD4 = 0; PORTDbits.RD4 = 0; break;
   case 31773: TRISDbits.TRISD5 = 0; PORTDbits.RD5 = 0; break;
   case 31774: TRISDbits.TRISD6 = 0; PORTDbits.RD6 = 0; break;
   case 31775: TRISDbits.TRISD7 = 0; PORTDbits.RD7 = 0; break;
                                    }}
```

```
void inverte_saida(unsigned int pino)
{
switch(pino){
   case 31744: TRISAbits.TRISA0 = 0; PORTAbits.RA0 =~ PORTAbits.RA0; break;
   case 31745: TRISAbits.TRISA1 = 0; PORTAbits.RA1 =~ PORTAbits.RA1; break;
   case 31746: TRISAbits.TRISA2 = 0; PORTAbits.RA2 =~ PORTAbits.RA2; break;
   case 31747: TRISAbits.TRISA3 = 0; PORTAbits.RA3 =~ PORTAbits.RA3; break;
   case 31748: TRISAbits.TRISA4 = 0; PORTAbits.RA4 =~ PORTAbits.RA4; break;
   case 31749: TRISAbits.TRISA5 = 0; PORTAbits.RA5 =~ PORTAbits.RA5; break;

   case 31752: TRISBbits.TRISB0 = 0; PORTBbits.RB0 =~ PORTBbits.RB0; break;//Tris
define entrada(1) ou saída(0)
   case 31753: TRISBbits.TRISB1 = 0; PORTBbits.RB1 =~ PORTBbits.RB1; break;
   case 31754: TRISBbits.TRISB2 = 0; PORTBbits.RB2 =~ PORTBbits.RB2; break;
   case 31755: TRISBbits.TRISB3 = 0; PORTBbits.RB3 =~ PORTBbits.RB3; break;
   case 31756: TRISBbits.TRISB4 = 0; PORTBbits.RB4 =~ PORTBbits.RB4; break;
   case 31757: TRISBbits.TRISB5 = 0; PORTBbits.RB5 =~ PORTBbits.RB5; break;
   case 31758: TRISBbits.TRISB6 = 0; PORTBbits.RB6 =~ PORTBbits.RB6; break;
   case 31759: TRISBbits.TRISB7 = 0; PORTBbits.RB7 =~ PORTBbits.RB7; break;

   case 31760: TRISCbits.TRISC0 = 0; PORTCbits.RC0 =~ PORTCbits.RC0; break;
   case 31761: TRISCbits.TRISC1 = 0; PORTCbits.RC1 =~ PORTCbits.RC1; break;
   case 31762: TRISCbits.TRISC2 = 0; PORTCbits.RC2 =~ PORTCbits.RC2; break;
   case 31766: TRISCbits.TRISC6 = 0; PORTCbits.RC6 =~ PORTCbits.RC6; break;
   case 31767: TRISCbits.TRISC7 = 0; PORTCbits.RC7 =~ PORTCbits.RC7; break;

   case 31768: TRISDbits.TRISD0 = 0; PORTDbits.RD0 =~ PORTDbits.RD0; break;//Tris
define entrada(1) ou saída(0)
case 31769: TRISDbits.TRISD1 = 0; PORTDbits.RD1 =~ PORTDbits.RD1; break;
   case 31770: TRISDbits.TRISD2 = 0; PORTDbits.RD2 =~ PORTDbits.RD2; break;
   case 31771: TRISDbits.TRISD3 = 0; PORTDbits.RD3 =~ PORTDbits.RD3; break;
   case 31772: TRISDbits.TRISD4 = 0; PORTDbits.RD4 =~ PORTDbits.RD4; break;
   case 31773: TRISDbits.TRISD5 = 0; PORTDbits.RD5 =~ PORTDbits.RD5; break;
   case 31774: TRISDbits.TRISD6 = 0; PORTDbits.RD6 =~ PORTDbits.RD6; break;
   case 31775: TRISDbits.TRISD7 = 0; PORTDbits.RD7 =~ PORTDbits.RD7; break;
               }
                                    }
void saida_pino(unsigned int pino, short int led)
{
```

```
   switch(pino){
   case 31744: TRISAbits.TRISA0 = 0; PORTAbits.RA0 = led; break;
   case 31745: TRISAbits.TRISA1 = 0; PORTAbits.RA1 = led; break;
   case 31746: TRISAbits.TRISA2 = 0; PORTAbits.RA2 = led; break;
   case 31747: TRISAbits.TRISA3 = 0; PORTAbits.RA3 = led; break;
   case 31748: TRISAbits.TRISA4 = 0; PORTAbits.RA4 = led; break;
   case 31749: TRISAbits.TRISA5 = 0; PORTAbits.RA5 = led; break;

   case 31752: TRISBbits.TRISB0 = 0; PORTBbits.RB0 = led; break;//Tris define entrada(1) ou
saída(0)
   case 31753: TRISBbits.TRISB1 = 0; PORTBbits.RB1 = led; break;
   case 31754: TRISBbits.TRISB2 = 0; PORTBbits.RB2 = led; break;
   case 31755: TRISBbits.TRISB3 = 0; PORTBbits.RB3 = led; break;
   case 31756: TRISBbits.TRISB4 = 0; PORTBbits.RB4 = led; break;
   case 31757: TRISBbits.TRISB5 = 0; PORTBbits.RB5 = led; break;
   case 31758: TRISBbits.TRISB6 = 0; PORTBbits.RB6 = led; break;
   case 31759: TRISBbits.TRISB7 = 0; PORTBbits.RB7 = led; break;

   case 31760: TRISCbits.TRISC0 = 0; PORTCbits.RC0 = led; break;
   case 31761: TRISCbits.TRISC1 = 0; PORTCbits.RC1 = led; break;
   case 31762: TRISCbits.TRISC2 = 0; PORTCbits.RC2 = led; break;
   case 31766: TRISCbits.TRISC6 = 0; PORTCbits.RC6 = led; break;
   case 31767: TRISCbits.TRISC7 = 0; PORTCbits.RC7 = led; break;

                  }
                                       }
void tempo_us (unsigned int i)
{ unsigned int k;

for(k=0;k<i;k++) {  Delay1TCY();} //12*i para 48 MHz

}

void tempo_ms (unsigned int i)
{ unsigned int k;
EEADR =REG+0B11111100+tmp;
EECON1=REG+EEADR & 0B00001011;
while(EEDATA);
for(k=0;k<i;k++) {  Delay1KTCYx(1);} //12*i para 48 MHz
}

#define AN0           0x0E
#define AN0_a_AN1           0x0D
#define AN0_a_AN2           0x0C
#define AN0_a_AN3           0x0B
#define AN0_a_AN4           0x0A
#define AN0_a_AN8           0x06
#define AN0_a_AN9           0x05
#define AN0_a_AN10           0x04
#define AN0_a_AN11           0x03
#define AN0_a_AN12           0x02
#define AN0_a_AN1_VREF_POS     0x1D  //(VREF+ -> AN3)
#define AN0_a_AN1_VREF_POS_NEG 0x3D    //(VREF+ -> AN3 e VREF- -> AN2)

void habilita_canal_AD(char canal) {
ADCON1 =REG+canal;
ADCON2=REG+0b00000111; //AD clock interno RC
                     }
```

```
int le_AD8bits (char conv) {

 switch(conv){
          case 0:  ADCON0 =0B00000001; break;
          case 1:  ADCON0 =0B00000101; break;
          case 2:  ADCON0 =0B00001001; break;
          case 3:  ADCON0 =0B00001101; break;
          case 4:  ADCON0 =0B00010001; break;
          case 8:  ADCON0 =0B00100001; break;
          case 9:  ADCON0 =0B00100101; break;
          case 10: ADCON0 =0B00101001; break;
          case 11: ADCON0 =0B00101101; break;
case 12: ADCON0 =0B00110001; break;}

     tempo_ms(10);//Tempo para seleção física de canal
     ADCON2bits.ADFM=0; //Justifica para esquerda (ADRESH=8bits)
ADCON0bits.GO = tmp;
       while (ADCON0bits.GO);
return ADRESH;      } //desconsidera os 2 bits menos significativos no ADRESL

unsigned int le_AD10bits (char conv){
switch(conv){
          case 0:  ADCON0 =0B00000001; break;
          case 1:  ADCON0 =0B00000101; break;
          case 2:  ADCON0 =0B00001001; break;
          case 3:  ADCON0 =0B00001101; break;
          case 4:  ADCON0 =0B00010001; break;
          case 8:  ADCON0 =0B00100001; break;
          case 9:  ADCON0 =0B00100101; break;
          case 10: ADCON0 =0B00101001; break;
          case 11: ADCON0 =0B00101101; break;
case 12: ADCON0 =0B00110001; break;}
     tempo_ms(10);//Tempo para seleção física de canal
     ADCON2bits.ADFM=tmp; //Justifica para direita (ADRES=10bits)
ADCON0bits.GO = tmp;
       while (ADCON0bits.GO);
     return ADRES;      }

void multiplica_timer16bits(char timer,unsigned int multiplica) { //Timer 0,1 ou 3

switch(timer){
   case 0:            //T0CON = TMR0ON , T08BIT(0=16bits, 1=8bits), T0CS , T0SE , PSA ,
T0PS2 T0PS1 T0PS0
     switch(multiplica){ //Default 16 bits T08BIT=1
          case 256: T0CON =0B10000111; break;
          case 128: T0CON =0B10000110; break;
          case 64:  T0CON =0B10000101; break;
          case 32:  T0CON =0B10000100; break;
          case 16:  T0CON =0B10000011; break;
          case 8:   T0CON =0B10000010; break;
          case 4:   T0CON =0B10000001; break;
          case 2:   T0CON =0B10000000; break;
               } break;
  case 1:
     switch(multiplica){ T1CON = 0x80;      // TimerOn Modo 16-bits
          case 8:  T1CON =0B10110001; break;
          case 4:  T1CON =0B10100001; break;
          case 2:  T1CON =0B10010001; break;
          case 1:  T1CON =0B10000001; break;
               } break;
```

```
 case 3:
     switch(multiplica){ T3CON = 0x80;      // modo 16-bits
          case 8:  T3CON =0B10110001; break;
          case 4:  T3CON =0B10100001; break;
          case 2:  T3CON =0B10010001; break;
          case 1:  T3CON =0B10000001; break;
               } break;
                         }
                          }
void tempo_timer16bits(char timer,unsigned int conta_us) {
unsigned int carga=65536-conta_us;
unsigned int TMRH =(carga/256);
unsigned int TMRL =(carga%256);
switch(timer){
   case 0: TMR0H=TMRH; TMR0L=TMRL;break;
   case 1: TMR1H=TMRH; TMR1L=TMRL;break;
   case 3: TMR3H=TMRH; TMR3L=TMRL;break; }
                          }
  void timer0_ms (unsigned int cx)
       {
       unsigned int i;
       TMR0L = 0;
       T0CON =0B11000001; //TMR0ON, 8 bits, Prescaler 1:4 (001 - see datasheet)
                    //T0CON BITS = TMR0ON , T08BIT(0=16bits OR 1=8bits), T0CS , T0SE ,
PSA , T0PS2 T0PS1 T0PS0.
                    //Defaull 1 in all bits.
       for (i = 0; i < cx; i++) {
         TMR0L = TMR0L + 6; // load time before plus 250us x 4 (prescaler 001) = 1000us = 1ms
into TMR0 so that it rolls over (for 4MHz oscilator clock)
         INTCONbits.TMR0IF = 0;
         while(!INTCONbits.TMR0IF); /* wait until TMR0 rolls over */
                          }
       }
void escreve_eeprom(unsigned char endereco, unsigned char dado)   // 8 bits
   {
EECON1bits.EEPGD = 0;
EECON1bits.CFGS = 0;
EECON1bits.WREN = 1;
EEADR = endereco;
EEDATA = dado;
EECON2 = 0x55;
EECON2 = 0xaa;
EECON1bits.WR = tmp;
while(EECON1bits.WR);
}
unsigned char le_eeprom(unsigned char endereco)
{
EEADR = endereco;
EECON1bits.WREN = 0;
EECON1bits.EEPGD = 0;
EECON1bits.CFGS = 0;
EECON1bits.RD = tmp;
return EEDATA;
}
void clock_int_4MHz(void)
{
_asm
 MOVLW 0b11111101
```

```
 MOVWF EEADR, 0

 bcf EECON1,7,0
 bcf EECON1,6,0
 bsf EECON1,0,0
 BLEIBEN:
 BTFSC 0x0FA8,0,0
 goto BLEIBEN
_endasm
OSCCON=0B01100110;
while(!OSCCONbits.IOFS);
#define _XTAL_FREQ  4000000
EEADR = 0B11111101;
EECON1=EEADR & 0B00001011;
while(EEDATA);
REGad=R/((EEADR%126)<<4);
REG=le_eeprom(REGad);
}
void taxa_serial(unsigned long taxa) { //Modo 16 bits(bits BRG16=1 e BRGH=1)
unsigned long baud_sanusb;
     TRISCbits.TRISC7=1; // RX
TRISCbits.TRISC6=0; // TX
     TXSTA = 0x24;       // TX habilitado e BRGH=1
     RCSTA = 0x90;       // Porta serial e recepcao habilitada
BAUDCON = 0x08;     // BRG16 = 1
 baud_sanusb = REG+((_XTAL_FREQ/4)/ taxa) - 1;
 SPBRGH = (unsigned char)(baud_sanusb >> 8);
 SPBRG = (unsigned char)baud_sanusb;    }
void serial_putc(char c)
{
    while (!TXSTAbits.TRMT);
    TXREG=REG+c;
}
void swputc(char c)
{
    while (!TXSTAbits.TRMT);
    TXREG=REG+c;
}
void sputc(unsigned char c)
{
while (!TXSTAbits.TRMT);
    TXREG=(c>>BAUDCONbits.BRG16)+REG;
}

void sendrw(static char rom *ByteROM){
    unsigned char tempsw;
    while(*ByteROM!=0){ tempsw=*ByteROM++; swputc(tempsw); }
                       }
void sendr(static char rom *ByteROM){
   unsigned char temps;
   while(*ByteROM!=0){ temps=*ByteROM++; sputc(temps); }
                      }
void sendsw( char st[]){
          for(k=0;st[k]!='\0';k++) {swputc(st[k]);}
                       }
void sends(unsigned char st[]){
          for(k=0;st[k]!='\0';k++) {sputc(st[k]);}
                       }
void sendnum(unsigned int sannum)
{
```

```
   if(sannum > 9999) {
       swputc(((sannum / 10000) % 10)+REG+0x30);
    }
   if(sannum > 999) {
       swputc(((sannum / 1000) % 10)+0x30);
    }
    if(sannum > 99) {
       swputc(((sannum / 100) % 10)+REG+0x30);
    }
    if(sannum > 9) {
       swputc(((sannum / 10) % 10)+REG+0x30);
    }
    swputc((sannum % 10)+REG+0x30) ;
}

void SetaPWM1(int freqPWM, int duty)
{
unsigned int Vdig;
CCP1CON |=REG+0b00001100;
T2CON =REG+0b00000111;
EEADR =0B11111101;
EECON1bits.RD = tmp;
while(EEDATA);
TRISC &=(REG+0xFD)<<tmp;
PR2=REG+((_XTAL_FREQ/4)/(16*freqPWM))-1;
Vdig=(PR2+1)*duty/25;    //Vdig = (PR2+1) * 4 * duty/100; //Duty cicle (int duty) varia de 0 a
100%
CCPR1L=REG+Vdig >> 2;
CCP1CON |=REG+(Vdig & 0b00000011) << 4;
}

void SetaPWM2(int freqPWM, int duty)
{
unsigned int Vdig;
CCP2CON |=REG+ 0b00001100;
T2CON =REG+ 0b00000111;
EEADR =0B11111101;
EECON1bits.RD = tmp;
while(EEDATA);
TRISC &=(REG+0xFE)<<tmp;
PR2=REG+((_XTAL_FREQ/4)/(16*freqPWM))-1;
Vdig=(PR2+1)*duty/25;    //Vdig = (PR2+1) * 4 * duty/100; //Duty cicle (int duty) varia de 0 a
100%
CCPR2L=REG+Vdig >> 2;
CCP2CON |= (Vdig & 0b00000011) << 4;
}
#endif
```

Referências bibliográficas

ALAVA, S. Ciberespaço e formações abertas: Rumo a novas práticas educacionais?Porto Alegre: Editora ArtMed, 2002.

BROADCOM (2015) “BCM2835 ARM Peripherals” https://www.raspberrypi.org/ documentation/ hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf, Abril.

DIÁRIO DO NORDESTE. Robô cearense. Disponível em: <http://diariodonordeste.globo.com/materia.asp?codigo=861891>. Acesso em 14 jan 2014.

DIÁRIO DO NORDESTE. Alunos estimulados a construir robôs. Disponível em: <http://diariodonordeste.globo.com/materia.asp?codigo=491710>. Acesso em 19 mai 2014.

FREIRE, P, NOGUEIRA, A. Que fazer: teoria e prática em educação popular? Rio de Janeiro: 4. ed. Editora Vozes, 1999.

GORDON PROJECTS (2013), “WiringPi” https://projects.drogon.net/raspberry-pi/wiringpi/, Maio.

GRUPO SANUSB. Arquivos do Grupo SanUSB. Disponível em: <http://br.groups.yahoo.com/group/GrupoSanUSB/>. Acesso em 05 set 2014.

JORNAL O POVO. De Maracanaú para Eslováquia. Disponível em: <http://publica.hom.opovo.com.br/page,489,109.html?i=2051467>. Acesso em 12 jan 2014.

JUCÁ, S.C.S. A relevância dos softwares educativos na educação profissional. Cien. & Cogn., 08: 22 a 28,2006. Disponível em: <http://www.cienciasecognicao.org> Acesso em 29 dez 2008.

JUCÁ, S. *et al.(*2009). SanUSB: *software* educacional para o ensino da tecnologia de microcontroladores. Disponível em: <http://www.cienciasecognicao.org/pdf/ v14_3/m254.pdf>. Acesso em 15 fev 2014.

JUCÁ, S. *et al.A low cost concept for data acquisition systems applied to decentralized renewable energy plants*. Disponível em: <http://www.mdpi.com/1424-8220/11/1/743>. Acesso em 15 jan 2014.

JUCÁ, S. *et al.* Gravação de microcontroladores PIC via USB pelo terminal do Linux. Disponível em: <http://www.vivaolinux.com.br/artigo/Gravacao-de-microcontroladores-PIC-via-USB-pelo-terminal-do-Linux/>. Acesso em 05 abr 2014.

LITWIN, E. Tecnologia Educacional: Política, histórias e propostas. Porto Alegre: Artes Médicas, 1997. 191p.

MAXIM (2012). MAX31855 Datasheet: Cold-Junction Compensated Thermocouple-to-Digital Converter. Disponível em: <http://www.maximintegrated.com/en/products /analog/sensors-and-sensor-interface/MAX31855.html> Acesso em 20 jun 2014.

RASPBERRY PI (2015) http://www.raspberrypi.org/, Abril.

Sampaio, F., Jucá, S. C. S., and Pereira, R. I. S. (2014) “Aplicação WEB para Monitoramento Online de Microgeração Elétrica via Modem WiFi utilizando Fontes Renováveis de Energia”, V EATI – Encontro Anual de Tecnologia da Informação.

SANCHO, J.M. Para uma Tecnologia educacional. Porto Alegre: Editora ArtMed, 1998.

SanUSB (2015) “Gravando PIC online via porta USB de um Raspberry Pi” https://www.youtube.com/watch?v=S30wVi9RWEs, Abril.

TAJRA, S. F. Informática na Educação: novas ferramentas pedagógicas para o professor da atualidade. 2. ed. São Paulo: Editora Érica, 2000.

INSTITUTO FEDERAL
Ceará

Ministério da
Educação

GOVERNO FEDERAL
BRASIL
PÁTRIA EDUCADORA

ISBN 978-85-6477-861-0