APOSTILA DE

MICROCONTROLADORES

PIC E PERIFÉRICOS

Sandro Jucá

1. INTRODUÇÃO

Um microcontrolador é um sistema computacional completo, no qual estão incluídos internamente uma CPU (*Central Processor Unit*), memórias RAM (dados), *flash* (programa) e E^2PROM, 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.

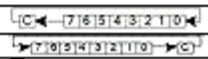
1.1. ASSEMBLY X LINGUAGEM C

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 MPLAB. Para edição e compilação em linguagem C (geração do código HEX), o programa mais utilizado é o PIC C Compiler CCS®.

Os microcontroladores PIC possuem apenas 35 instruções em assembly para a família de 12 bits (PIC12) e 14 bits (PIC16), descritas nas tabelas abaixo, e 77 instruções para a família de 16 bits (PIC18). A tabela abaixo mostra algumas instruções em assembly.

Menemónica		Descrição		us (4MHz)
		Transferência de dados		
MOVLW	k	Mova literal para W (acumulador)	k → W	1
MOVWF	f	Mova W para f (registro de dados)	W → f	1
MOVF	f, d	Mova f	f → d	1
CLRW	-	Clear W (limpar W)	0 → W	1
CLRF	f	Clear f (limpar f)	0 → f	1
SWAPF	f, d	Troca nibbles de f -> d	(d é o registro de destino)	1
		Lógicas e Aritméticas		
ADDLW	k	Adicionar literal a W	W+k → W	1
ADDWF	f, d	Adicionar W a f	W+f → d	1
SUBLW	k	Subtrair W de literal	k-W →W	1
SUBWF	f, d	Subtrair W de f	f-W → d	1
ANDLW	k	AND literal com W	W .AND. k → W	1
ANDWF	f, d	AND W com f	W .AND. f → d	1
IORLW	k	Inclusivo OR de literal com W	W .OR. k → W	1
IORWF	f, d	Inclusivo OR de W com f	W .OR. f → d	1
XORWF	f, d	Exclusivo OR de W com f	W . XOR. f → d	1
XORLW	k	Exclusivo OR de literal com W	W .XOR. k → W	1
INCF	f, d	Incrementar f	f+1 → f	1
DECF	f, d	Decrementar f	f-1 → f	1
RLF	f, d	Rode f p/ esquerda com o carry		1
RRF	f, d	Rode f p/ a direita com o carry		1
COMF	f, d	Complementar o byte f	f → d	1

Operações sobre bits				
BCF	f, b	Bit Clear f (bit de f a '0')	0 → f (b)	1
BSF	f, b	Bit Set f (bit de f a '1')	1 → f (b)	1
Direcionamento do programa				(2) * se existir salto
BTFSC	f, b	Bit Test f, Salte se Clear ('0')	salte se f(b)=0	1(2) *
BTFSS	f, b	Bit Test f, Salte se Set ('1')	salte se f(b)=1	1(2) *
DECFSZ	f, d	Decremente f, salte se der 0	f-1 → d, salte se der 0	1(2) *
INCFSZ	f, d	Incremente f, salte se der 0	f+1 → d, salte se der 0	1(2) *
GOTO	k	Go to address (Ir p/ endereço)	k → PC	2
CALL	k	Chamar subrotina	PC → TOS, k → PC	2
RETURN	-	Retorno de subrotina	TOS → PC	2
RETLW	k	Retorno com literal em W	k → W, TOS → PC	2
RETFIE	-	Retorno de interrupção	TOS → PC, 1 → GIE	2
Outras instruções				
NOP	-	Nenhuma operação		1
CLRWDT	-	Temporizador do Watchdog=0	0 → WDT, 1 → TO, 1 → PD	1
SLEEP	-	Entrar no modo 'sleep'	0 → WDT, 1 → TO, 0 → PD	1

Como pode ser visto, 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 industriais). 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.

1.2. 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 por você anteriormente.

- 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 industrial, a linguagem C para PIC 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 nível, 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.

1.3 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.

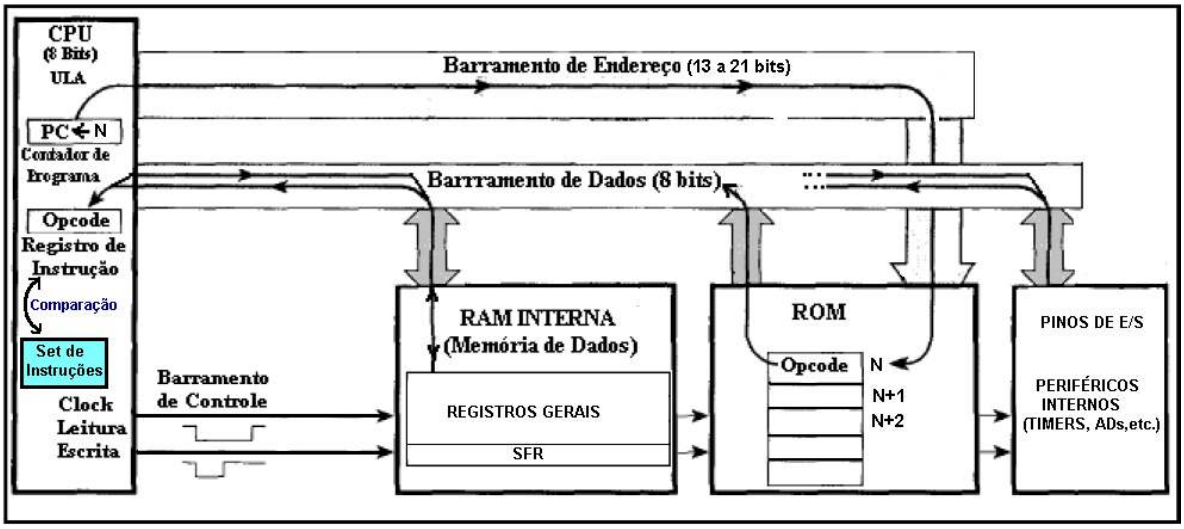
1.4. O CONTADOR DE PROGRAMA (PC)

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 8K words 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.

1.5. 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.

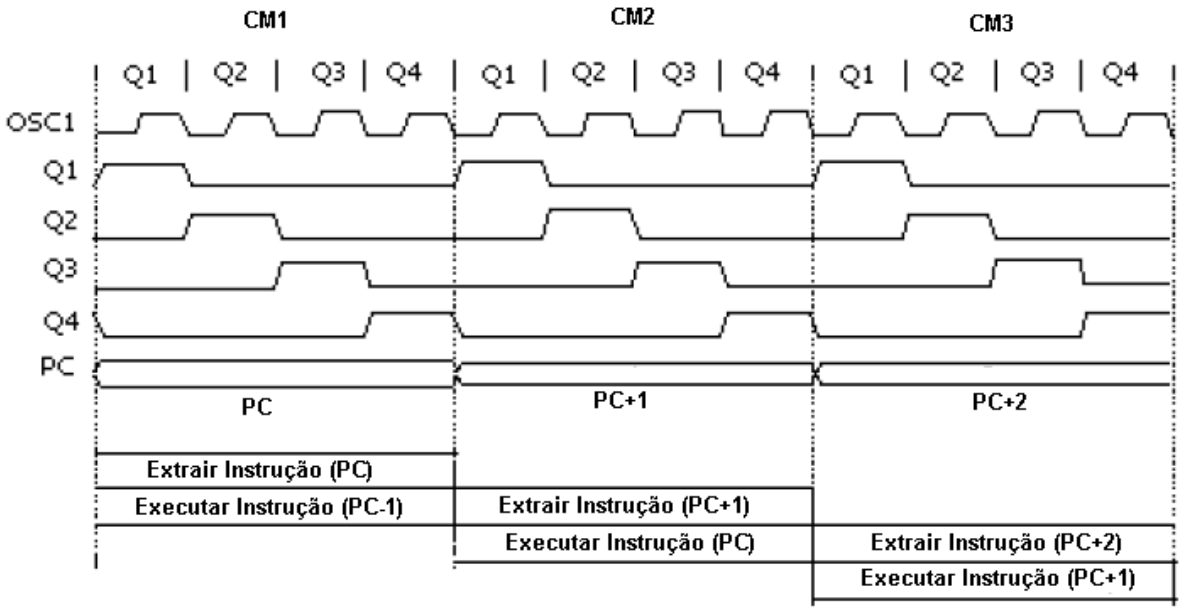
1.6. A PILHA (STACK)

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.

1.7. 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.

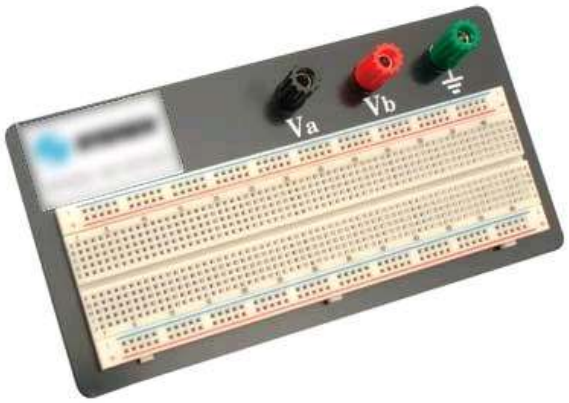
Nos microcontroladores PIC, um ciclo de máquina (CM) possui quatro fases de clock que são Q1, Q2, Q3 e Q4. Dessa forma, para um clock externo de 4MHz, temos um ciclo de máquina (CM=4 x 1/F) igual a 1µs.



O Contador de Programa (PC) é incrementado automaticamente na fase Q1 do ciclo de máquina e a instrução seguinte é resgatada da memória de programa e armazenada no registro de instruções da CPU no ciclo Q4. Ela é decoficada e executada no próximo ciclo, no intervalo de Q1 e Q4. Essa característica de buscar a informação em um ciclo de máquina e executá-la no próximo, ao mesmo tempo em que outra instrução é "buscada", é chamada de *PIPELINE (sobreposição)*. Ela permite que quase todas as instruções sejam executadas em apenas um ciclo de máquina, gastando assim 1 µs (para um clock de 4 MHz) e tornando o sistema muito mais rápido. As únicas exeções referem-se às instruções que geram "saltos" no contador de programa, como chamadas de funções em outro local da memória de programa e os retornos dessas funções.

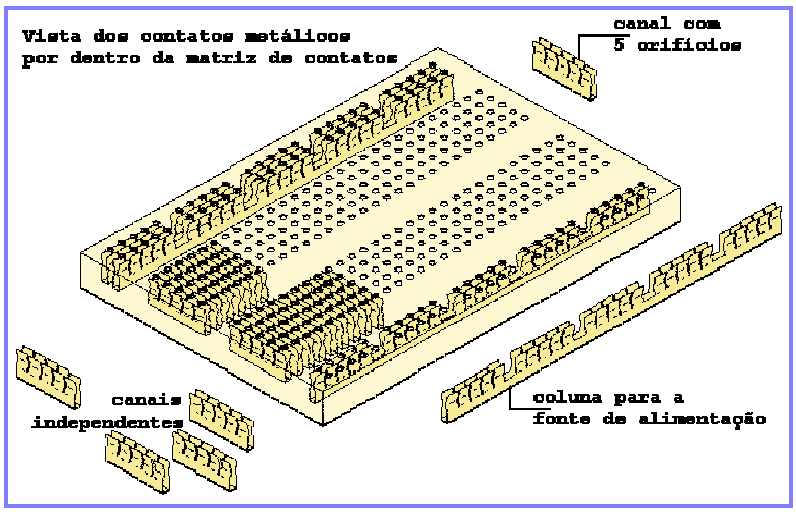
1.8. MATRIZ DE CONTATOS OU *PROTOBOARD*

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 abaixo, 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).



Protoboard

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 de acordo como é ilustrado na figura abaixo.



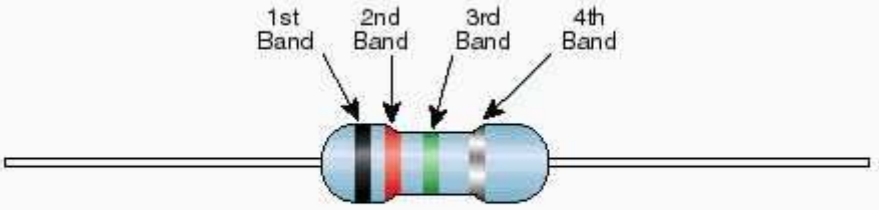
Estrutura de uma *protoboard*

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.

1.9. RESISTORES

Os resistores geralmente são feitos de carbono. Para identificar qual a resistência de um resistor específico, comparamos ele com a seguinte tabela:

Standard EIA Color Code Table 4 Band: ±2%, ±5%, and ±10%



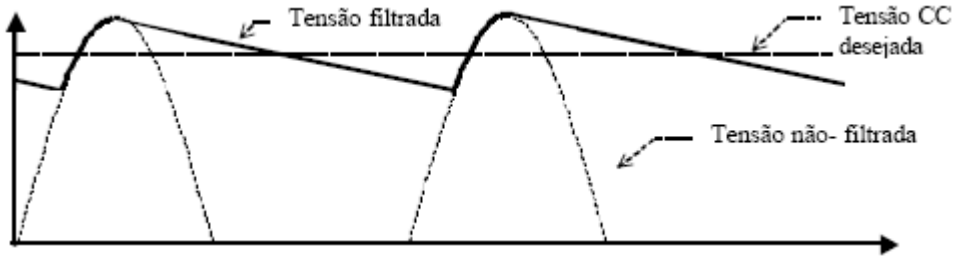
Color	1st Band (1st figure)	2nd Band (2nd figure)	3rd Band (multiplier)	4th Band (tolerance)
Black	0	0	10^0	
Brown	1	1	10^1	
Red	2	2	10^2	±2%
Orange	3	3	10^3	
Yellow	4	4	10^4	
Green	5	5	10^5	
Blue	6	6	10^6	
Violet	7	7	10^7	
Gray	8	8	10^8	
White	9	9	10^9	
Gold			10^{-1}	±5%
Silver			10^{-2}	±10%

No caso da imagem, o resistor é de $2x10^5\Omega \pm 5\%$.

1.10. CAPACITORES

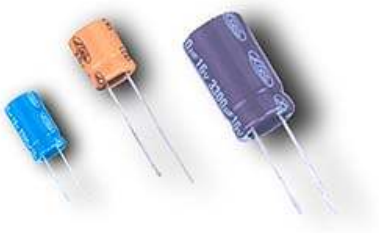
Capacitor ou condensador é um componente que armazena energia num campo elétrico. consistem em dois eletrodos ou placas que armazenam cargas opostas. Estas duas placas são condutoras e são separadas por um isolante ou por um dielétrico. Eles são utilizados desde armazenar bits nas memórias voláteis dinâmicas (DRAM) dos computadores, até corrigir o fator de potência de indústrias fornecendo reatância capacitiva para compensar a reatância indutiva provocada por bobinas e motores elétricos de grande porte.

A função mais comum é filtrar ruídos em circuitor elétricos e estabilizar as fontes, absorvendo os picos e preenchendo os vales de tensão. Os capacitores descarregados são um curto e carregados abrem o circuito, por isso são utilizados também para isolar fontes CC.



Os capacitores podem ser carregados e descarregados muito rapidamente, por isso são utilzados também no flash eletrônico em uma câmera fotográfica, onde pilhas carregam o capacitor do flash durante vários segundos, e então o capacitor descarrega toda a carga no bulbo do flash quase que instantaneamente gerando o alto brilho. Isto pode tornar um capacitor grande e carregado extremamente perigoso. Eles são utilizados também em paralelo com motores elétricos para fornecer energia para que as bobinas energizadas possam vencer a inércia quando os motores são ligados.

As Unidades de Medida de capacitância são Farad (F), Microfarad (μF), Nanofarad (nF) e Picofarad (pF). Os capacitores mais comuns são os eletrolíticos, lstrados na figura abaixo, os cerâmicos e os de poliéster.



A figura abaixo mostra a identificação de capacitores cerâmicos.

A figura abaixo mostra a identificação de capacitores de poliéster.

Tabela para identifcação dos valores dos capacitores de poliéster

Cor	Faixas 1 e 2	Faixa 3	Faixa 4	Faixa 5
	0	-	20%	-
	1	x 10	-	250V
	2	x100	-	-
	3	x1000	-	400V
	4	x10000	-	-
	5	x100000	-	630V
	6	x1000000	-	-
	7	-	-	-
	8	-	-	-
	9	-	10%	-

1.11. FONTES DE ALIMENTAÇÃO

As fontes mais comuns em sistemas embarcados com microcontroladores são baterias recarregáveis ou conversores CA-CC como carregadores de celulares.

As baterias ou pilhas são dispositivos que armazenam energia química e a torna disponível 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.

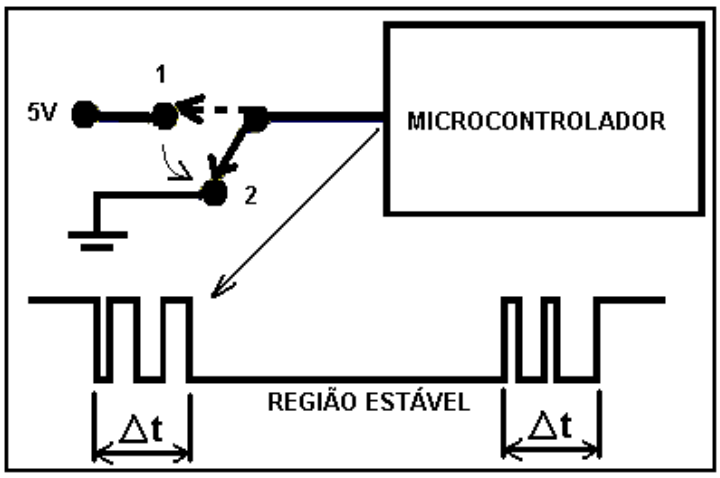
As baterias ditas 12V, por exemplo, devem operar de 13,8V (tensão a plena carga), até 10,5V (tensão de corte), quando 100% de sua capacidade terá sido utilizada, e é este o tempo que deve ser medido como autonomia da bateria.

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 100Ah 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.

1.12. RUÍDO (*BOUNCING)* E FILTRO (*DEBOUNCING*)

Em operações de Liga/Desliga e mudança de nível lógico, 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 (Deb*ouncing*) 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.



PROTOCOLO DE COMUNICAÇÃO USB

A USB, sigla para 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.

Métodos de comunicação USB

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

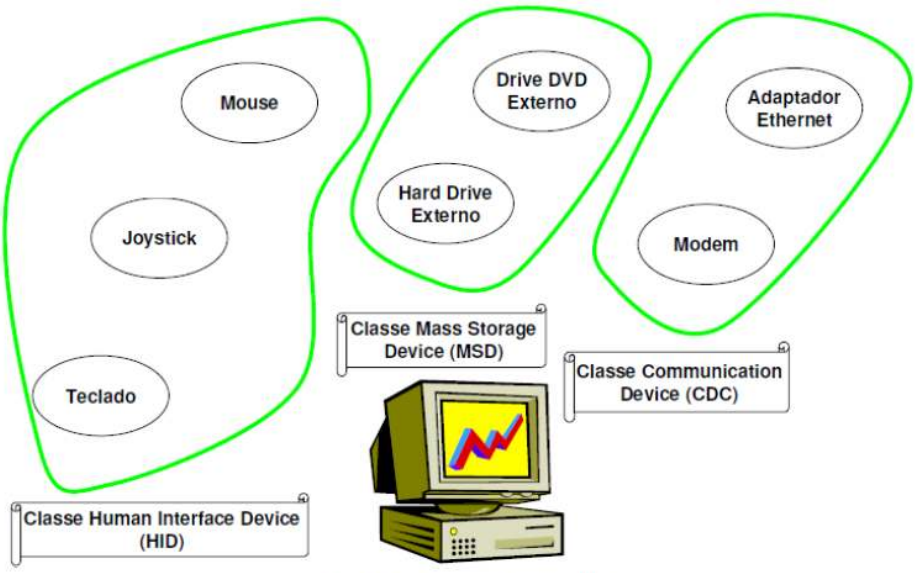
Human Interface Device (HID) - O dispositivo USB é reconhecido automaticamente pelo sistema operacional windows ou linux como um Dispositivo de Interface Humana (HID), não sendo necessário a instalação de driver especiais para a aplicação. Este método apresenta velocidade de comunicação de até 64 kB/s e é utilizado pelo gerenciador de gravação da ferramenta SanUSB no linux. Mais detalhes na video-aula disponível em http://www.youtube.com/watch?v=h6Lw2qeWhlM .

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. Mais detalhes em uma aplicação Windows com protocolo Modbus RTU http://www.youtube.com/watch?v=KUd1JkwGJNk e em uma aplicação de comunicação bidirecional no Linux http://www.youtube.com/watch?v=cRW99T_qa7o.

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 pen-drives, scanners, câmeras digitais. Foi utilizado juntamente com a ferramenta SanUSB para comunicação com software de

supervisão programado em Java. Mais detalhes na video-aula disponível em http://www.youtube.com/watch?v=Ak9RAl2YTr4.

Como foi visto, a comunicação USB é baseada em uma central (host), onde o computador enumera os dispositivos USB conectados a ele. 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.



Categorias de periféricos USB

Nos sitemas operacionais Windows e Linux, o modo mais fácil de comunicar com o PIC USB é o CDC, por uma razão simples, os programas para PCs são baseados na comunicação via porta serial, o que torna o processo ainda mais simples. O método CDC no Linux 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.

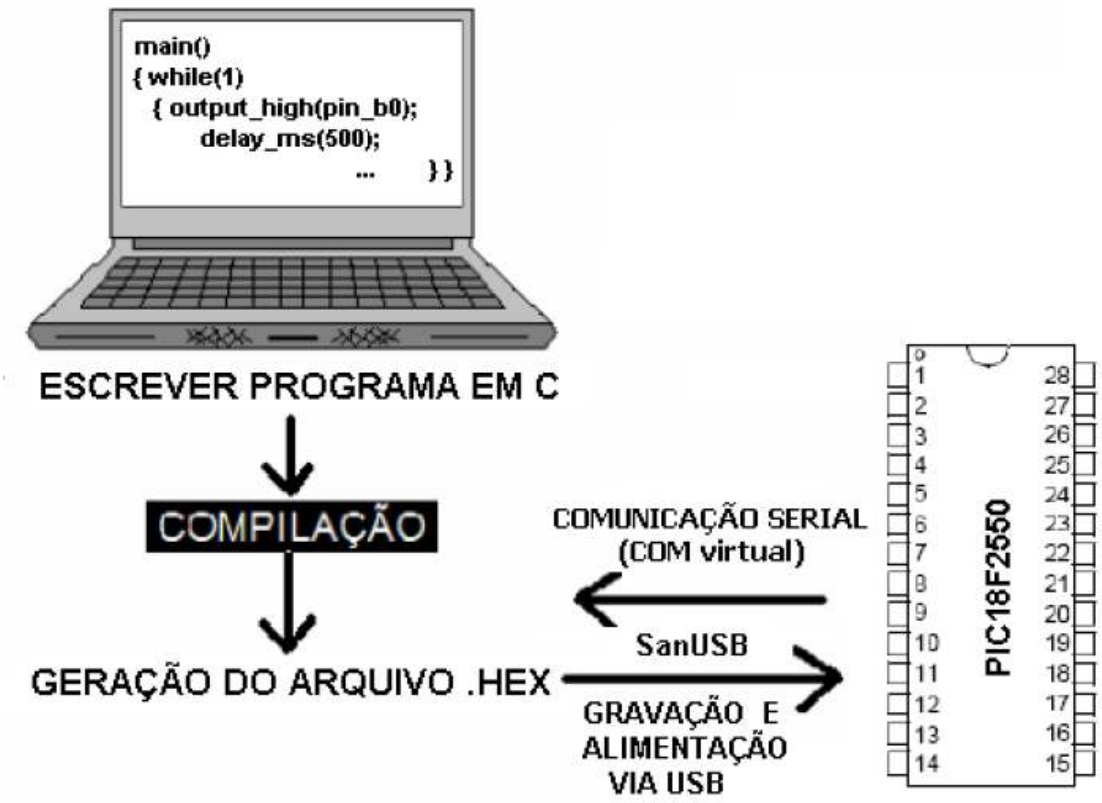
2. FERRAMENTA SanUSB

O circuito de desenvolvimento *SanUSB* é uma ferramenta composta de *software* e *hardware* básico da família PIC18Fxx5x com interface USB. Esta ferramenta livre é capaz de substituir:

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

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

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



2.1. GRAVAÇÃO COM O *SanUSB*

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 GerenciadorWindows.hex ou GerenciadorLinux.hex disponível na pasta completa da ferramenta:

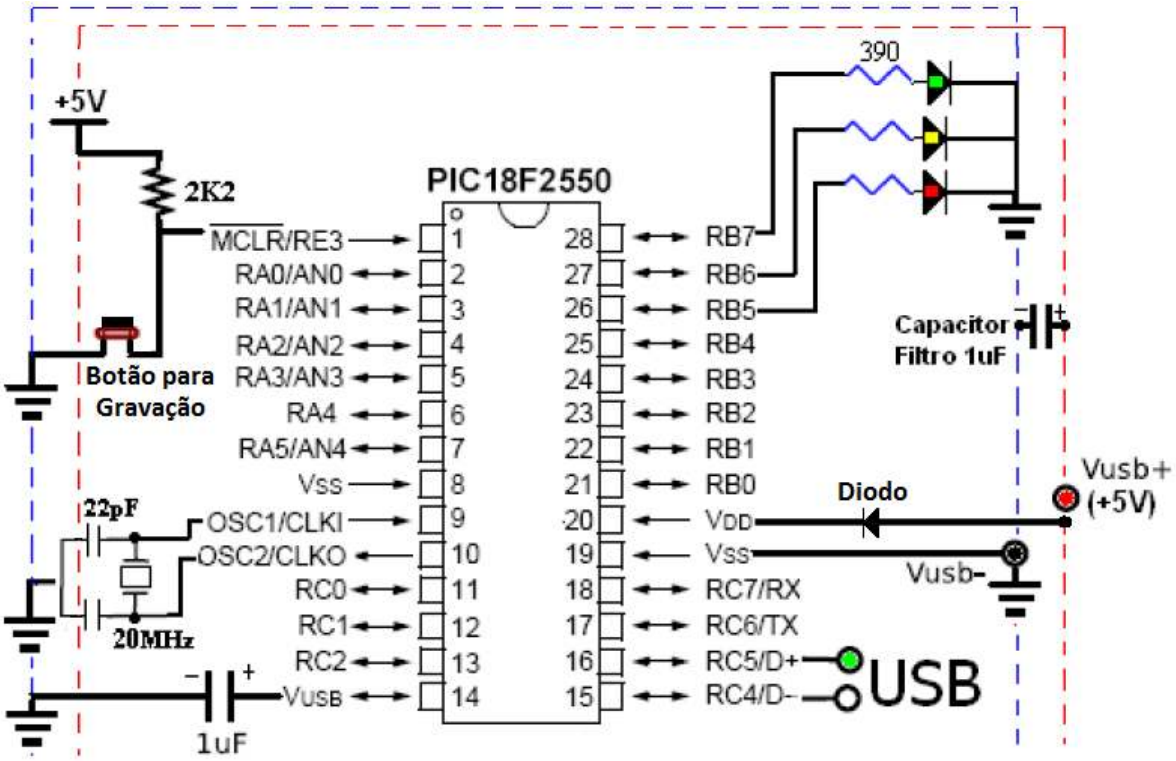
http://www.4shared.com/file/sIZwBP4r/100727SanUSB.html

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 C18, o Hi-Tech e o CCS. Devido à didática das funções e bibliotecas disponíveis para emulação serial, diversos periféricos e *multitasking*, o compilador mais utilizado nos projetos é o CCS. Uma versão demo funcional do compilador utilizado com bibliotecas de suporte a USB é possível baixar em:

http://www.4shared.com/file/Mo6sQJs2/100511Compilador.html .

Para instalar o CCS também no Linux através do Wine basta seguir as seguintes instruções da seguinte vídeo-aula: http://www.youtube.com/watch?v=5-kigedbfxg

Caso grave o novo gerenciador de gravação pela USB GerenciadorLinux.hex, não esqueça de colar o novo arquivo cabeçalho SanUSB.h dentro da pasta *Drivers* localizada na pasta instalada do compilador (C:\Arquivos de programas\PICC\Drivers). A representação básica do circuito *SanUSB* montado em *protoboard* é mostrada a seguir:

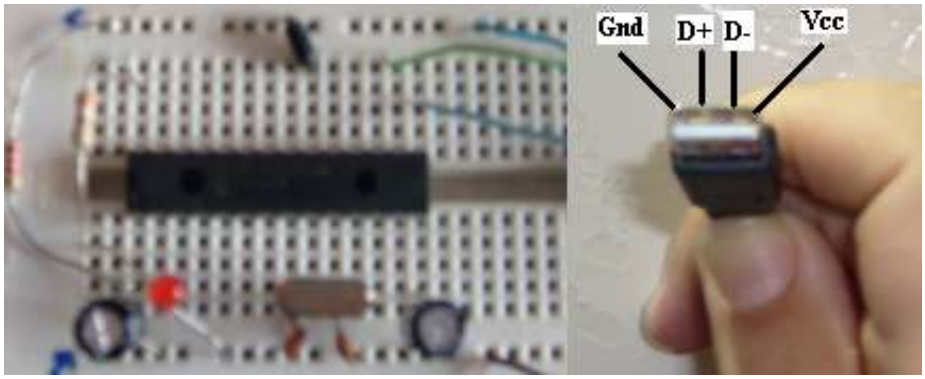


Note que, ao conectar o cabo USB e alimentar o microcontrolador, com o pino 1 no Gnd (0V), através do botão ou de um simples fio, é Estado para Gravação via USB (*led* no pino B7 aceso) e que, após o *reset* com o pino 1 no Vcc (+5V através do resistor fixo de 2K2 sem o *jump*) é Estado para Operação 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 marron ou preto, o D+ é azul ou verde e o D- é amarelo ou branco. Note que a fonte de alimentação do microcontrolador nos pinos 19 e 20 e dos barramentos vermelho (+5V) e azul (Gnd) do circuito provem da própria porta USB do computador. Para ligar o cabo USB no circuito é possível cortá-lo e conectá-lo direto no *protoboard*, com fios rígidos soldados, como também é possível conectar sem cortá-lo, em um

protoboard ou numa placa de circuito impresso, utilizando um conector USB fêmea. O diodo de proteção colocado no pino 20 entre o Vcc da USB e a alimentação do microcontrolador serve para proteger contra corrente reversa caso a tensão da porta USB esteja polarizada de forma inversa.

A figura abaixo mostra a ferramenta SanUSB montada em *protoboard* seguindo o circuito anterior e a posição de cada terminal no conector USB a ser ligado no PC. Cada terminal é conectado diretamente nos pinos do microcontrolador pelos quatro fios correspondentes do cado USB.

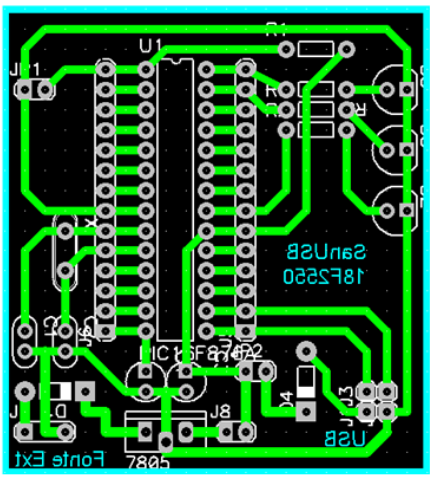


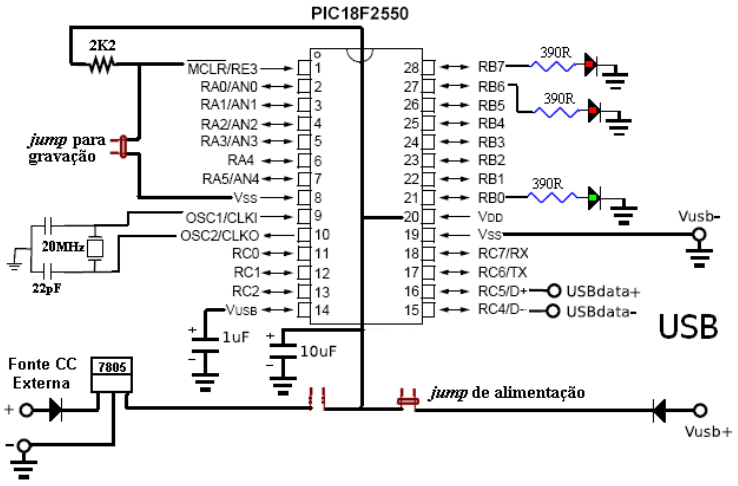
É importante salientar que, para o perfeito funcionamento da gravação via USB, o circuito desta ferramenta deve conter um capacitor de filtro entre 0,1uf e 1uF na alimentação que vem da USB, ou seja, colocado entre os pinos 20 (+5V) e 19 (Gnd).

Caso o sistema microcontrolado seja embarcado como, por exemplo, um robô, um sistema de aquisição de dados ou um controle de acesso, ele necessita de uma fonte de alimentação externa, que pode ser uma bateria comum de 9V ou um carregador de celular. A figura abaixo mostra o PCB, disponível nos Arquivos do Grupo SanUSB, e o circuito para esta ferramenta com entrada para fonte de alimentação externa.

Para quem deseja obter o sistema pronto para um aprendizado mais rápido, é possível também encomendar placas de circuito impresso da ferramenta SanUSB, como a foto da placa abaixo, entrando em contato com o grupo SanUSB através do e-mail: sanusb_laese@yahoo.com.br .

Se preferir confeccionar a placa, basta seguir o circuito abaixo:





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

Durante a programação do microcontrolador basta inserir, no inicio do programa em C, a biblioteca cabeçalho SanUSB (#include <SanUSB.h>) contida dentro da pasta SanUSB_User\Exemplos_SanUSB e que você já adicionou dentro da *Drivers* localizada na pasta instalada do compilador (C:\Arquivos de programas\PICC\Drivers). Essa biblioteca contém instruções do PIC18F2550 para o sistema operacional, configurações de fusíveis e 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 freqüência.

Como a freqüência 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.

```
#include <SanUSB.h>

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

while (1)
  {
  output_toggle(pin_B7); // comuta Led na função principal
  delay_ms(500);
  }
}
```

O programa pisca3 abaixo pisca três *leds* conectados nos pinos B5, B6 e B7.

```
#include <SanUSB.h>

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

while (1)
        {
        output_high(pin_B5); // Pisca Led na função principal
        delay_ms(500);
        output_low(pin_B5);
        output_high(pin_B6);
        delay_ms(500);
        output_low(pin_B6);
        output_high(pin_B7);
        delay_ms(500);
        output_low(pin_B7);
        }
}
```

Os arquivos compilados .hex assim como os firmwares estão disponíveis na pasta 100727SanUSB.

GRAVANDO O MICROCONTROLADOR VIA USB NO WINDOWS

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

1. Copie, para um diretório raiz C ou D, a pasta SanUSB obtida do grupo www.tinyurl.com/SanUSB.

2. Grave no microcontrolador, somente uma vez, com um gravador específico para PIC, o novo gerenciador de gravação pela USB GerenciadorWin.hex disponível na pasta SanUSB.

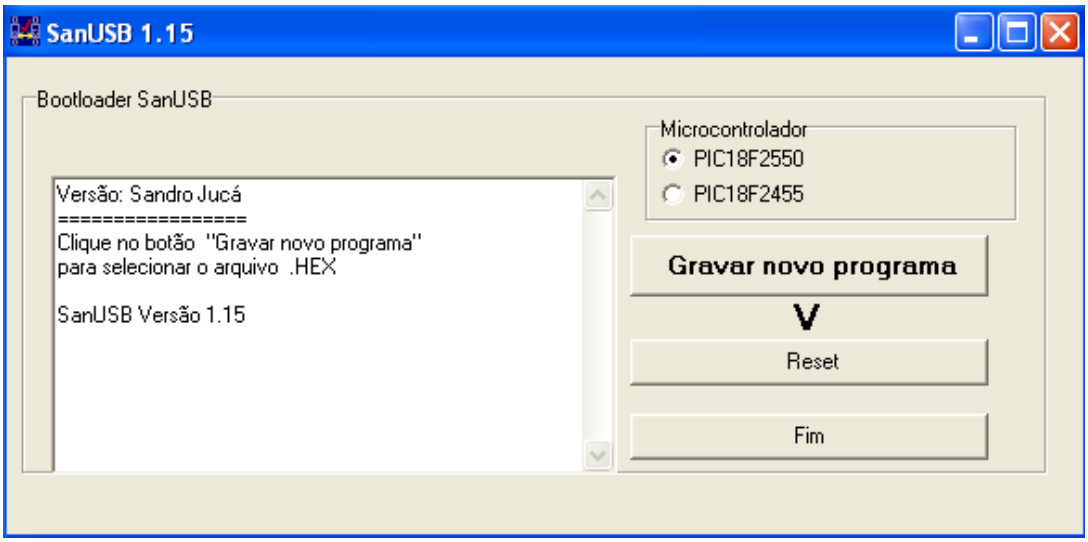
3. Pressione o botão ou conecte o *Jump* de gravação do pino 1 do circuito SanUSB no Gnd para a transferência de programa do PC para o microcontrolador.

4. Conecte o cabo USB entre o PIC e o PC. Se o circuito SanUSB estiver correto acenderá o *led* do pino B7.

Se for o **Windows 7**, vá em propriedades do sistema -> Configurações avançadas do sistema -> Hardware -> Gerenciador de dispositivos e clique com botão direito no driver USB do microcontrolador e atualizar Driver, apontando para a pasta DriverWindows7SanUSB. Após a instalação no Windows 7, clique com o botão direito sobre o ícone de gravação SanUSB  , selecione Propriedades, Compatibilidade e escolher Windows Vista (Service Pack 1).

Se for o **Windows XP**, o windows irá perguntar, somente a primeira vez, onde está o *Driver* de instalação, então escolha a opção *Instalar de uma lista ou local especifico (avançado)*. Após Avançar, escolha a opção Incluir este local na pesquisa e selecione a pasta DriverWinXPSanUSB, onde está o driver sanusb_device. Durante a instalação, o Windows abrirá uma janela sobre a instalação, selecione a opção continuar assim mesmo e o Driver será instalado.

5. Abra o aplicativo **SanUSB**  . Se estiver conectado corretamente, o *led* conectado no pino B7 acende e aparecerá a seguinte tela:



6. Clique em *1. Gravar novo programa* e escolha o programa *.hex* que deseja gravar, como por exemplo, o programa compilado *pisca.hex* da pasta Exemplos_SanUSB. Este programa pisca 3 *leds* conectados nos pinos B5, B6 e B7;

7. Após a gravação do programa, retire o *jump* do pino de gravação e clique em Reset. Pronto o programa estará em operação.

Para programar novamente, basta colocar o *jump* de gravação, retire o *jump* de alimentação, coloque-o novamente e repita os passos anteriores a partir do passo 4. Se a nova programação não funcionar, retire o conector USB do computador e repita os passos anteriores a partir do passo 3.

SanUSB CDC – Emulação de Comunicação Serial no Windows

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 emula uma porta COM RS-232 virtual, através do canal USB 2.0. Dessa forma, é possível se comunicar com caracteres ASCII via USB através de qualquer software monitor serial RS-232 como o HyperTerminal, o SIOW do CCS® Compiler ou o ambiente de programação Delphi®. O driver CDC instalado no PC 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 biblioteca CDC para o programa.c do microcontrolador está dentro da pasta de exemplos, a qual deve estar na mesma pasta onde está o programa.c a ser compilado para a emulação da comunicação serial RS-232. Além disso, o programa.c deve inserir a biblioteca usb_san_cdc.h, como mostra a o exemplo de leitura e escrita em um buffer da EEPROM interna do microcontrolador. As funções CDC mais utilizadas contidas na biblioteca usb_san_cdc.h para comunicação com a COM virtual são:

- **usb_cdc_putc() –** o microcontrolador envia caracteres ASCII emulados via USB.
  Ex.: printf(**usb_cdc_putc**, "\r\nEndereco para escrever: ");

- **usb_cdc_getc()** – retém um caractere ASCII emulado pela USB.

  Ex.: dado = **usb_cdc_getc();** //retém um caractere na variável dado

- **gethex_usb()** – retém um número hexadecimal digitado no teclado.

  Ex.: valor = **gethex_usb();**//retém um número hexadecimal na variável valor

- **usb_cdc_kbhit( )** – Avisa com TRUE (1) se acabou de chegar um novo caractere no buffer de recepção USB do PIC.

  Ex.: **if (usb_cdc_kbhit(1)) {dado = usb_cdc_getc();}**

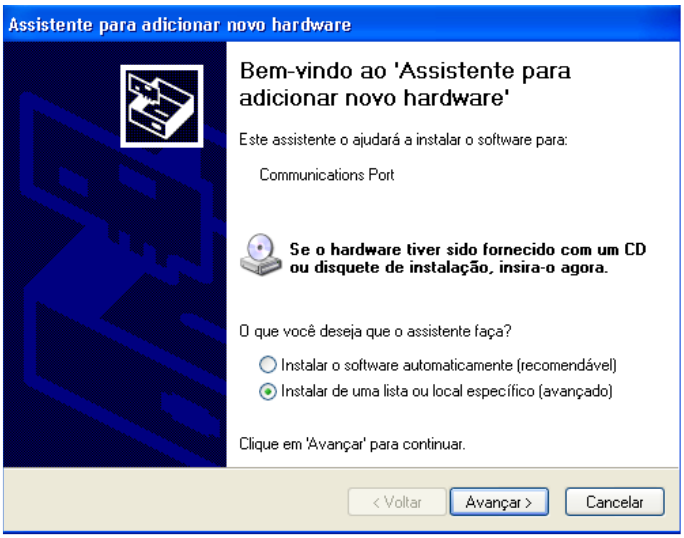
O exemplo abaixo mostra a leitura e escrita em um buffer da EEPROM interna do microcontrolador com emulação da serial através da USB:

```
#include <SanUSB.h>

#include <usb_san_cdc.h>// Biblioteca para comunicação serial

BYTE i, j, endereco, valor;
boolean led;

main() {
clock_int_4MHz();
  usb_cdc_init(); // Inicializa o protocolo CDC
  usb_init();    // Inicializa o protocolo USB
```

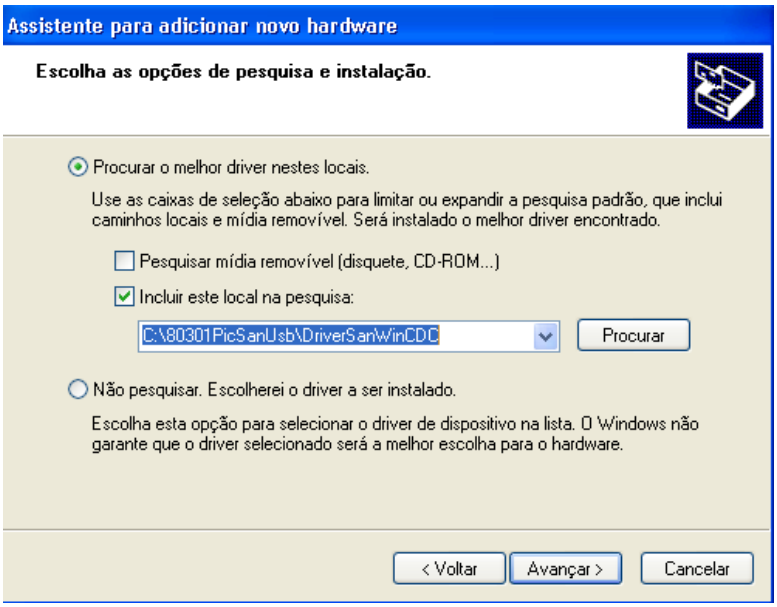
```
  usb_task();  // Une o periférico com a usb do PC

  output_high(pin_b7);  // Sinaliza comunicação USB Ok

while (1) {
      printf(usb_cdc_putc, "\r\n\nEEPROM:\r\n"); // Display contém os primeiros 64 bytes em hex
      for(i=0; i<=3; ++i) {
        for(j=0; j<=15; ++j) {
          printf(usb_cdc_putc, "%2x ", read_eeprom( i*16+j ) );
        }
        printf(usb_cdc_putc, "\n\r");
      }

      printf(usb_cdc_putc, "\r\nEndereco para escrever: ");
      endereco = gethex_usb();
      printf(usb_cdc_putc, "\r\nNovo valor: ");
      valor = gethex_usb();

      write_eeprom( endereco, valor );
led = !led; // inverte o led de teste
output_bit (pin_b7,led);
    }
}
```

Após gravação de um programa que utilize comunicação serial CDC no microcontrolador pelo SanUSB e resetar o microcontrolador, vá, se for o **Windows 7**, em propriedades do sistema -> Configurações avançadas do sistema -> Hardware -> Gerenciador de dispositivos e clique com botão direito no driver CDC do microcontrolador e atualizar Driver, apontando para a pasta DriverCDCXPseven32e64.

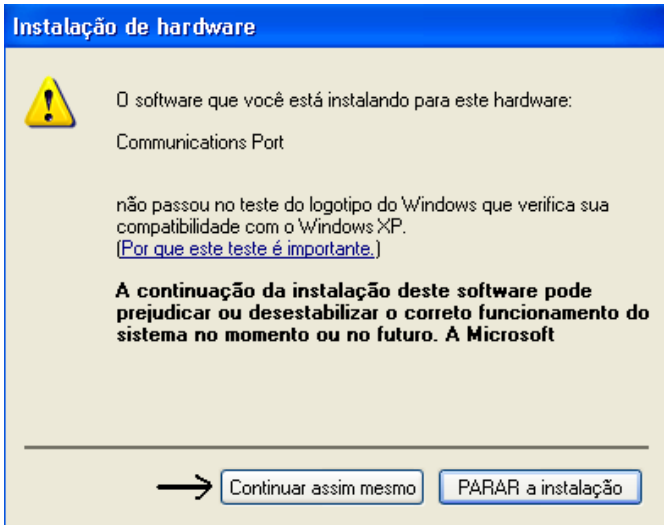
No Windows, após a gravação de um programa que utilize comunicação serial CDC no microcontrolador pelo SanUSB e resetar o microcontrolador, o sistema vai pedir a instalação do driver CDC (se for a primeira vez).



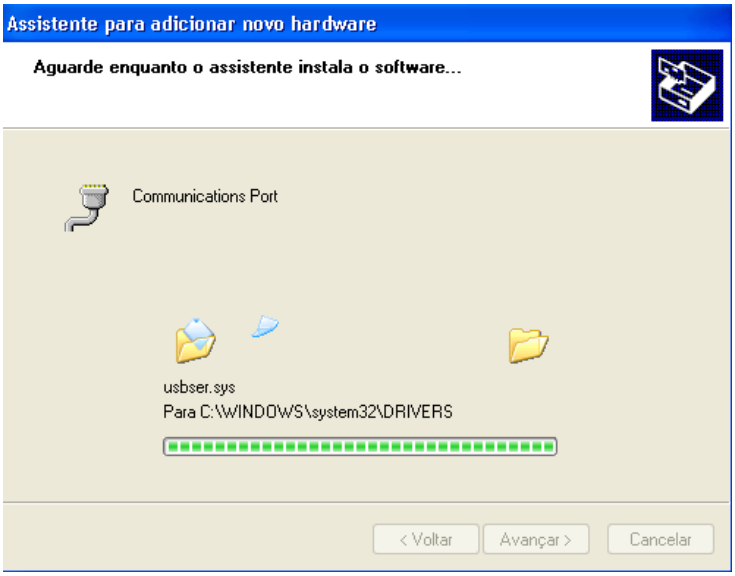
Escolha a opção *Instalar de uma lista ou local especifico (avançado)*. Após Avançar, selecione a *opção Incluir este local na pesquisa* e selecione a pasta DriverSanWinCDC, onde está o driver CDC.



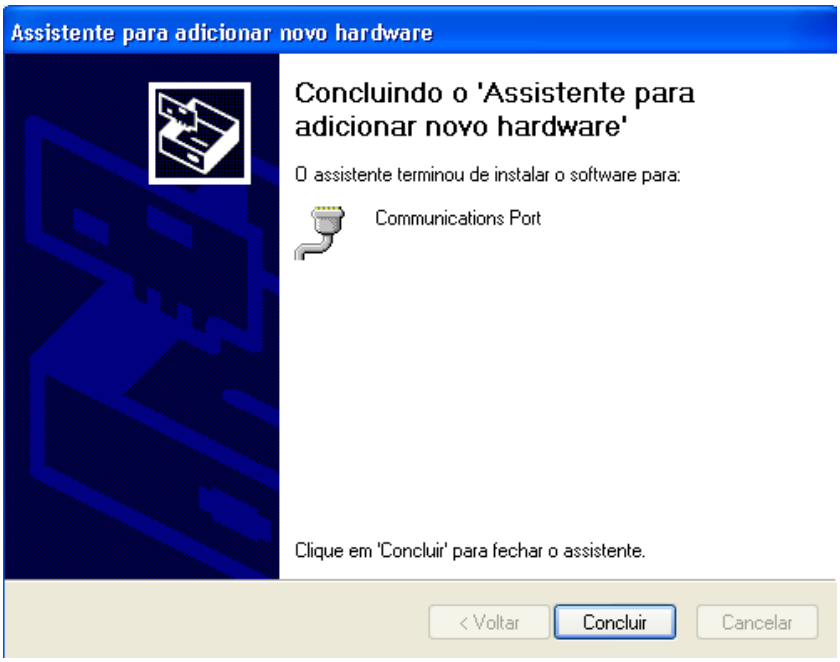
Após Avançar, clique em *Continuar assim mesmo*.



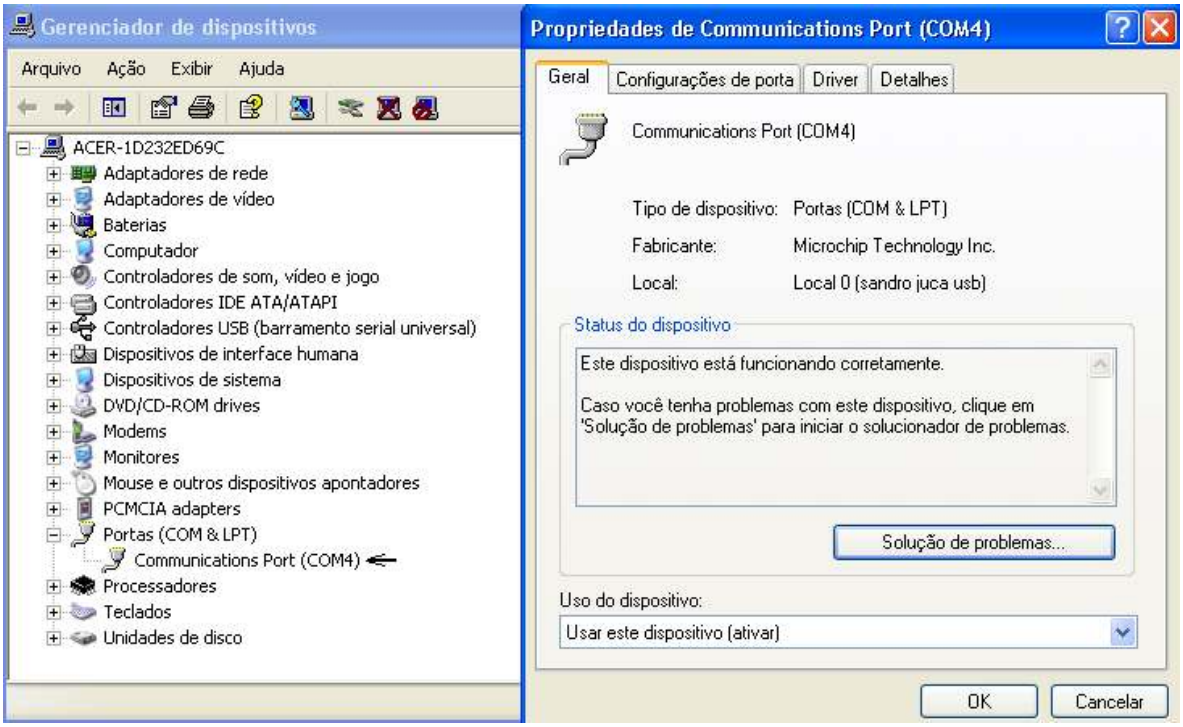
Aguarde enquanto o Driver CDC é instalado no Windows.



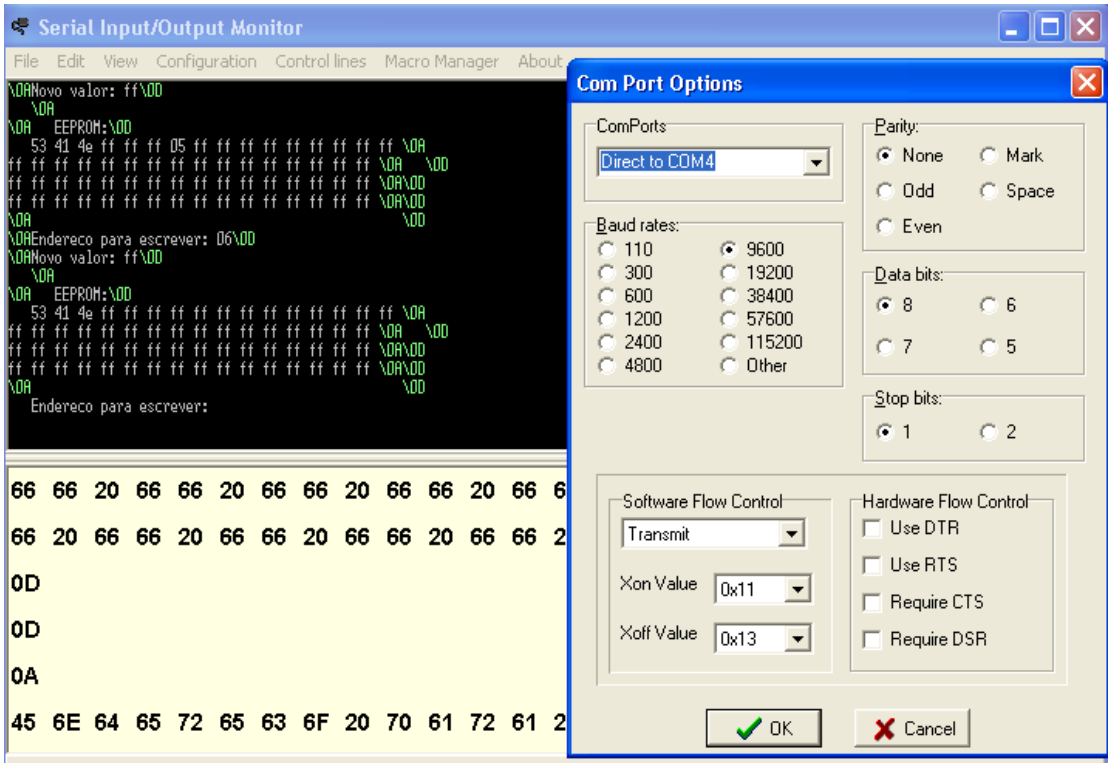
Clique em *Concluir* para terminar a instalação.



Vá em painel de controle -> sistema -> Hardware -> Gerenciador de dispositivos -> Portas (COM & LPT) e confira qual é a porta COM virtual instalada.

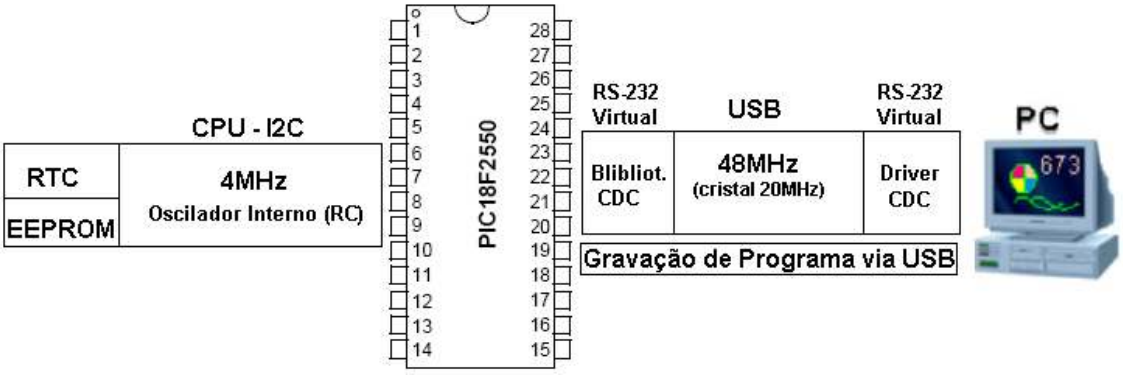


Abrindo qualquer programa monitor de porta serial RS-232, como o SIOW do CCS, direcionando para a COM virtual instalada (COM3,COM4,COM5,etc.) em *configuration > set port options* entraremos em contato com o PIC.



CLOCK DO SISTEMA

Devido à incompatibilidade entre as freqüências necessárias para a gravação e emulação serial via USB e a freqüência padrão utilizada pela CPU, temporizadores e interface I^2C, 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 abaixo.



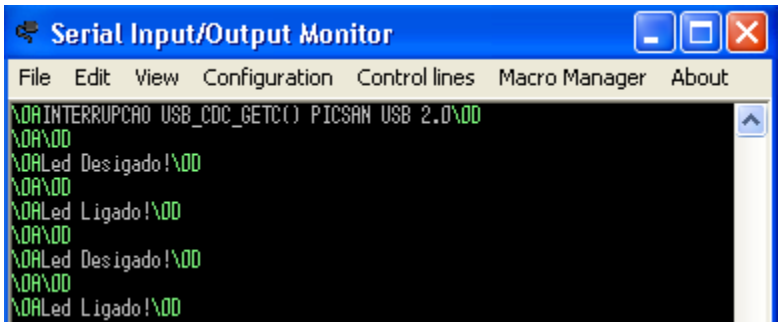
Esse princípio permite que um dado digitado no teclado do computador, trafegue para o microcontrolador em 48 MHz via USB, depois para o relógio RTC ou para a memória EEPROM em 4 MHz via I^2C e vice-versa, como mostra o programa exemplo abaixo.

A interrupção do canal USB do PIC já é utilizada no protocolo CDC. Dessa forma, para utilizar uma função que necessite de atendimento imediato quando um caractere for digitado é necessário inserir a condição ***if (usb_cdc_kbhit(1))*** **{dado=usb_cdc_getc();}** no laço infinito da função principal do programa para verificar, de forma constante, se chegou um novo byte enviado pelo PC. Este comando também evita que o programa fique parado no usb_cdc_getc (que fica esperando um caractere para prosseguir o programa). Veja o programa abaixo, que pisca um led na função principal (pino B6) e comanda o estado se outro led (pino B7) pelo teclado de um PC via USB*:*

```
#include <SanUSB.h>
#include <usb_san_cdc.h>// Biblioteca para comunicação serial virtual

BYTE comando;

main() {
```

```
clock_int_4MHz();//Função necessária para habilitar o dual clock (48MHz para USB e 4MHz para CPU)
usb_cdc_init(); // Inicializa o protocolo CDC
usb_init(); // Inicializa o protocolo USB
usb_task(); // Une o periférico com USB do PC

while (TRUE)
{
if (usb_cdc_kbhit(1)) //avisa se chegou dados do PC
{ //verifica se tem um novo byte no buffer de recepção, depois o kbhit é zerado para próximo byte
comando=usb_cdc_getc(); //se chegou, retém o caractere e compara com 'L' ou 'D' em ASCII

if (comando=='L') {output_high(pin_b7); printf(usb_cdc_putc, "\r\nLed Ligado!\r\n");}
if (comando=='D') {output_low(pin_b7); printf(usb_cdc_putc, "\r\nLed Desigado!\r\n");}
}
output_high(pin_B6); // Pisca Led na função principal
delay_ms(500);
output_low(pin_B6);
delay_ms(500);
} }
```

Após gravar o programa, lembre de direcionar o monitor serial SIOW ou Hyperterminal para a COM virtual instalada (COM3,COM4,COM5,etc.) em *configuration > set port options*.

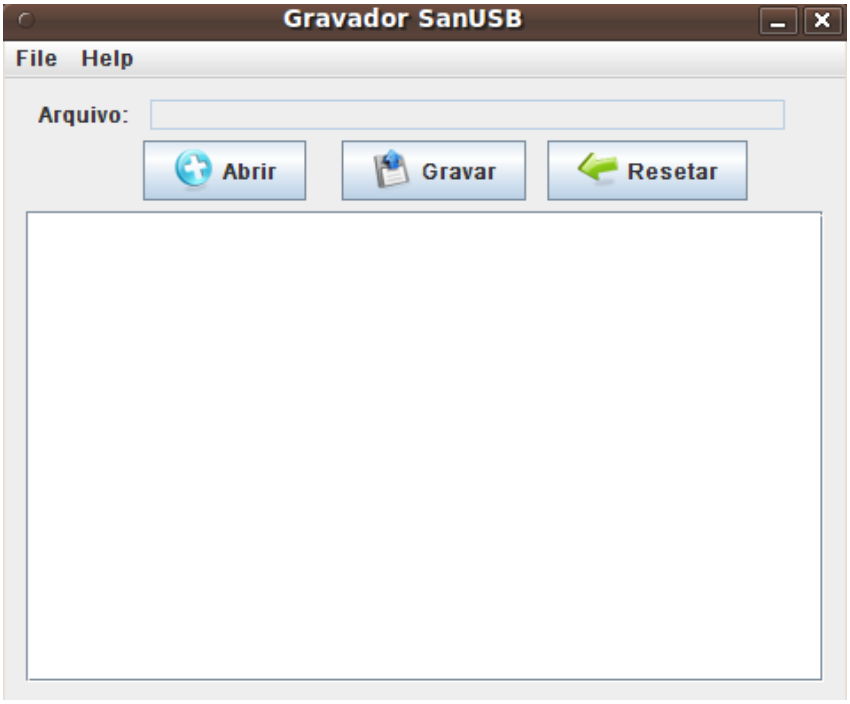


GRAVANDO O MICROCONTROLADOR VIA USB NO LINUX

Esta aplicação é a forma mais simples e direta de gravação, pois com apenas dois cliques no instalador automático.deb SanUSB é possível utilizá-lo em qualquer máquina com Linux (Ubuntu 10.04, equivalente ou posterior). Depois de instalado, o aplicativo é localizado em Aplicativos -> acessórios. O instalador.deb está disponível em:

http://www.4shared.com/file/3mhWZS5g/sanusb.html

A figura abaixo mostra a interface gráfica desenvolvida para gravação direta de microcontroladores via USB:



Neste aplicativo está disponível 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.

GRAVAÇÃO PELO TERMINAL DO LINUX

Outra possibilidade de gravar via USB é de forma manual através de linhas de comando utilizando o terminal do Linux. Inicialmente, é necessário baixar a biblioteca de desenvolvimento **libhid**, a partir da linha de comando:

#sudo apt-get install libhid-dev

Para iniciar a gravação com linhas de comando é importante seguir os seguintes passos:

1. Mova a pasta de arquivos 100727SanUSB obtida do grupo www.tinyurl.com/SanUSB para um diretório do Linux como, por exemplo, a pasta pessoal.

2. Grave no microcontrolador, somente uma vez, com um gravador específico para PIC, o gerenciador de gravação pela USB GerenciadorLinux.hex disponível na pasta SanUSB.

3. Pelo Terminal do Linux acesse a pasta de arquivos SanUSB utilizando alguns comandos básicos descritos abaixo:

cd dir -> entra no diretório dir;
cd dir1/dir2/dir3 -> acessa o sub-diretório dir3;
cd .. -> sai do diretório atual;
cd -> retorna diretório padrão home;
ls -> lista os arquivos de um diretório;
pwd -> mostra o diretório atual;
find /home -name sanusb -> procura no diretório home o arquivo sanusb.

Para gravar no Linux é necessário estar *logado* com permissão para acessar a porta USB como, por exemplo, super-usuário (*sudo su*). A figura abaixo mostra o tela do processo de acesso à pasta e também do processo de gravação do programa exemplo1.hex:

4. Após entrar na pasta SanUSB, acesse o conteúdo do executável *sanusb*, digitando:
. / sanusb-h

5. Coloque o circuito SanUSB em modo de gravação (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.

6. Para gravar no microcontrolador, o firmware desejado, como o exemplo1.hex, deve estar no mesmo diretório do arquivo sanusb, então para a gravação via USB, digita-se:
. / sanusb -w exemplo1.hex

7. Depois de gravar, remova o botão ou *jump* de gravação, então *reset* digitando:
. / sanusb –r

Para gravar no microcontrolador, é possível também, de forma mais simples após remover o *jump* de gravação, digitar na mesma linha os comandos –w (gravar) e –r (resetar):
. / sanusb -w exemplo1.hex –r

Para programar novamente, basta colocar o jump de gravação no pino 1, 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 se *logando* como super-usuário (sudo su) e repita os passos anteriores a partir do passo 4.

SanUSB CDC – Emulação de Comunicação Serial 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 programados em Java. O driver CDC 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 biblioteca CDC para o programa.c do microcontrolador está dentro da pasta de exemplos, a qual deve estar na mesma pasta onde está o programa.c a ser compilado para a emulação da comunicação serial RS-232. Além disso, o programa.c deve inserir a biblioteca usb_san_cdc.h, como mostra a o exemplo de leitura e escrita em um buffer da EEPROM interna do microcontrolador. As funções CDC mais utilizadas contidas na biblioteca usb_san_cdc.h para comunicação com a COM virtual são:

- **usb_cdc_putc() –** o microcontrolador envia caracteres ASCII emulados via USB.
  Ex.: printf(**usb_cdc_putc**, "\r\nEndereco para escrever: ");

- **usb_cdc_getc()** – retém um caractere ASCII emulado pela USB.

Ex.: dado = **usb_cdc_getc();** //retém um caractere na variável dado

- **gethex_usb()** – retém um número hexadecimal digitado no teclado.

  Ex.: valor = **gethex_usb();**//retém um número hexadecimal na variável valor

- **usb_cdc_kbhit( )** – Avisa com TRUE (1) se acabou de chegar um novo caractere no buffer de recepção USB do PIC.

  Ex.: **if (usb_cdc_kbhit(1)) {dado = usb_cdc_getc();}**

Veja o programa abaixo compilado no CCS, instalado no Linux através do Wine, que pisca um led na função principal (pino B6) e comanda o estado se outro led (pino B7) pelo teclado de um PC via USB através do protocolo CDC*:*

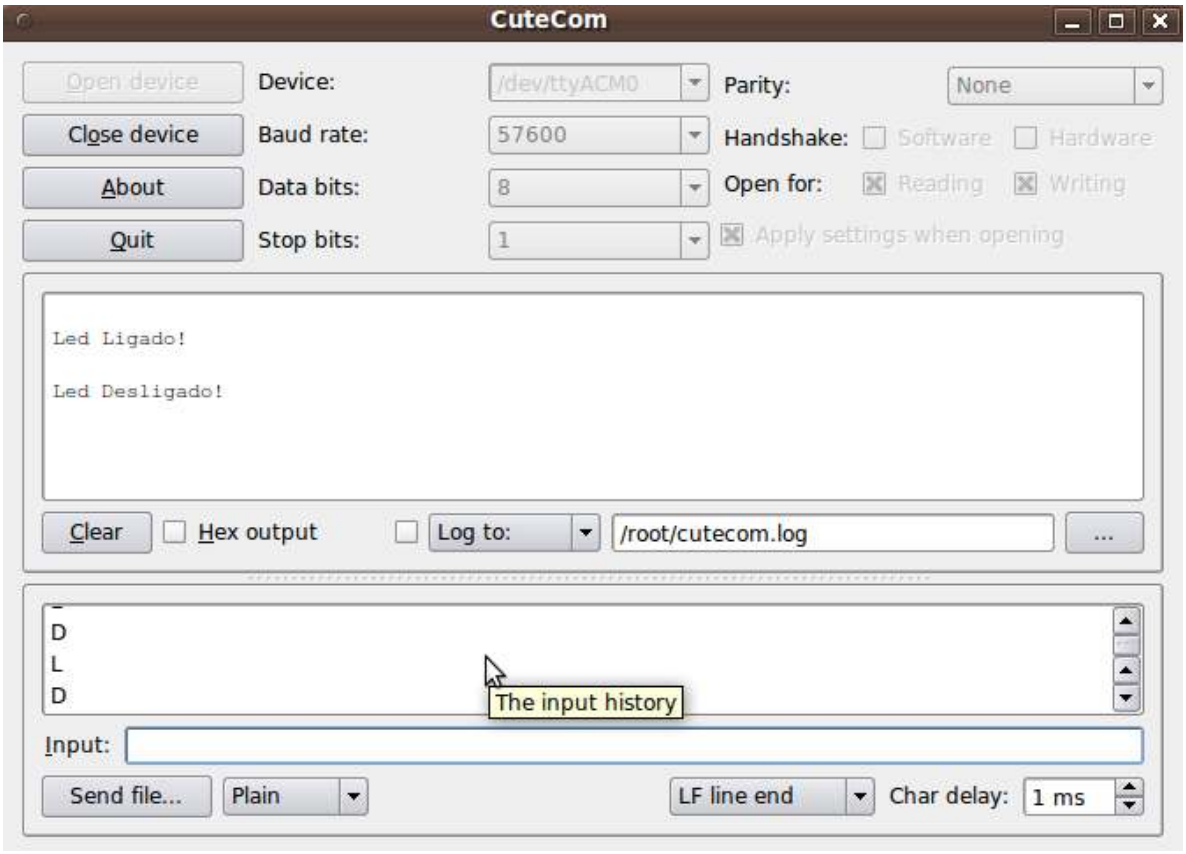
```
#include <SanUSB.h>
#include <usb_san_cdc.h>// Biblioteca para comunicação serial virtual via USB

BYTE comando;

main() {
clock_int_4MHz();//Função necessária para habilitar o dual clock (48MHz para USB e 4MHz para CPU)
usb_cdc_init(); // Inicializa o protocolo CDC
usb_init(); // Inicializa o protocolo USB
```

```
usb_task(); // Une o periférico com USB do PC

while (TRUE)
{
if (usb_cdc_kbhit(1)) //avisa se chegou dados do PC
{ //verifica se tem um novo byte no buffer de recepção, depois o kbhit é zerado para próximo byte
comando=usb_cdc_getc(); //se chegou, retém o caractere e compara com 'L' ou 'D' em ASCII

if (comando=='L') {output_high(pin_b7); printf(usb_cdc_putc, "\r\nLed Ligado!\r\n");}
if (comando=='D') {output_low(pin_b7); printf(usb_cdc_putc, "\r\nLed Desigado!\r\n");}
}
output_high(pin_B6); // Pisca Led na função principal
delay_ms(500);
output_low(pin_B6);
delay_ms(500);
} }
```

Este firmware realiza a comunicação serial virtual com o protocolo CDC inserido no firmware do microcontrolador através da biblioteca usb_san_cdc.h. Este protocolo é padrão no sistema operacional Linux.

Após gravar o firmware via USB com o executável linux *sanusb*, instale o software de comunicação serial CuteCom.deb disponível nesta pasta.

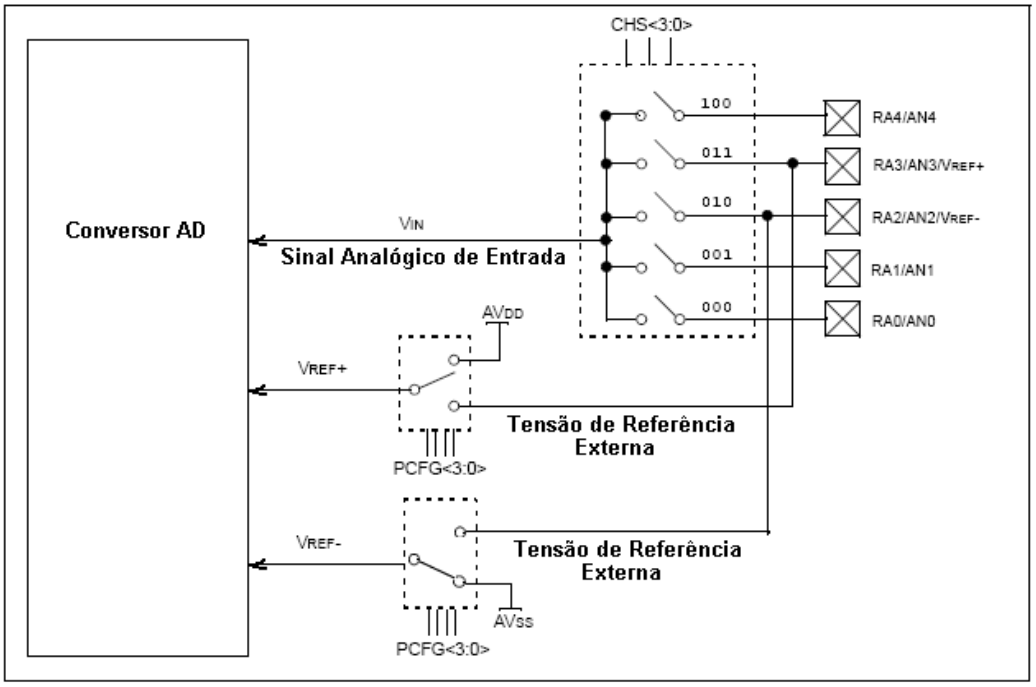
Verifique a porta serial virtual criada digitando *dmesg* no terminal. Abra o Cutecom, digitando cutecom no terminal e direcione a porta virtual criada em Device do Cutecom, geralmente a porta é ttyACM0 ou ttyACM1. Mais informações podem ser obtidas no video: http://www.youtube.com/watch?v=cRW99T_qa7o .



3. CONVERSOR ANALÓGICO-DIGITAL (AD)

O objetivo do conversor analógico-digital (AD) é converter um sinal analógico, geralmente de 0 a 5V, em equivalentes digitais. Como pode ser visto, 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.

DIAGRAMA DE BLOCOS DO CONVERSOR AD

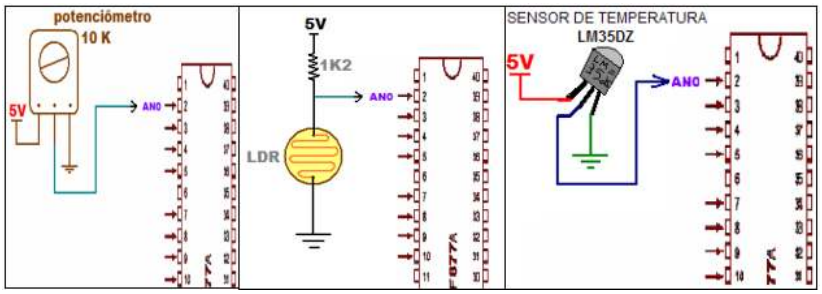


Em C, o conversor AD pode ser ajustado para resolução de 8 bits (#device adc=8 armazenando o resultado somente no registro ADRESH) ou 10 bits (#device adc=10).

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, etc..



Para utilizar este periférico interno, basta:

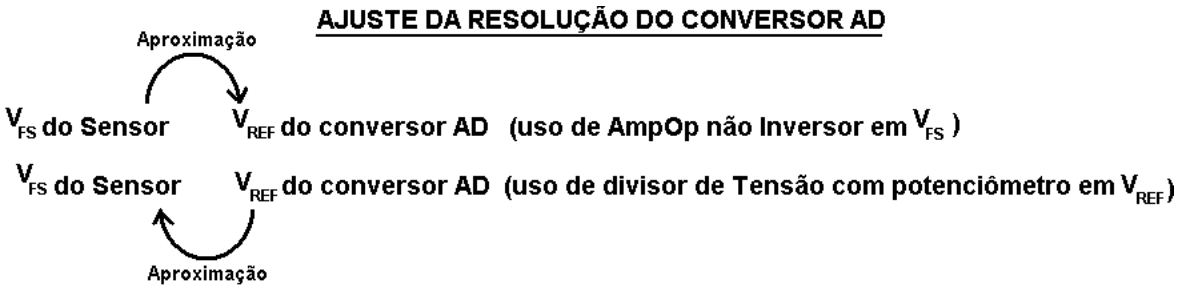
```
setup_adc_ports (AN0_TO_AN2);              //(Seleção dos pinos analógicos 18F2550)
setup_adc(ADC_CLOCK_INTERNAL );    //(selecionar o clock interno)
```

Veja a nomenclatura dos canais analógicos de cada modelo, dentro da biblioteca do CCS 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.

```
set_adc_channel(0);          //Seleciona qual canal vai converter
delay_ms (10);               // aguarda 10ms
valor = read_adc();          // efetua a leitura da conversão A/D e guarda na variável valor
```

3.1. 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*:

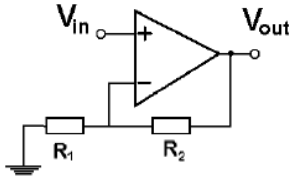


Para este tópico é utilizado como exemplo de ajuste da resolução do conversor AD, o sensor de temperatura LM35 que fornece uma saída de tensão linear e proporcional com uma resolução de 10mV a cada °C. A precisão nominal é de 0,5°C a 25°C.

3.2. AJUSTE DA TENSÃO DE FUNDO DE ESCALA COM AMPOP

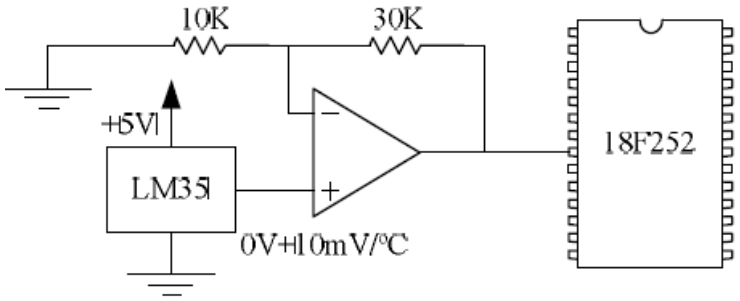
Para conversores AD de 8 bits e V_{REF} de 5V, a resolução máxima é de 19,6mV ($R= V_{REF} / (2^n-1)$). Dessa forma, como a Resolução do sensor é 10mV/°C (R_S), é necessário aplicar um ajuste de resolução com um ganho na tensão de fundo de escala do sensor para que cada grau possa ser percebido pelo 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 $\mathbf{V_{FS} = R_S}$(10mV/°C) * $\mathbf{T_{MAX}}$ e o Ganho (G) de aproximação da tensão de fundo de escala (V_{FS}) à tensão de referencia (V_{REF}) é dado por $\mathbf{G = V_{REF} / V_{FS}}$, ou seja, para uma Temperatura Máxima desejada de 100°C, o ganho deve ser aproximadamente 5.

Amplificador operacional não inversor e o respectivo ganho



$$G = \frac{V_{out}}{V_{in}} = 1 + \frac{R_2}{R_1}$$

A aproximação da tensão de fundo de escala (V_{FS}) à tensão de referencia (V_{REF}) para diminuir a relevância de ruídos em determinadas faixas de temperatura. Por exemplo, 100°C apresenta uma tensão de fundo de escala de apenas 1Volt (100°C *10mV/°C) para uma V_{REF} de 5V do conversor AD. O circuito abaixo mostra um exemplo de uso de amplificador para melhorar a resolução do conversor.



Circuito2: Temperatura máxima desejada de 125°C e Ganho de 4 (1+3/1)

Neste exemplo, a tensão de fundo de escala (V_{FS}) foi aumentada, em uma faixa de 0 a 100°C para 4 Volts (Ganho de 4). Dessa forma, na tensão de referência de 5V, o conversor mede 125°C. Como neste caso o conversor AD é de 10 bits, então, ele percebe uma variação de temperatura a cada 0,12°C (125°C /1023).

3.3. 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 de 2,55, 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 a cada °C.

3.4. 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 do circuito 3 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 em uma temperatura é:

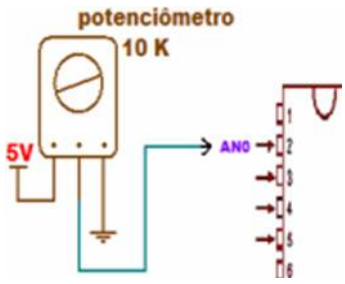
ANALÓGICO		**DIGITAL**
5V	->	1023
T(°C)* 10mV/°C	->	(int32)read_adc()

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.

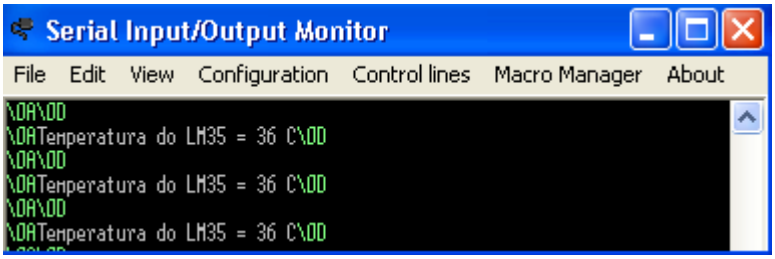
OBTENÇÃO DE UM VOLTÍMETRO ATRAVÉS DO CONVERSOR AD COM A VARIAÇÃO DE UM POTENCIÔMETRO

```
#include <SanUSB.h> //Leitura de tensão em mV com variação de um potenciômetro
#include <usb_san_cdc.h>// Biblioteca para comunicação serial virtual

int32 tensao;

main() {
clock_int_4MHz();

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 C\r\n",tensao); // Imprime pela serial virtual

output_high(pin_b7);
delay_ms(500);
output_low(pin_b7);
delay_ms(500);            }}
```



LEITURA DE TEMPERATURA COM O LM35 ATRAVÉS DO CONVERSOR AD

```
#include <SanUSB.h>

#include <usb_san_cdc.h>// Biblioteca para comunicação serial virtual

int16 temperatura;

main() {
clock_int_4MHz();
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){
set_adc_channel(0);
delay_ms(10);
temperatura=430*read_adc()/1023; //Vref = 4,3V devido à queda no diodo, então (430*temp)
printf (usb_cdc_putc,"\r\nTemperatura do LM35 = %lu C\r\n",temperatura);

output_high(pin_b7);    // Pisca Led em operação normal
delay_ms(500);
output_low(pin_b7);
delay_ms(500);              }}
```



4. INSTRUÇÕES LÓGICAS E ARITMÉTICAS

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

4.1. INSTRUÇÕES LÓGICAS PARA TESTES CONDICIONAIS DE NÚMEROS

Nesse caso, os operadores são utilizados para realizar operações de testes condionais geralmente entre números inteiros.

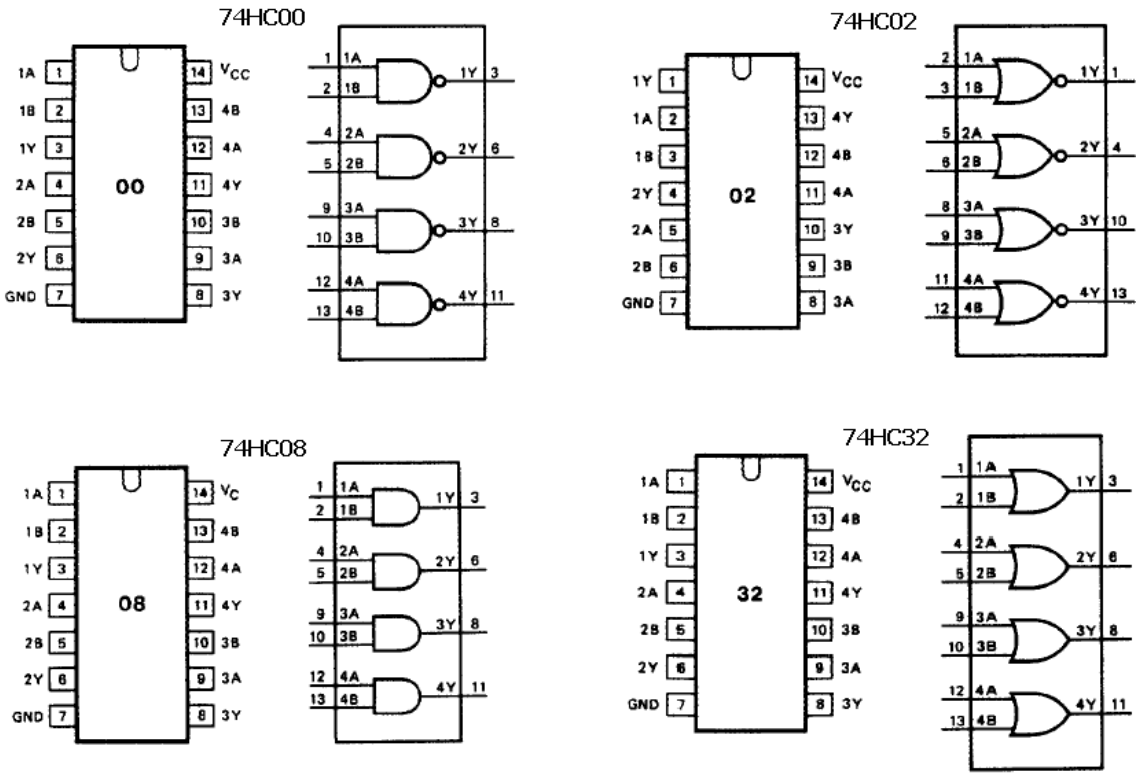
OPERADOR	COMANDO
&&	Operação E (AND)
ll	Operação OU (OR)
!	Operação NÃO (NO)

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.
```

4.2. INSTRUÇÕES LÓGICAS BOOLANAS BIT A BIT

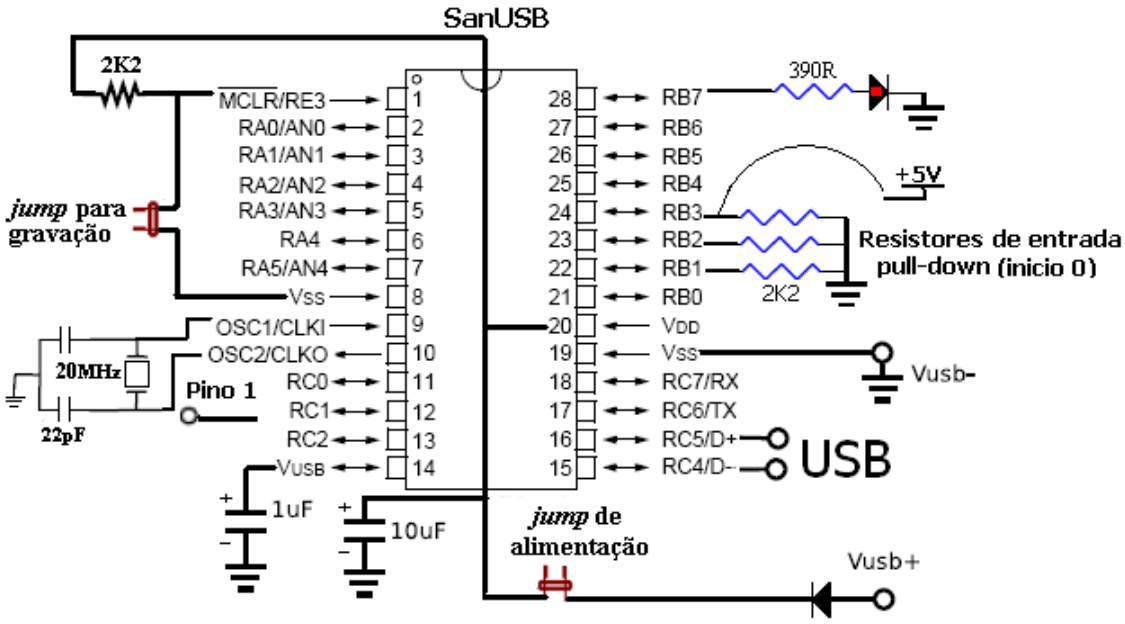
Considere as portas lógicas abaixo:



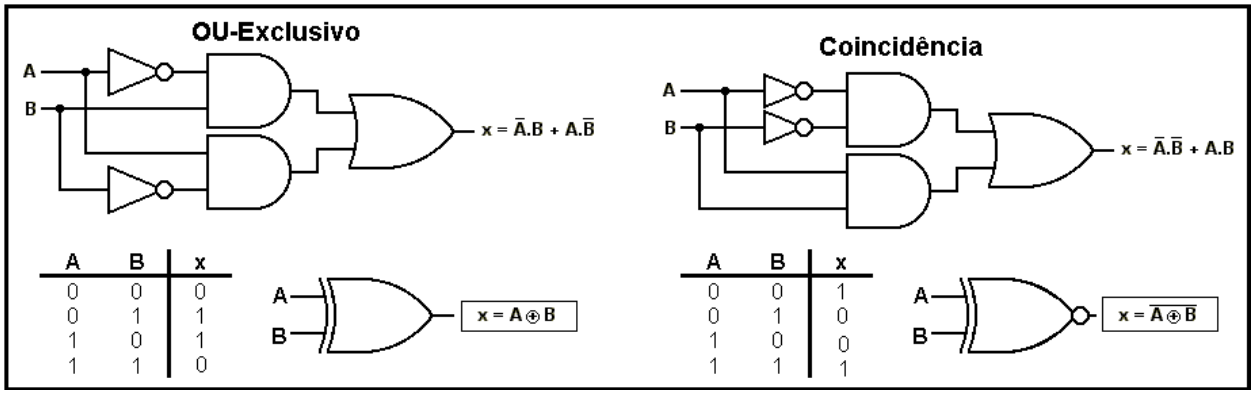
É importante salientar que emulação é a reprodução via software das funções de um determinado sistema real. Através do circuito SanUSB, é possível emular as portas lógicas físicas e as diversas combinações das mesmas com apenas uma função booleana no programa.

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 I B
Operação NÃO (NO)	S= $\overline{A}$	S= !A
OU exclusivo (XOR)	S= A $\oplus$ B	S= A ^ B

O circuito abaixo 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.



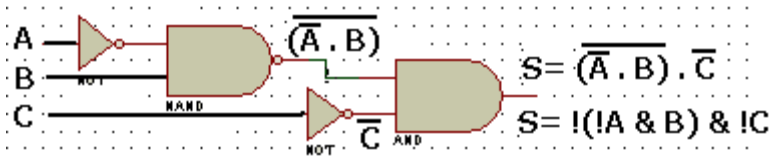
É importante salientar que através das operações básicas E, OU, NÃO e OU-Exclusivo é possível construir outras operações como NAND, NOR e Coincidência, que é o inverso do OU-Exclusivo. Isto é realizado transformando a expressão booleana literal em uma expressão booleana em C e apresentando o resultado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



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

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

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
delay_ms(500);
}
}
```

Exemplo 2: Elabore um programa e a tabela verdade para emular uma o circuito lógico booleano abaixo.



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

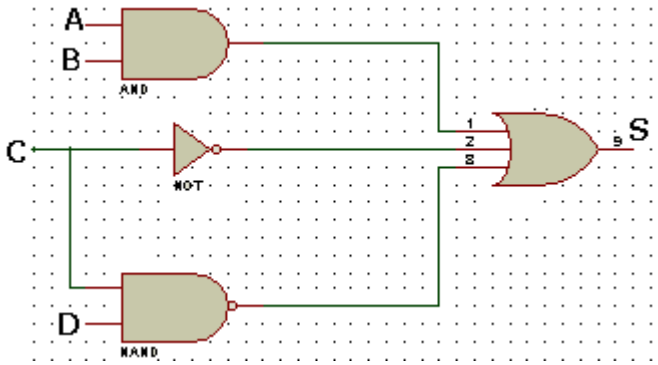
```
#include <SanUSB.h> // Emulação de circuitos lógicos booleanos
short int A, B, C, saida, ledpisca;

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 dos pinos 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
delay_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 mostarada abaixo:

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

Exemplo 3: Elabore um programa e a tabela verdade para emular uma o circuito lógico booleano abaixo.



S= (A & B) | !C | !(C & D)

```
#include <SanUSB.h> // Emulação de circuitos lógicos booleanos
short int A, B, C, D, saida, ledpisca;

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
delay_ms(500);
}
}
```

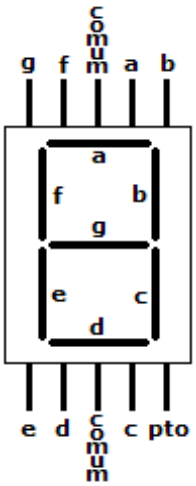
A tabela verdade deste circuito lógico é mostrada abaixo:

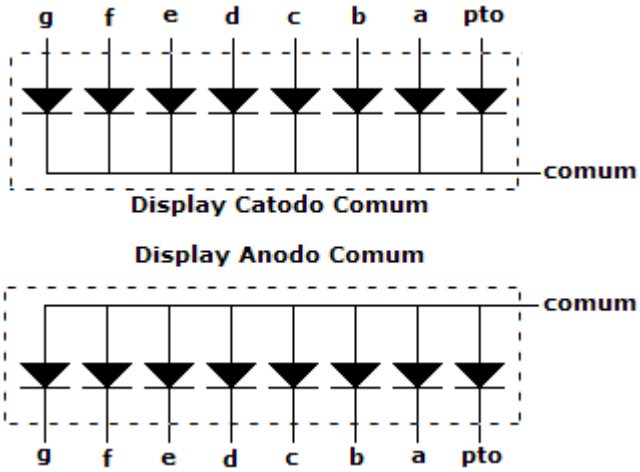
ENTRADAS				SAIDA
A	**B**	**C**	**D**	**S**
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

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

4.3. 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 figura abaixo, 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, pois se não 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.

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.

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

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 na tabela abaixo:

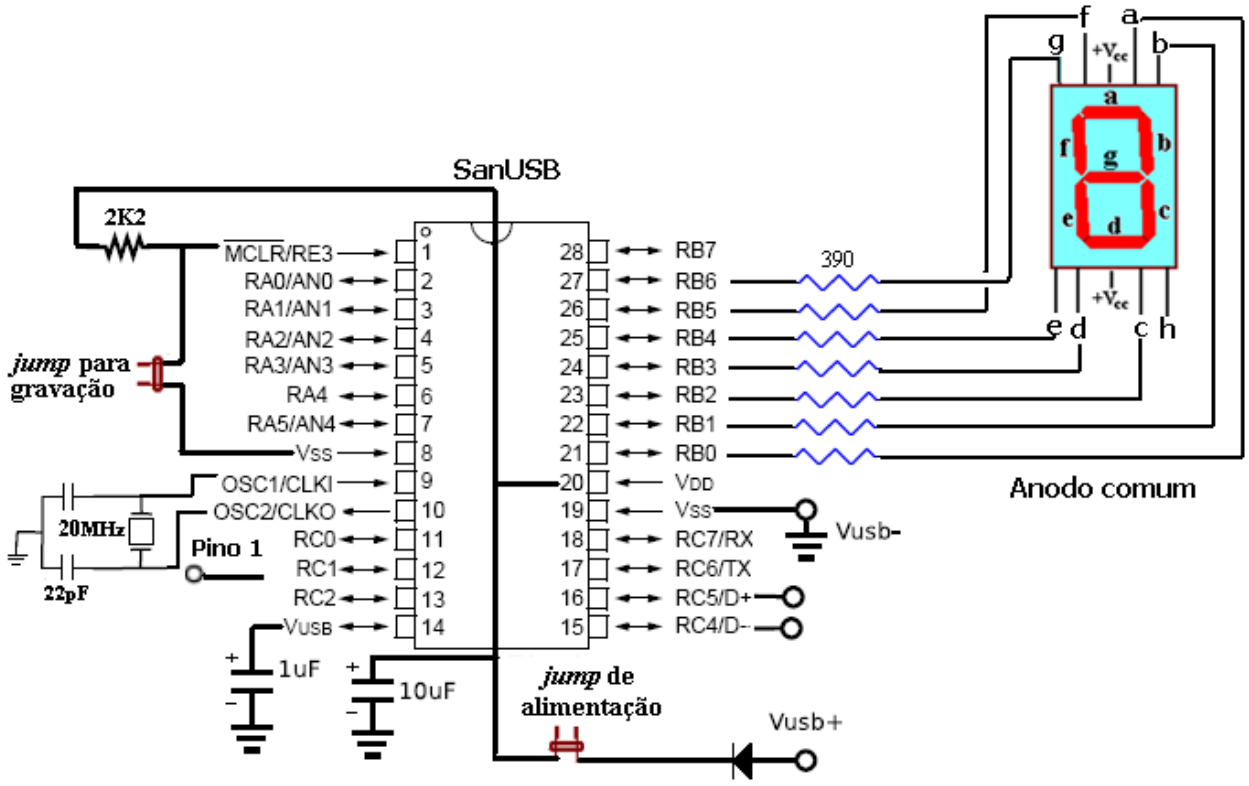
TABELA (Anodo comum – zero no pino acende segmento)								
NÚMERO DYSPLAY	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	**Porta B Hexadecimal**
	g	**f**	**e**	**d**	**c**	**b**	**a**	
0	1	0	0	0	0	0	0	0x40
1	1	1	1	1	0	0	1	0x79
2	0	1	0	0	1	0	0	0x24
3	0	1	1	0	0	0	0	0x30
4	0	0	1	1	0	0	1	0x19
5	0	0	1	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

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 <SanUSB.h>
#byte port_b = 0xf81//Atribuição do nome portb para o registro da porta B localizado na posição 0xf81 da memória de dados
```

```
int setseg[10] = {0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10}; //Vetor com 10 elementos que desenham de 0 a 9
int i;  //índice i (ponteiro)

void main ()
{
 clock_int_4MHz();
 set_tris_b(0b00000000);// Define os pinos da porta B como saída

 while(1)
 {
  for (i=0;i<10;i++)
  {
   port_b = setseg[i];
   delay_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 abaixo mostra a ligação do display de 7 segmentos.



Os segmentos ascendem com zero no pino do microcontrolador (anodo comum). Note que como são somente quatro letras só é necessário duas entradas (Xe Y) para a decodificação.

	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

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 abaixo para a construção da função de emulação do decodificador fixo.

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

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

```
#include <SanUSB.h>  // Emulação de decodificador para display de 7 segmentos - palavra StoP
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;          output_bit(pin_b0,a);      //Anodo comum
        b=!X | !Y;          output_bit(pin_b1,b);
        c=Y;                output_bit(pin_b2,c);
        d=X & Y;            output_bit(pin_b3,d);
        e=!X & !Y;          output_bit(pin_b4,e);
        f=X & !Y;           output_bit(pin_b5,f);
        g=0 ;               output_bit(pin_b6,g);
}
```

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

  decodificador(0,1); // Saída letra t
  delay_ms(1000);

  decodificador(1,0); // Saída letra o
  delay_ms(1000);

  decodificador(1,1); // Saída letra P
  delay_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 ascendem com zero no pino do microcontrolador.

	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

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 abaixo para a construção da função de emulação do decodificador fixo.

a	**/Y**	**Y**	**b**	**/Y**	**Y**	**c**	**/Y**	**Y**	**d**	**/Y**	**Y**
/X	1	0	**/X**	0	1	**/X**	0	0	**/X**	0	0
X	1	0	**X**	1	0	**X**	0	1	**X**	0	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

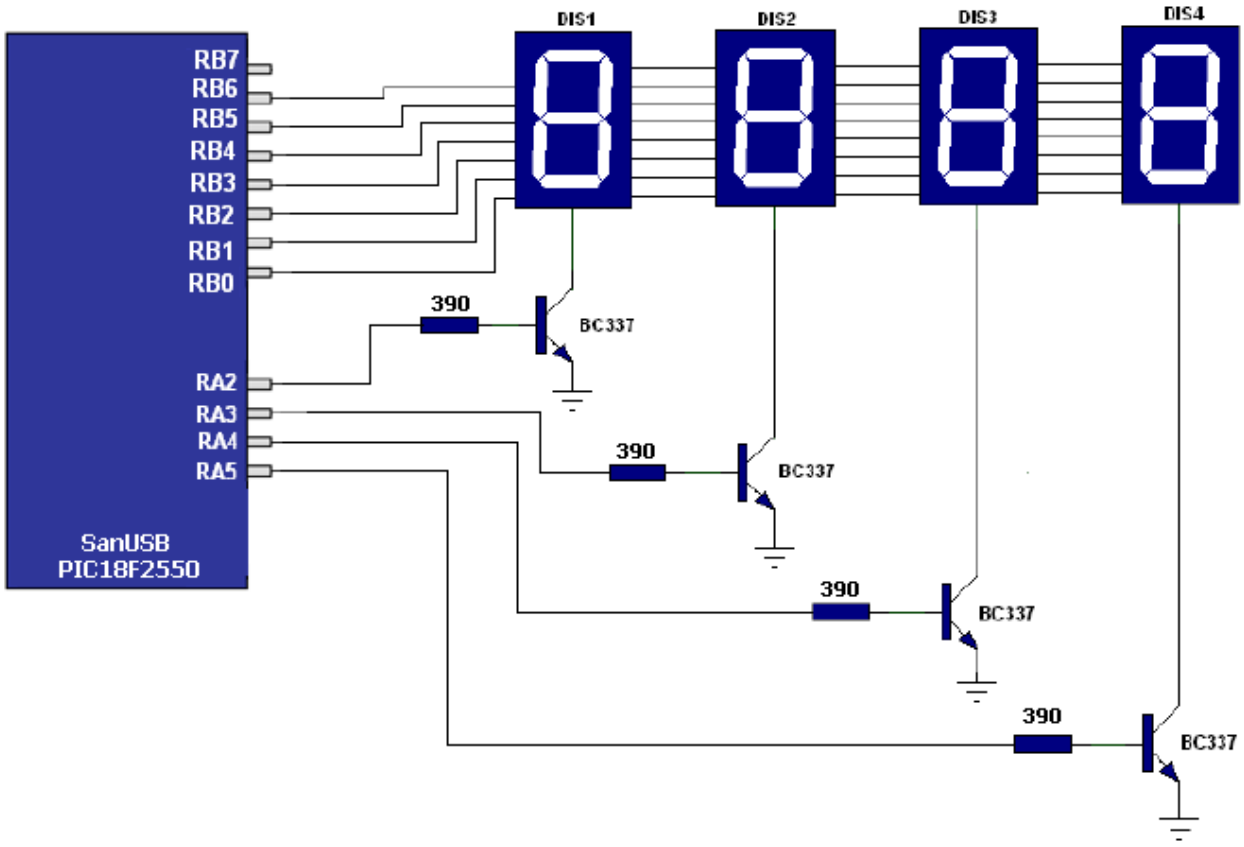
O programa abaixo mostra as funções de emulação do decodificador fixo para a palavra USb2 obtidas dos diagramas de Karnaugh.

```
#include <SanUSB.h>  // Emulação de decodificador para display de 7 segmentos - palavra Usb2
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;              output_bit(pin_b0,a);   //Anodo comum
b=X^Y;             output_bit(pin_b1,b);
c=X&Y;             output_bit(pin_b2,c);
d=0;               output_bit(pin_b3,d);
e=!X&Y;            output_bit(pin_b4,e);
f=X&Y;             output_bit(pin_b5,f);
g=!X&!Y;           output_bit(pin_b6,g);
}
main(){
clock_int_4MHz();
while (1)
{
  decodificador(0,0); // Insere as entradas X=0 e Y=0 no decodiicador fixo – Saída letra S
  delay_ms(1000);

  decodificador(0,1); // Saída letra t
  delay_ms(1000);

  decodificador(1,0); // Saída letra o
  delay_ms(1000);

  decodificador(1,1); // Saída letra P
  delay_ms(1000);
}}
```

4.4. 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 freqüência 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 <SanUSB.h>
#byte port_b = 0xf81//Atribuição do nome portb para o registro da porta B localizado na posição 0xf81 da memória de dados

int setseg[10] = {0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10}; //Vetor com 10 elementos que desenham de 0 a 9
int i, z, dezena, unidade;  //índice dezena,unidade (ponteiro)

void main ()
{
 clock_int_4MHz();
 set_tris_b(0b00000000);// Define os pinos da porta B como saída

 while(1)
 {
  for (i=0;i<99;i++)
  {
  for(z=0;z<20;z++){

  dezena=i/10;  //dezena recebe o número inteiro da divisão por 10
  unidade=i%10; //unidade recebe o resto  da divisão por 10

    output_high(pin_a0);  //pin_a0 aciona transistor do comum das dezenas
    output_low(pin_a1);    //pin_a3 aciona transistor do comum das unidades
    port_b = setseg[dezena]; //A porta B recebe o desenho do número das dezenas apontado pela variável dezena
    delay_ms(10);

    output_high(pin_a1);  //selecionei a unidade
    output_low(pin_a0);
```

```
    port_b = setseg[unidade]; //A porta B recebe o desenho do número das unidades apontado pela variável unidade
    delay_ms(10);
  }}
 }
}
```

5. 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.

5.1. 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_interrupts(int_ext);* como mostra o exemplo com aplicação de interrupção externa e também interrução do temporizador 1.

```
#include <SanUSB.h>

BYTE comando;

short int led;
int x;

#int_timer1
```

```
void trata_t1 ()
{
  led = !led; // inverte o led - pisca a cada 0,5 seg.
  output_bit (pin_b7,led);
  set_timer1(3036 + get_timer1());
}


#int_ext
void bot_ext()
{
  for(x=0;x<5;x++) // pisca 5 vezes após o pino ser aterrado (botão pressionado)
  {
  output_high(pin_B5); // Pisca Led em B5
  delay_ms(1000);
  output_low(pin_B5);
  delay_ms(1000);
  }
}

main() {
clock_int_4MHz();
enable_interrupts (global); // Possibilita todas interrupcoes
enable_interrupts (int_ext); // Habilita interrupcao externa 0 no pino B0
enable_interrupts (int_timer1); // Habilita interrupcao do timer 1
setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8);// configura o timer 1 em 8 x 62500 = 0,5s
set_timer1(3036);                // Conta 62.500us x 8 para estourar= 0,5s

while (1){};  //Loop infinito (parado aqui)
                                        }
```

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 nível lógico do pino quando o mesmo for liberado evitando outra interrupção, pois o processador entende *tristate* e níveis intermediários de tensão como nível lógico baixo.

5.2 INTERRUPÇÃO DOS TEMPORIZADORES

O microcontrolador PIC 18F2550 tem quatro temporizadores, que são os timers 0, 1, 2 e 3. O *timer* 0 tem 16 bits, ou seja, pode contar até 65535µs (2^{16}) e um *prescaler* (divisor de freqüência ou multiplicador de tempo) de até 256 (RTCC_DIV_256). Os *timers* 1 e 3 são

idênticos com 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).

Os timers incrementam até estourar, quando estouram, processamento é desviado para **#int_timer**, para que a interrupção possa ser tratada por uma função específica, que no caso do exemplo é *void trata_t0 () e void trata_t1 ()*.

O programa 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 <SanUSB.h>

short int led0, led1;
int vart1=2, vart3=4; // multiplicador de tempo

#int_timer0
void trata_t0 () //Função de taratamento, o Timer0 é configurado com o nome RTCC
{
   led0 = !led0; // inverte o led a cada 4 seg pois tem prescaler igual a 64 (RTCC_DIV_64)
  output_bit (pin_b7,led0);
  set_timer0(3036 + get_timer0()); // get_timer() carrega o timer compensando o tempo gasto no tratamento da interrupção
}

#int_timer1 //O timer 1 e o timer 3 são idênticos, só basta modificar 1 por 3 na configuração
void trata_t1 ()
{
  --vart1;
  if(vart1==0)
   {
  led1 = !led1; // inverte o led - pisca a cada 1 seg (vart1=2 x 0,5 seg)
  output_bit (pin_b6,led1);
  vart1=2;  // necessita de multiplicador de tempo, pois o prescaler máximo é 8 (T1_DIV_BY_8)
  set_timer1(3036 + get_timer1()); // get_timer() carrega o timer compensando o tempo gasto no tratamento da interrupção
   }
}

main(){
clock_int_4MHz();

enable_interrupts (global); // Possibilita todas interrupcoes
enable_interrupts (int_timer0); // Habilita interrupcao do timer 0
enable_interrupts (int_timer1); // Habilita interrupcao do timer 1

setup_timer_0 ( RTCC_INTERNAL | RTCC_DIV_64);// configura o prescaler do timer 0 em 64, tem prescaler até 256
set_timer0(3036);                              // Conta 62.500us x 64 para estourar= 4 seg

setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8); // configura o prescaler do timer 1 em 8 x 62500us = 0,5 seg
set_timer1(3036);                             // Conta 62.500us x 8 para estourar= 0,5 seg

while (1){                                    //Função principal pisca led em a5
```

```
            output_high(pin_b5);
            delay_ms(500);
            output_low(pin_b5);
            delay_ms(500);
           }
    }
```

5.3 MULTIPLEXAÇÃO 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 <SanUSB.h>

#byte port_b = 0xf81//Atribuição do nome portb para o registro da porta B localizado na posição 0xf81
int setseg[10] = {0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10}; //Vetor com 10 elementos

int flag=0;
int i=0, z, dezena, unidade;  //índice dezena,unidade (ponteiro)

//*****************************************
#int_timer0
void trata_t0 () //O Timer0 é configurado com o nome RTCC
{
if(i<=99) {++i;}
if(i>99) {i=0;}
set_timer0(3036 + get_timer0()); // get_timer() carrega o timer compensando o tempo gasto na interrupção
}
//*****************************************

#int_timer1 //O timer 1 e o timer 3 são idênticos, só basta modificar 1 por 3 na configuração
void trata_t1 ()
{
dezena=i/10;  //dezena recebe o número inteiro da divisão por 10
unidade=i%10; //unidade recebe o resto  da divisão por 10

switch(flag)
{
case 0: {
     output_high(pin_a0);  //pin_a0 aciona transistor do comum das dezenas
     output_low(pin_a1);    //pin_a3 aciona transistor do comum das unidades
     port_b = setseg[dezena]; //A porta B recebe o desenho do número das dezenas apontado pela variável dezena
               flag=1; break;}

case 1: {
     output_high(pin_a1);  //selecionei a unidade
     output_low(pin_a0);
     port_b = setseg[unidade]; //A porta B recebe o desenho do número das unidades apontado pela variável unidade
     flag=0; break;}

          }

set_timer1(55536 + get_timer1());
}
```

```
main(){
clock_int_4MHz();
set_tris_b(0b00000000);// Define os pinos da porta B como saída

enable_interrupts (global); // Possibilita todas interrupcoes
enable_interrupts (int_timer0); // Habilita interrupcao do timer 0
enable_interrupts (int_timer1); // Habilita interrupcao do timer 1

setup_timer_0 ( RTCC_INTERNAL | RTCC_DIV_8);// configura o prescaler do timer 0 em 64, prescaler até 256
set_timer0(3036);                                    // Conta 62.500us x 8 para estourar= 0,5 seg

setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_1);
set_timer1(55536);                                   // Conta 10000 us (10ms) para estourar

while (1){                                           //Função principal
            output_high(pin_a5);
            delay_ms(300);
            output_low(pin_a5);
            delay_ms(300);
           }
    }
```

6. 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.

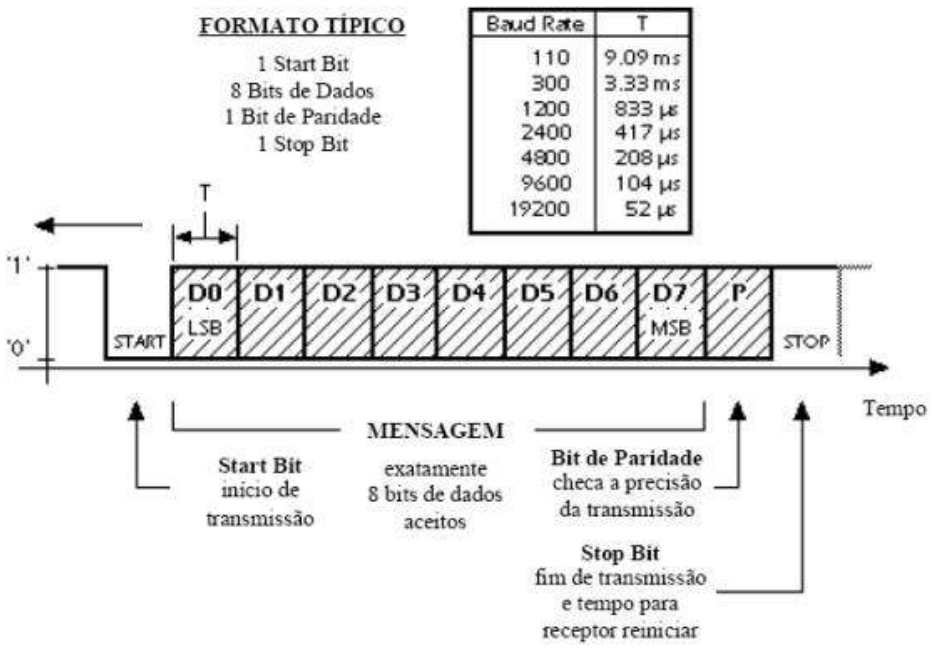
A Comunicação serial é feita pela transmissão de bits em seqüência. É um modo de comunicação muito recomendado para transmissão de dados a longa distância. Nesse caso, a comunicação serial apresenta um menor custo pelo número reduzido de fios e conseqüentemente menor velocidade em relação à comunicação 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 (nível lógico baixo)* e um bit que indica o final da transmissão chamado de *stop bit (nível 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 nível lógico ‘1’. Um pacote de dados sempre começa com um nível 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”).

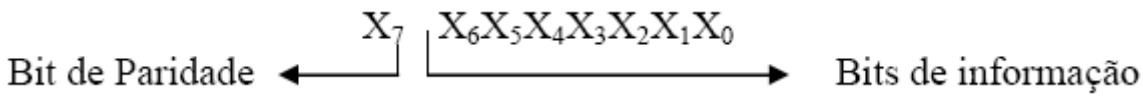
O bit de paridade é usado como nono bit com o propósito de detecção de erro. Nessa convenção, quando o número total de dígitos ‘1’ , o valor do bit de paridade é 1 e quando for ímpar é 0.



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 (nível 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.

6.1. 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.



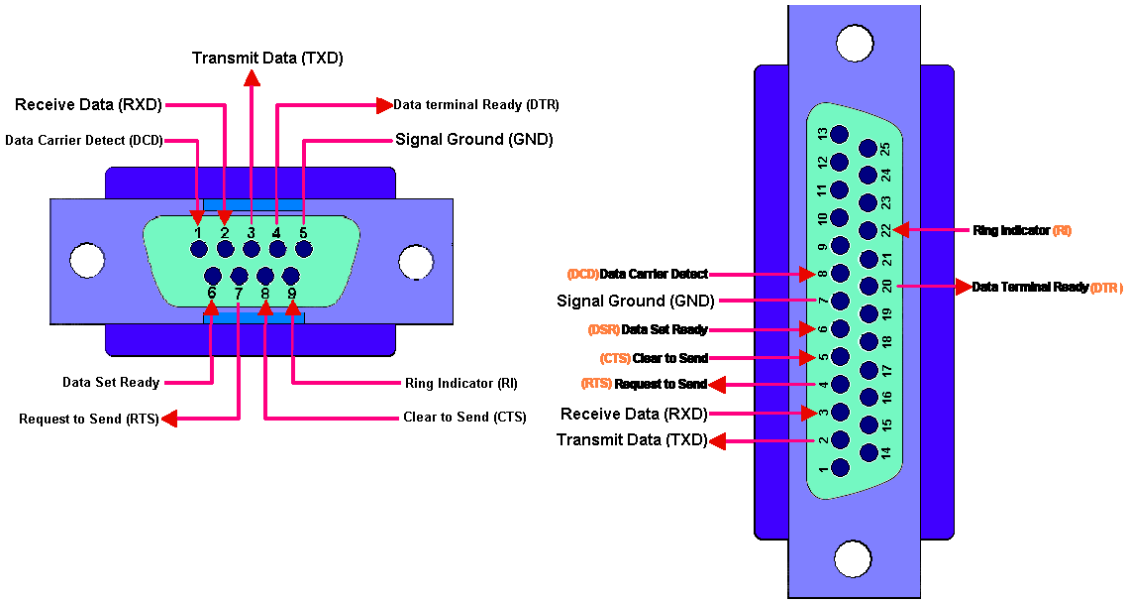
Se o número de “1s” for par, o bit de paridade X_7 é zero e, se for ímpar, X_7 é um.

A Tabela de Caracteres ASCII é mostrada abaixo:

HEX	CHR	HEX	CHR	HEX	CHR	HEX	CHR
00	NUL	20	SPC	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL

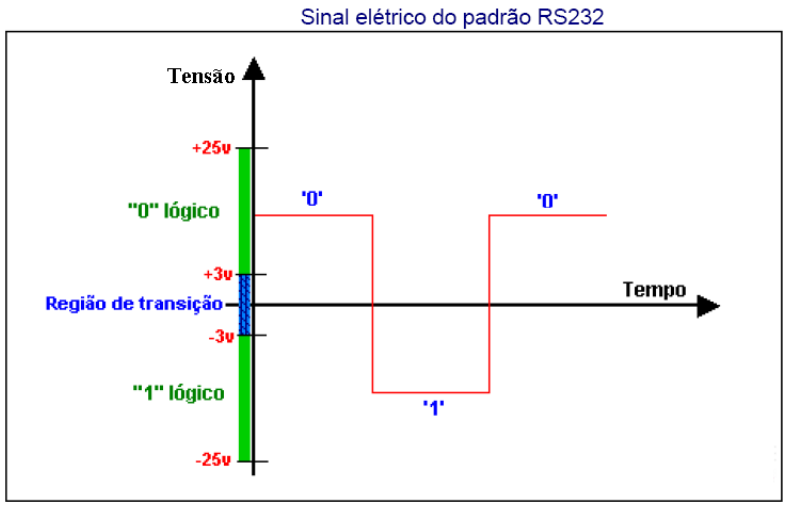
6.2. INTERFACE USART DO MICROCONTROLADOR

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 nível lógico baixo e encerrado por um stop-bit de nível lógico alto. Os conectores utilizados são o DB9 e o DB25, como mostra a figura abaixo:

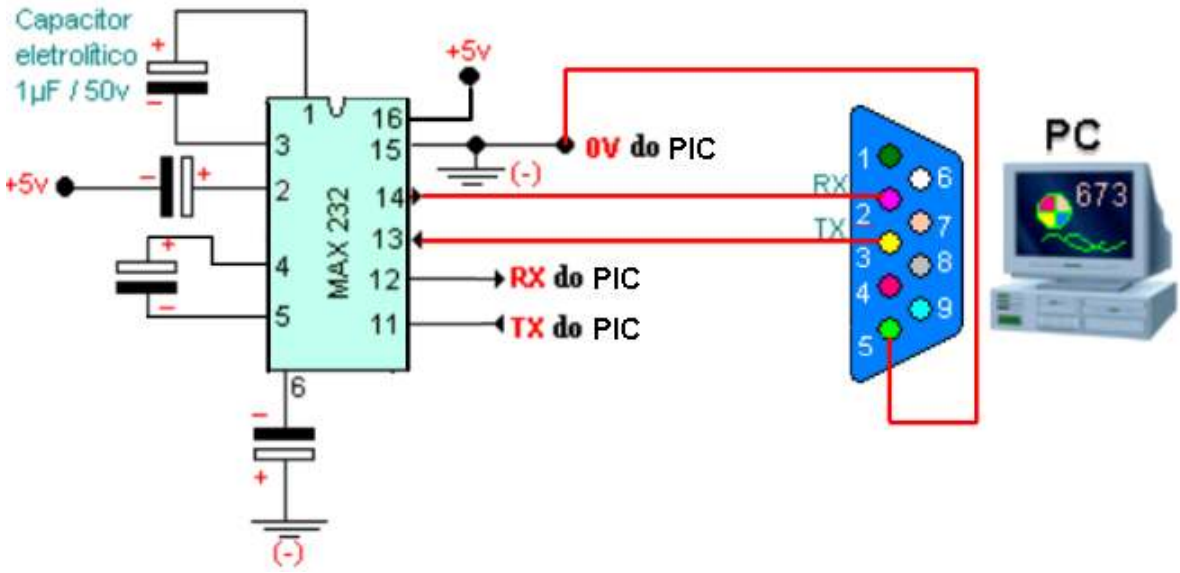


Em suma, os pinos utilizados na comunicação serial entre computadores e microcontroladores são o TXD, o RXD e o Terra (GND).

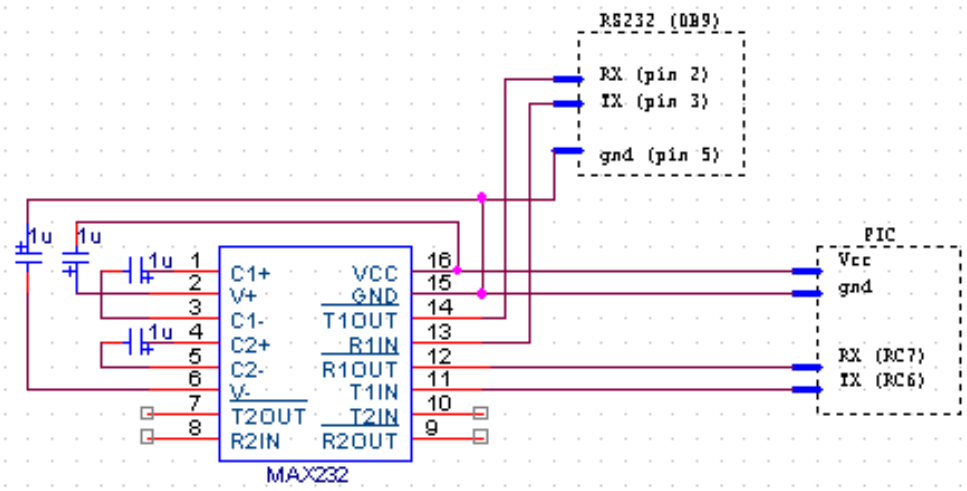
O nível lógico alto no padrão RS232 está entre **–3 e –25V** e o nível lógico baixo está entre **+3 e +25V**. Para a comunicação entre um PC e um PIC são utilizados chips que convertem os níveis de tensão TTL/RS232.



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 a seguir:

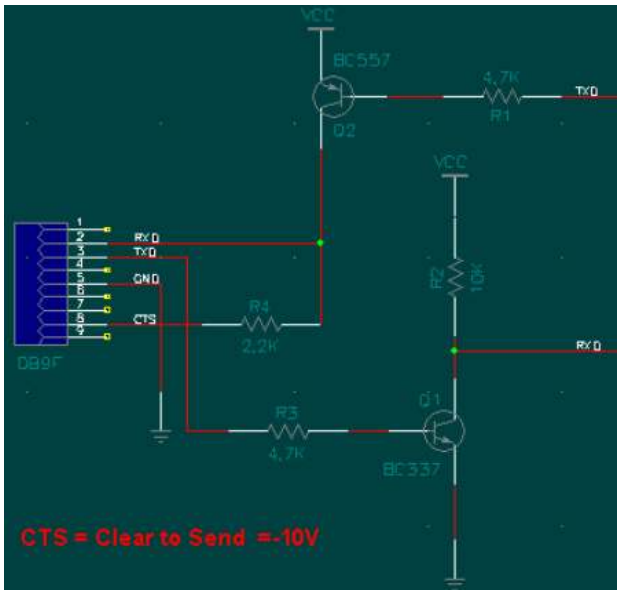


O circuito acima pode ser representado também como:



6.3. CIRCUITO EQUIVALENTE AO MAX232

Este circuito utiliza o pino 8 (Clear To Send igual a –10V) para fornecer tensão negativa para o pino 2 (Rx) quando o bit de recepção tiver nivel lógico alto. Ele é válido para pequenos cabos e velocidade de transmissão relativamente baixas, utiliza basicamente dois transistores, BC337 (NPN) e outro BC557 (PNP), 2 resistores de 4,7K, um de 2,2K e um de 10K para a conversão TTL/RS232, como mostra a figura abaixo. Note que, o nível lógico alto "1" em RS232 varia de –3 a –25V, e o nível lógico baixo "0" varia de +3 a +25V.



Quando o PC enviar o bit "1" (-10V) no DB9, o BC337 é cortado e o Rx do PIC recebe +5V, através do resistor de 10K, ou seja, "1".

Quando o PC enviar o bit "0" (+10V) no DB9, o BC337 é saturado, aterrando o pino Rx do PIC, ou seja, "0".

Quando o PIC enviar o bit "1" (+5V), o BC557 é cortado e a tensão no Rx do PC é igual a tensão de CTS (-10V) menos a queda no resistor de 2,2K, que corresponde em RS232 "1".

Quando o PIC enviar o bit "0" (0V), o BC557 é saturado e a tensão no Rx do PC é aproximadamente a 4,3V (VCC – 0,7V), ou seja, nível RS232 "0". Neste caso, o cabo entre o conector DB9 e o PC deve ser o menor possível para não provocar uma queda de tensão nos 4,3V (para menor que 3V), o que não seria compreendido como nível RS232 "0" pelo PC.

7. COMUNICAÇÃO SERIAL EIA/RS-485

No padrão EIA/RS-232, os sinais são representados por níveis de tensão referentes ao Gnd. Há um fio para transmissão, outro para recepção e o fio terra para referência dos níveis de tensão. Este tipo de interface é útil em comunicações ponto-a-ponto e baixas velocidades de transmissão. Visto a necessidade de um terra comum entre os dispositivos, há limitações do comprimento do cabo a apenas algumas dezenas de metros. Os principais problemas são a interferência e a resistência do cabo.

O padrão RS-485 utiliza um princípio diferente de multiponto, no qual o transmissor gera uma **tensão** diferencial entre **-1,5** e **-6** V entre o terminal A em relação ao terminal B para sinalizar um **bit 1** e gera uma **tensão** diferencial entre **+1,5** e **+6** V no terminal A em relação ao terminal B para sinalizar um **bit 0.** Com uma queda de tensão máxima de até 1,3 V, o receptor mede a diferença de tensão entre os terminais A e B e aceita tensões acima de 0,2 V como nível lógico 0 e abaixo de –0,2 V como bit 1. Portanto tensões diferenciais entre **-0,2** e **0,2** não são identificadas como sinal válido. As tensões medidas entre os terminais A e GND ou B e GND (**modo comum**) devem estar, respectivamente, entre -7 e +12 V.

Comparando o padrão RS-485 com o RS-232 encontramos um menor custo devido a possibilidade de uso de **fontes de alimentação** assimétricas, enquanto que o RS-232 exige o uso de fontes simétricas (terra) nos transmissores e receptores.

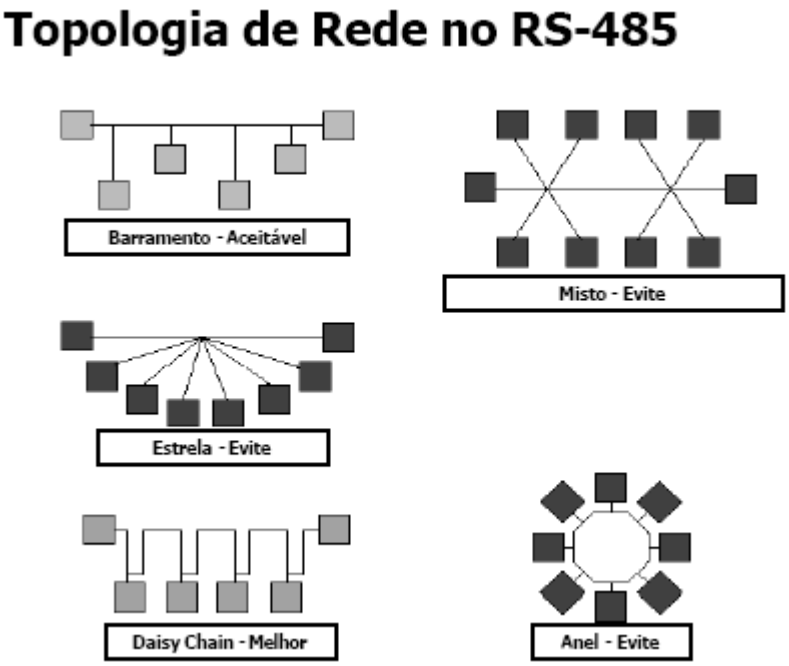
O RS-485 permite ainda a montagem de uma **rede de comunicação** sobre dois fios habilitando uma comunicação serial de dados confiável com:

- Distâncias de até 1200 metros (4000 pés);
- Velocidades de até 10Mbps;
- Até 32 nós na mesma linha de comunicação.

Este protocolo é muito utilizado ema rede mestre/escravo que adota o princípio de difusão da informação (*Broadcast*), onde todos recebem, em interrupção serial, um pacote (conjunto de bytes) de informações, mas só responde quem tem o endereço do pacote. Tem-se assim uma forma de evitar colisões de dados na rede, visto que apenas o mestre ou o escravo escolhido está transmitindo.

7.1. CABOS NO PADRÃO EIA/RS-485

É recomendado cabos par trançado ou triaxial com 1 ou dois pares de fios 24 AWG com impedância característica de 120 W. Os cabos utilizados em ambientes industriais adiciona ao par trançado a blindagem dupla com folha de alumínio (proteção capacitiva) e malha de cobre (proteção magnética) com conector dreno. Um erro comum nas montagens de rede RS-485 é a **troca** da ligação entre os terminais A e B de dispositivos distintos.



O uso de resistores de **terminação**, tipicamente de 120Ω , são necessários somente nos extermos do barramento para evitar os efeitos de reflexão de sinais, típicos de uma linha de transmissão.

Note que as derivações que conectam nós intermediários à topologia barramento precisam ser tão **curtas** quanto possível (se aproximando da Daisy Chain), pois uma longa derivação cria uma anomalia na impedância do cabo, que leva a reflexões indesejadas.

Se as linhas ou derivações intermediárias são menores que **100 metros** e a velocidade é menor ou igual a **9600 bps**, **o resistor de terminação da derivação torna-se desnecessário**, a não ser que o fabricante recomende.

7.2. DISTÂNCIA DE TRANSMISSÃO

Umas das vantagens da transmissão balanceada é sua robustez à ruídos e interferências. Se um ruído é introduzido na linha, ele é induzido nos dois fios de modo que a diferença entre A e B dessa interferência é tende a ser quase nula, com isso o alcance pode chegar a 4000 pés, **aproximadamente 1200 metros**.[4] Vale citar que o padrão RS-232 em sua taxa máxima de comunicação alcança em torno de 50 pés, **aproximadamente 15 metros**.[3]

COMUNICAÇÃO MULTIPONTO

Como o padrão RS-485 (**half-duplex**) foi desenvolvido para atender a necessidade de **comunicação multiponto** o seu formato permite conectar até 32 dispositivos, sendo 1 transmissor e 1 receptor por dispositivo.[4]

Funcionamento físico

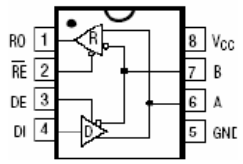


Figura 1. Transceptor RS 485

Ro: Saída para recepção[6]
RE: habilitação da recepção[6]
DE: habilitação da transmissão[6]
DI: Entrada para transmissão[6]
VCC,GND: Alimentação do circuito integrado[6]
A: Entrada não inversora[6] **(Transmissor)**
B: Entrada inversora[6] **(Receptor)**

Exemplo de um sistema RS-485

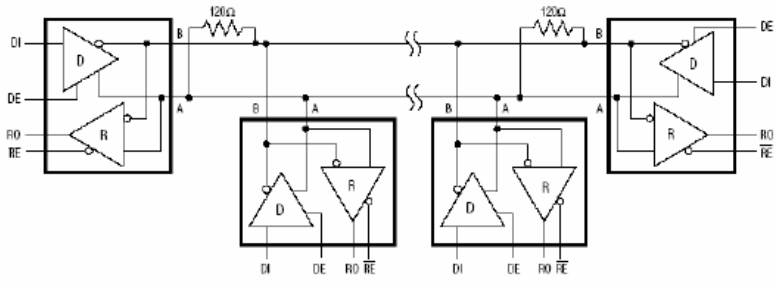
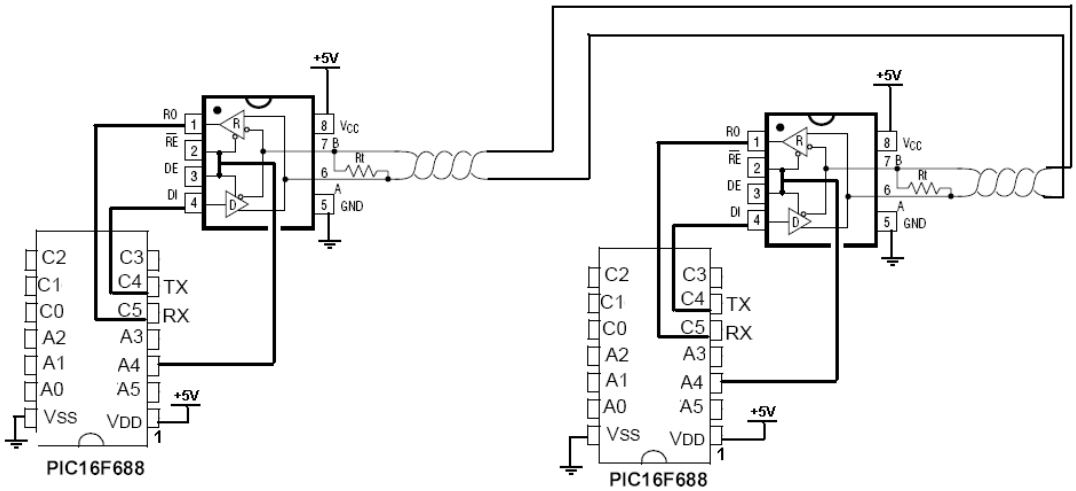


Figura 2. Sistema RS-485 com comunicação Half-duplex[6]

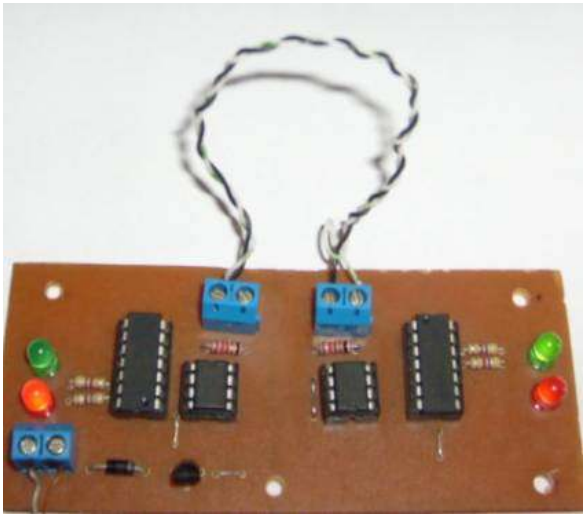
7.3. MODO DE OPERAÇÃO

Normalmente para o modo de transmissão e recepção simultâneo, uni-se os pinos /RE e DE constuindo um habilitador (*enable*) geral de forma que o transceptor esteja apenas recebendo ou transmitindo. Para que um dispositivo transmita um dado pelo barramento, é necessário *setar*

o pino DE, fazendo com que RE seja desabilitado, para então transmitir a informação necessária pelo pino DI, e ao fim da transmissão, desabilitar *ressetando* DE e reabilitando /RE, de forma que o transceptor volte ao modo de recepção. O CI deve sempre permanecer em modo de recepção. O circuito abaixo mostra dois microcontroladores PIC16F688 se comunicando no protocolo 485. Note que o pino A4 assume a função de *Enable* para a transmissão e recepção.



Protótipo montado com o modelo PIC16F688:



Veja o protótipo desse projeto em http://www.youtube.com/watch?v=oIxFXttjg-U. Um exemplo de programa é mostrado abaixo, onde o microcontrolador contrário comanda o acionamento do led de recepção em B6 através da interrupção serial. Para testar o firmware sem os transceptores RS-485, é possível ligar diretamente o Tx com o Rx de comunicação serial dos microcontroladores contrários para observar a comunicação ponto a ponto em TTL.

```
 #include <16F688.h>  // programa que pisca led de transmissão e outro de recepção  por
#include <SanUSB.h>

#INT_RDA
void recepcao_serial()// Interrupicao serial
{
char escravo,estado;
escravo=getc(); //comando é o Byte recebido pela serial
if (escravo=='A')
        {
        estado=getc();
        switch (estado)
                {
                case '0': {output_high(pin_B6);}
                break;
                case '1': {output_low(pin_B6);}
                break;
                }
}       }
main()
{
Clock_int_4MHz();
enable_interrupts(GLOBAL); // Possibilita todas interrupcoes
enable_interrupts(INT_RDA); // Habilita interrupcao da serial
output_low(PIN_b6);
//#############################################################################
output_low(PIN_A4); // Max485 inicia em modo de recepção
//#############################################################################
while(1)
{
//#############################################################################
output_high(PIN_A4); //Habilita Max485 em modo de Transmissão
printf ("A0\r\n"); //Transmite dado
output_low(PIN_A4); //Habilita Max485 em Modo de Recepção
//#############################################################################
output_high(pin_B7);
delay_ms (1000);
//#############################################################################
output_high(PIN_A4); //Habilita Max485 em modo de Transmissão
printf ("A1\r\n"); //Transmite dado
output_low(PIN_A4); //Habilita Max485 em Modo de Recepção
//#############################################################################
```

```
output_low(pin_B7);// Apaga
delay_ms (1000);
}}      }}
```

7.4. PROBLEMAS FÍSICOS DO PADRÃO EIA-485

Quando todos os dispositivos estão em modo de recepção, o nível lógico do barramento pode ficar indefinido, assim adicionam-se resistores de pull-up no pino A e pull-down no pino B.

Outro problema, já comentado, que ocorre é a reflexão do sinal devido a capacitância e indutância da linha, este problema pode ser evitado colocando-se dois **resistores de terminação** de igual valor (aproximadamente 120Ω) entre as linhas A e B.

São encontrados no mercado **circuitos integrados transceptores idênticos**, como **MAX 485 e DS75176**, dedicados a implementar interfaces de comunicação de microcontroladores como 8051 e família PIC no padrão RS-485.

A **isolação ótica** da interface de comunicação é interessante em linhas de comunicação com distancias significativas e previne a queima dos microprocessadores em caso de sobre-tensões de origem atmosférica. Esta isolação está presente dentro dos circuitos integrados mais recentes.

8. MEMÓRIAS DO MICROCONTROLADOR

O microcontrolador apresenta diversos tipos de memória, entre elas:

8.1. MEMÓRIA DE PROGRAMA

A memória de programa *flash*, é o local onde são gravados o código hexadecimal 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.

8.2. 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.

8.4. 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=read_eeprom (posicao);**. O projeto de controlde de acesso com teclado matricial abaixo mostra o uso desta memória.

1.5. MEMÓRIA DE DADOS (RAM)

A memória RAM do microcotrolador 18F2550 possui 2Kbytes 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.

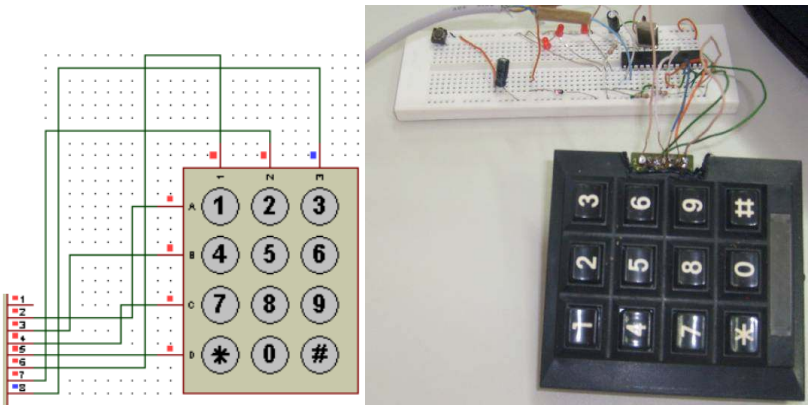
Esses registros podem ser configurados bit a bit através do seu endereço com a diretiva **#byte** que funciona para apontar uma posição de memória (#byte OSCCON=0XFD3 -> OSCCON=0B01100110; //Configura oscilador interno para 4MHz) ou por funções proprietárias do compilador (setup_timer_1 (T1_INTERNAL | T1_DIV_BY_8); set_timer1(3036); // configura o timer 1 com clock interno e dividido por 8 para contar 62500 = 65536-3036 – Tempo total = 62500 x 8us = 0,5 segundos).

DATA RAM -> 000h a 7FFh (2Kbytes)
SFR -> F60h a FFFh SPECIAL FUNCTION REGISTER MAP FOR PIC18F2550

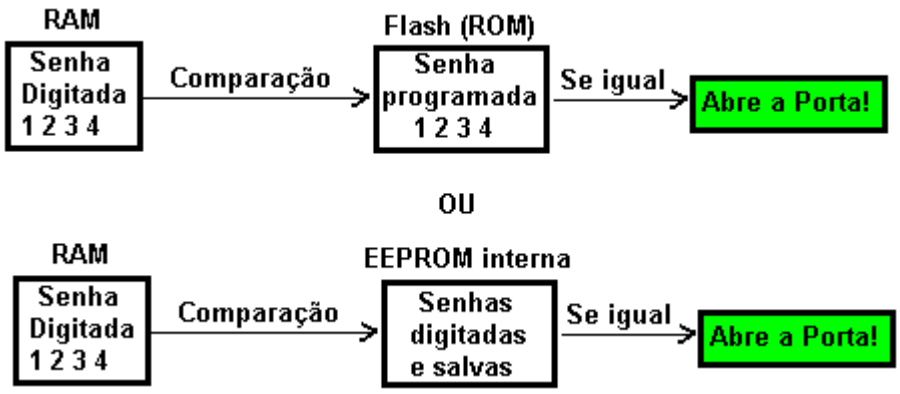
Address	Name		Address	Name		Address	Name		Address	Name		Address	Name	
FFFh	TOSU	Pilha	FDFh	INDF2(1)		FBFh	CCPR1H		F9Fh	IPR1		F7Fh	UEP15	USB
FFEh	TOSH		FDEh	POSTINC2(1)		FBEh	CCPR1L		F9Eh	PIR1		F7Eh	UEP14	
FFDh	TOSL		FDDh	POSTDEC2(1)		FBDh	CCP1CON		F9Dh	PIE1		F7Dh	UEP13	
FFCh	STKPTR		FDCh	PREINC2(1)		FBCh	CCPR2H		F9Ch	—(2)		F7Ch	UEP12	
FFBh	PCLATU	PC	FDBh	PLUSW2(1)		FBBh	CCPR2L		F9Bh	OSCTUNE		F7Bh	UEP11	
FFAh	PCLATH		FDAh	FSR2H		FBAh	CCP2CON		F9Ah	—(2)		F7Ah	UEP10	
FF9h	PCL		FD9h	FSR2L		FB9h	—(2)		F99h	—(2)		F79h	UEP9	
FF8h	TBLPTRU		FD8h	STATUS		FB8h	BAUDCON		F98h	—(2)		F78h	UEP8	
FF7h	TBLPTRH		FD7h	TMR0H	TIMER 0	FB7h	ECCP1DEL		F97h	—(2)		F77h	UEP7	
FF6h	TBLPTRL		FD6h	TMR0L		FB6h	ECCP1AS		F96h	TRISE(3)		F76h	UEP6	
FF5h	TABLAT		FD5h	T0CON		FB5h	CVRCON		F95h	TRISD(3)		F75h	UEP5	
FF4h	PRODH		FD4h	—(2)		FB4h	CMCON		F94h	TRISC	I/O PORTAS	F74h	UEP4	
FF3h	PRODL		FD3h	OSCCON	OSCILADOR	FB3h	TMR3H	TIMER 3	F93h	TRISB		F73h	UEP3	
FF2h	INTCON	INTERRUPÇÕES	FD2h	HLVDCON		FB2h	TMR3L		F92h	TRISA		F72h	UEP2	
FF1h	INTCON2		FD1h	WDTCON		FB1h	T3CON		F91h	—(2)		F71h	UEP1	
FF0h	INTCON3		FD0h	RCON		FB0h	SPBRGH		F90h	—(2)		F70h	UEP0	
FEFh	INDF0(1)		FCFh	TMR1H	TIMER 1	FAFh	SPBRG		F8Fh	—(2)		F6Fh	UCFG	
FEEh	POSTINC0(1)		FCEh	TMR1L		FAEh	RCREG		F8Eh	—(2)		F6Eh	UADDR	
FEDh	POSTDEC0(1)		FCDh	T1CON		FADh	TXREG	USART	F8Dh	LATE(3)		F6Dh	UCON	
FECh	PREINC0(1)		FCCh	TMR2	TIMER 2	FACh	TXSTA		F8Ch	LATD(3)		F6Ch	USTAT	
FEBh	PLUSW0(1)		FCBh	PR2		FABh	RCSTA		F8Bh	LATC		F6Bh	UEIE	
FEAh	FSR0H		FCAh	T2CON		FAAh	—(2)		F8Ah	LATB		F6Ah	UEIR	
FE9h	FSR0L		FC9h	SSPBUF		FA9h	EEADR		F89h	LATA		F69h	UIE	
FE8h	WREG		FC8h	SSPADD		FA8h	EEDATA		F88h	—(2)		F68h	UIR	
FE7h	INDF1(1)		FC7h	SSPSTAT		FA7h	EECON2(1)		F87h	—(2)		F67h	UFRMH	
FE6h	POSTINC1(1)		FC6h	SSPCON1		FA6h	EECON1		F86h	—(2)		F66h	UFRML	
FE5h	POSTDEC1(1)		FC5h	SSPCON2		FA5h	—(2)		F85h	—(2)		F65h	SPPCON(3)	
FE4h	PREINC1(1)		FC4h	ADRESH	AD	FA4h	—(2)		F84h	PORTE		F64h	SPPEPS(3)	
FE3h	PLUSW1(1)		FC3h	ADRESL		FA3h	—(2)		F83h	PORTD(3)		F63h	SPPCFG(3)	
FE2h	FSR1H		FC2h	ADCON0		FA2h	IPR2		F82h	PORTC	PORTAS	F62h	SPPDATA(3)	
FE1h	FSR1L		FC1h	ADCON1		FA1h	PIR2		F81h	PORTB		F61h	—(2)	
FE0h	BSR		FC0h	ADCON2		FA0h	PIE2		F80h	PORTA		F60h	—(2)	

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.



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-definida contida na memória de programa *flash* (“ROM”) ou na EEPROM interna.



PONTEIROS

Ponteiros guardam 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 <SanUSB.h>
#include <usb_san_cdc.h>// Biblioteca para comunicação serial


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;
```

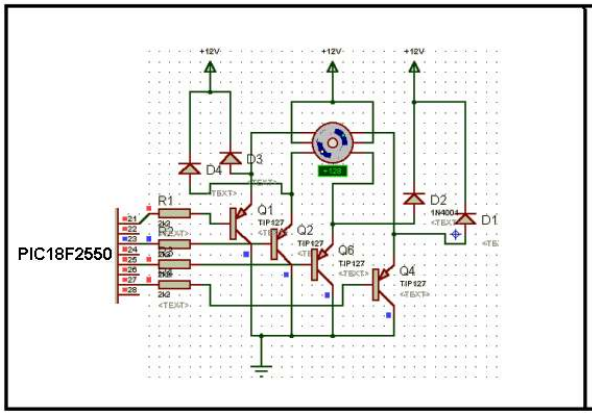
```
////////////////////////////////////////////////////////////////////////////
#int_timer1 // Interrupção do timer 1
void trata_t1 () // Conta 62.500us x 8 = 0,5s
{--mult;
if (!mult)
        {mult=8; // 8 x0,5s - 4 seg
        p=100; tecla0='F';tecla1='F';tecla2='F';tecla3='F'; // volta a posição de origem a cada 4 seg

                                }}
////////////////////////////////////////////////////////////////////////////
main() {
clock_int_4MHz();
port_b_pullups(true);
   usb_cdc_init(); // Inicializa o protocolo CDC
   usb_init();    // Inicializa o protocolo USB
   usb_task();  // Une o periférico com a usb do PC


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(read_eeprom(239)>0 && read_eeprom(239)<40) {k=read_eeprom(239);} // Carrega a útima posição livre da eeprom (k) antes
do reset armazenada em 239

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

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

output_high(pin_b0);output_high(pin_b1);output_low(pin_b2);
if(input(pin_b3)==0) {*p='3'; flag=1; while(input(pin_b3)==0);delay_ms(200);}
if(input(pin_b4)==0) {*p='6'; flag=1; while(input(pin_b4)==0);delay_ms(200);}
if(input(pin_b5)==0) {*p='9'; flag=1; while(input(pin_b5)==0);delay_ms(200);}
if(input(pin_b6)==0) {*p='!'; flag=1; while(input(pin_b6)==0);delay_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 (usb_cdc_putc,"\r\nValor das teclas digitadas: %c %c %c %c\r\n",tecla0,tecla1,tecla2,tecla3);
printf (usb_cdc_putc,"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
output_high(pin_c0);printf (usb_cdc_putc,"\r\nSenha Mestre!\r\n");delay_ms(1000); output_low(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 (usb_cdc_putc,"\r\nSenha armazenada\r\n");

write_eeprom( 239, k); printf(usb_cdc_putc,"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(usb_cdc_putc,"%2x ", read_eeprom(i*40+j) );}//Leitura da eeprom interna

               printf(usb_cdc_putc,"\r\n");
          }
                                                  }
//*****************************************************************************

// Compara conjunto de teclas pressionadas com senhas armazenadas na eeprom
if (flag2==1) {
for(n=0;n<=k;n++)
        {
if       (tecla0==read_eeprom(4*n)       &&       tecla1==read_eeprom(4*n+1)       &&       tecla2==read_eeprom(4*n+2)&&
tecla3==read_eeprom(4*n+3))
{ output_high(pin_c0); printf (usb_cdc_putc,"\r\nAbre a porta!\r\n");delay_ms(3000); output_low(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);
delay_ms(100);                }}
```

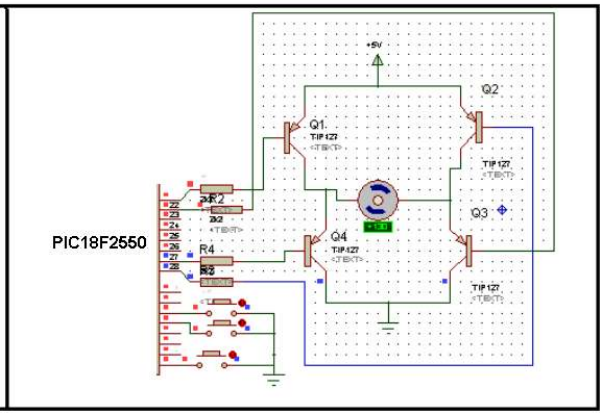
2. ACIONAMENTO DE MOTORES MICROCONTROLADOS

Os motores mais utilizados com sistemas microcontrolados são os motores CC , motores de passo e servo-motores. A figura abaixo mostra a disposição dos transistores de potência para atuação bidirecional de motores de passo e motores CC.

ACIONAMENTO BIDIRECIONAL MOTOR DE PASSO



ACIONAMENTO BIDIRECIONAL MOTOR CC - PONTE H



ACIONAMENTO DE MOTORES CC DE BAIXA TENSÃO

MOTORES DE EQUIPAMENTOS ELETRÔNICOS

São abundantes no mercado em função da ampla gama de utilização, conseqüentemente, existem em várias dimensões, tensões, pesos, características e são fáceis de encontrar em sucatas como video-cassete, brinquedos, impressoras, etc.

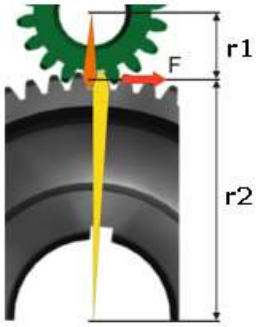
MOTORES DE ELÉTRICOS UTILIZADOS EM AUTOMÓVEIS

Os motores utilizados em automóveis são todos com tensão nominal a 12 volts, 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.

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.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:

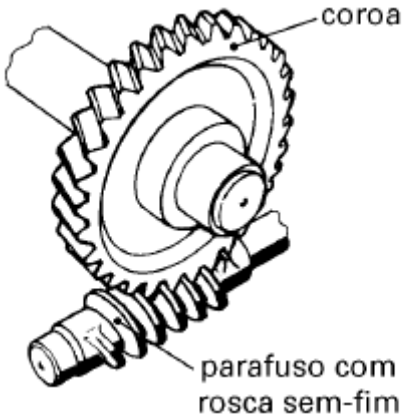
$$Fconst \rightarrow T_1/r_1 = T_2/r_2 \rightarrow T_1/D_1 = T_2/D_2$$

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

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

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:

$$i = Nc \cdot Zc = Np \cdot Zp$$

$$\mathbf{Nc = Np \cdot Zp / Zc}$$

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

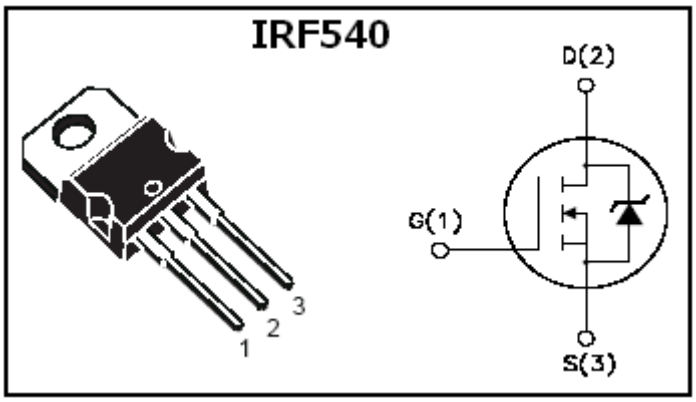
As possibilidades mais comuns de controle digital de motores CC são:

CHAVEAMENTO DE MOTORES CC COM TRANSISTORES MOSFET

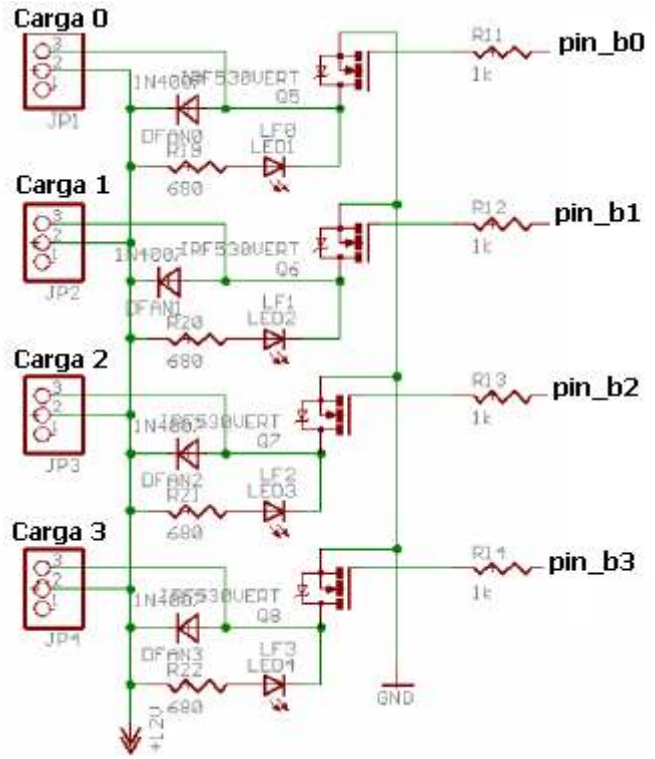
Os transistores de efeito de campo MOSFET executam o chaveamento por tensão na base e podem ser utilizados no lugar dos transistores Darlington para acionamento de dispositivos de média potência, devido principalmente à menor queda de tensão e à menor dissipação em forma de calor.

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

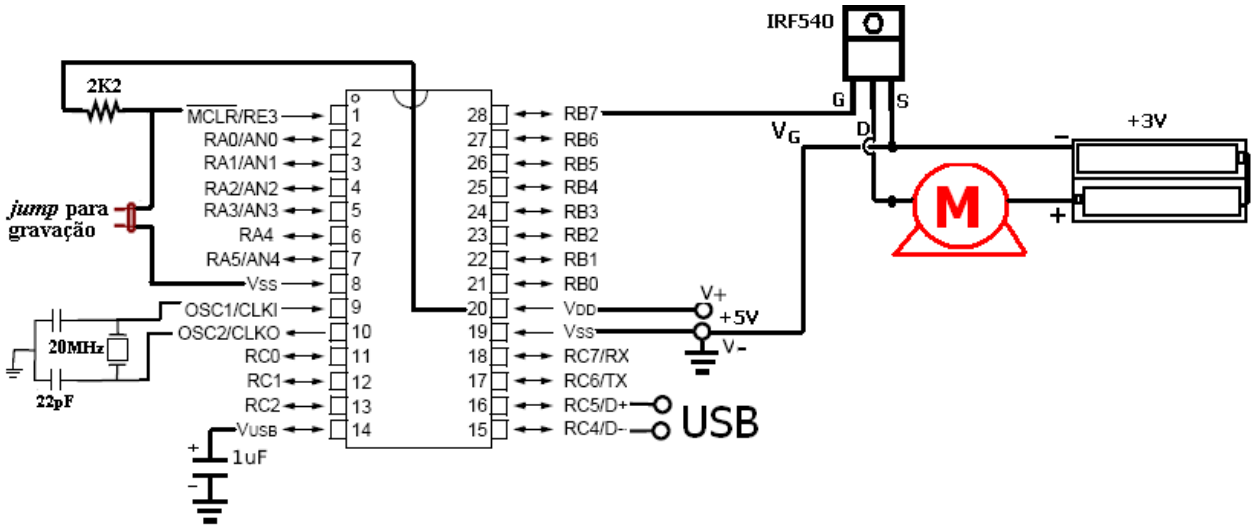
baixa, o que não acontece com transistores Darlington. A fgura abaixo mostra a configuração dos pinos de um MOSFET, onde o pino 1 é o *Gate* (base), o pinos 2 é o *Drain* e o pino 3 é o *Source*.



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).



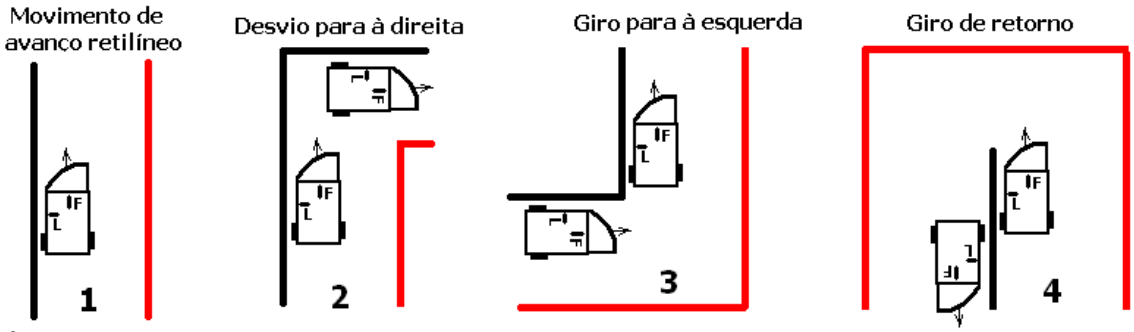
A figura abaixo mostra o acionamento de um motor CC de 3V utilizado em robótica móvel através de um mosfet IRF540.



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:



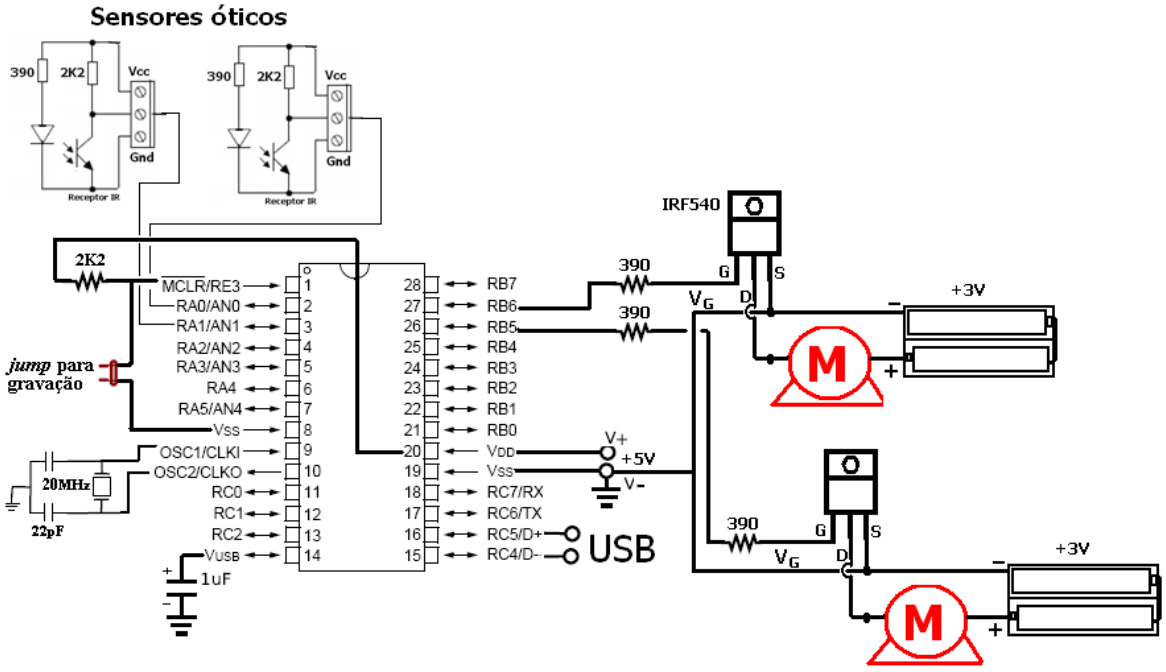
É importante salientar que no princípio de seguir o obstáculo, o robô não pode tocar no outro lado, em vermelho, pois ele pode tomá-lo como referência e voltar, o que fará sair por onde entrou. Mais detalhes: http://www.youtube.com/watch?v=QRDrG2iEFpM

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 circuito é mostrado abaixo:

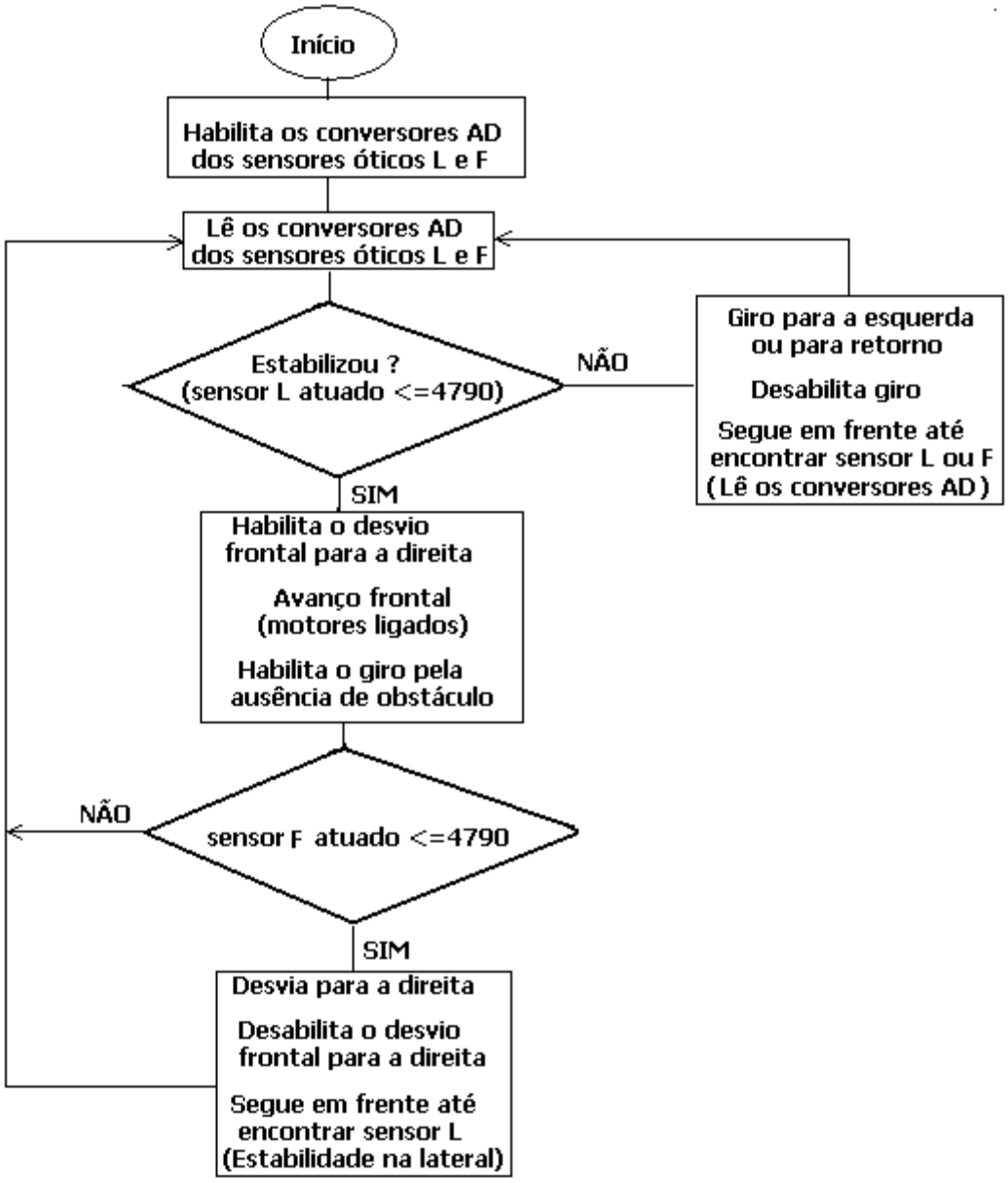


Programa:

```
#include <SanUSB.h> //Leitura de tensão em mV com variação de um potenciômetro

#define esquerdo pin_b6
#define direito  pin_b5

int32 tensaofrente, tensaolateral, aproximacao=4790; //int32 devido aos cálculos intermediários
short int ledpisca;
unsigned int flagnaosensor=0, flagfrente=0, flaggiro=1;

main() {
clock_int_4MHz();

setup_adc_ports(AN0_TO_AN1);  //Habilita entrada analógica - A0
setup_adc(ADC_CLOCK_INTERNAL);

while(1){
output_high(esquerdo);output_high(direito);  //Os dois motores em frente
//*****************************************************************************
set_adc_channel(1);
delay_ms(10);
tensaolateral= (5000*(int32)read_adc())/1023;
if (tensaolateral<=4790)  {  flagnaosensor=0; flagfrente=0; flaggiro=0; } // Estabilizou, Habilita o giro e a virada de //frente para a
direita
//*****************************************************************************

if (flagnaosensor>=3 && flaggiro==0) {flagnaosensor=0; flaggiro=1; // Sem barreira (flagnaosensor>=3) gire 170 ///graus
                        output_low(esquerdo);output_low(direito); output_high(pin_b7);delay_ms(500);//pára
output_low(esquerdo);output_high(direito);  output_low(pin_b7);delay_ms(2000);  //só  gira  de  novo  se  tiver  //estabilizado  na
lateral (flaggiro==0)

                output_high(esquerdo);output_high(direito); // Segue em frente

                while (tensaofrente>4790 &&  tensaolateral>4790) //Espera até encontrar barreira frontal ou lateral
                {set_adc_channel(0);
                delay_ms(10);
```

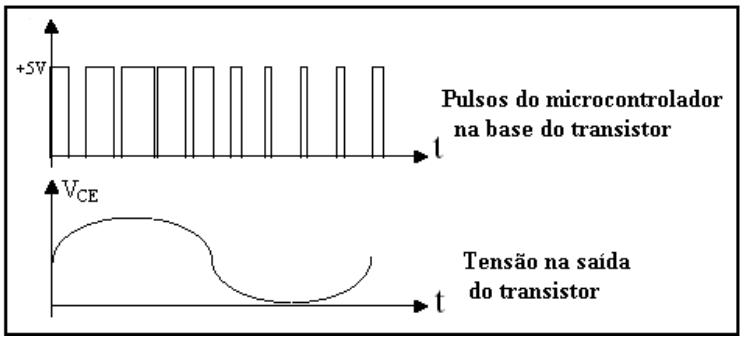
```
                tensaofrente= (5000*(int32)read_adc())/1023;
                set_adc_channel(1);
                delay_ms(10);
                tensaolateral= (5000*(int32)read_adc())/1023;
                                        }}
//*************************************************************************
                        //ANALÓGICO         DIGITAL(10 bits)
set_adc_channel(0);             // 5000 mV              1023
delay_ms(10);                   // tensao               read_adc()
tensaofrente= (5000*(int32)read_adc())/1023;
//*************************************************************************
if (tensaofrente<=4790 && flagfrente==0) {   flagnaosensor=0; flagfrente=1; //encontrou barreira frontal
                output_low(direito);delay_ms(500);//vira para a direita

                output_high(esquerdo);output_high(direito); // Segue em frente até estabilizar

                while (tensaolateral>4790) //Espera até estabilizar a lateral
                {set_adc_channel(1);       //Colado no canto frontal e lateral também estabiliza
                 delay_ms(10);
                 tensaolateral= (5000*(int32)read_adc())/1023;
                                        }}

++flagnaosensor;
output_high(esquerdo);output_high(direito);  //Os dois motores em frente
ledpisca=!ledpisca;
output_bit(pin_b7,ledpisca);
delay_ms(20);
                        }}
```

O fluxograma do programa do microcontrolador é mostrado abaixo:



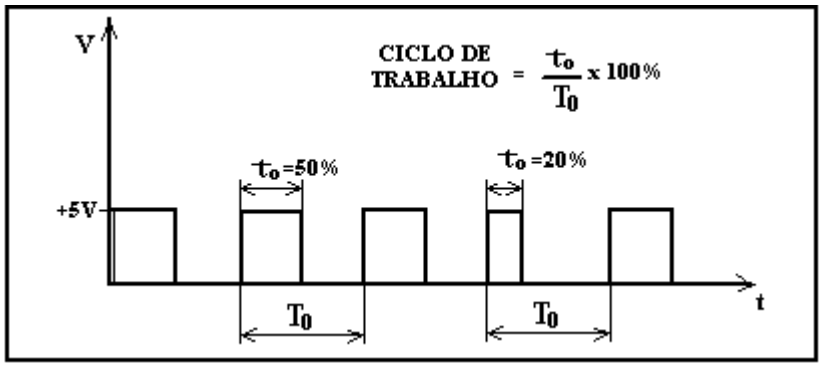
CONTROLE PWM DE VELOCIDADE DE UM MOTOR CC

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 freqüência.



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

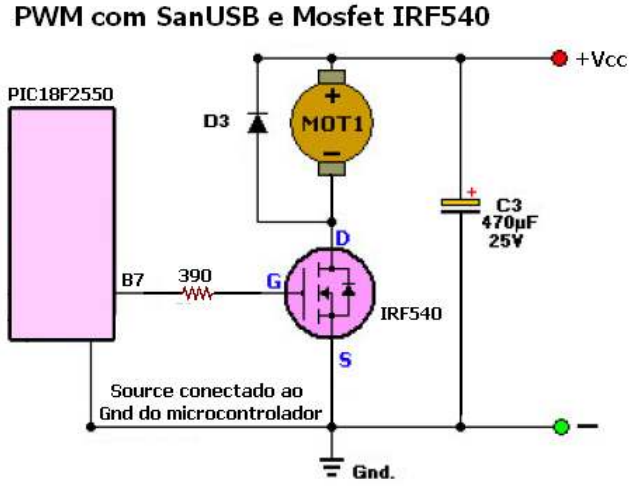
O período To é o intervalo de tempo que se registra o Período repetitivo do pulso e o τ_0 é o ciclo de trabalho (*duty-cicle*), ou seja, o tempo em que o pulso permanece com a amplitude em nível lógico alto.



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.

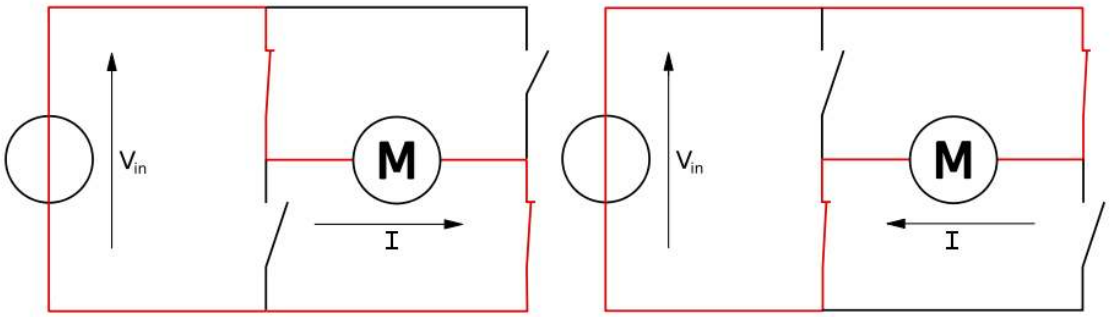
```
#include <SanUSB.h>

#define motor pin_b7
#define led pin_b0

unsigned int ton,toff,incton,inctoff,guardaton,guardatoff;
int1 flag1, flag2;

main() {
clock_int_4MHz();

port_b_pullups(true);
incton=2; inctoff=18; //Período de 20ms - Passo mínimo de tempo = 1ms (5% (1/20) do duty cicle )

guardaton=read_eeprom(10);guardatoff=read_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;output_high(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;output_low(led);
write_eeprom(10,incton);write_eeprom(11,inctoff);
                                    }

        output_high(motor);
        while(ton) {--ton;delay_ms(1); } //Parte alta da onda quadrada

        output_low(motor);
        while(toff) {--toff;delay_ms(1); } //Parte baixa da onda quadrada
                                                    }}
```

A figura abaixo mostra o circuito montado para com este exemplo. Veja o funcionamento desse circuito no vídeo http://video.google.com/videoplay?docid=-8422519485287713O4&hl=en# .



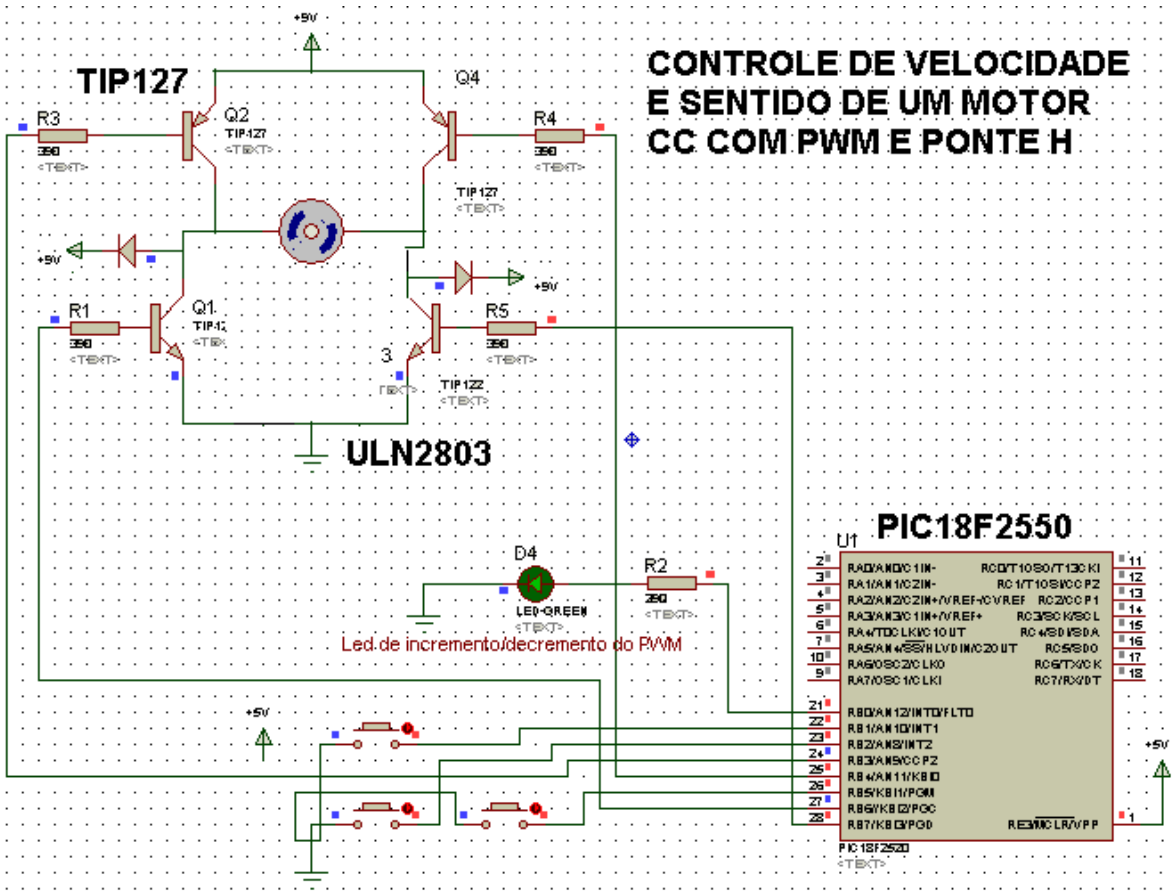
PONTE H

O acionamento da ponte H 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.



PONTE H COM PWM:

O exemplo abaixo mostra o controle de velocidade de um motor CC por PWM nos dois sentidos com uma ponte H, o incmento da onda quadrada de tensão é feito no pino B2 e o decremento no pino B3. O sentido de rotação é invertido com um botão no pino B5. Na prática no lugar dos transitores é utilizado o driver ULN2803 que contém internamente oito transistores com diodos de roda livre. A figuras abaixo mostra a foto do circuito de simulação deste exemplo que pode ser visto em http://www.youtube.com/watch?v=cFTYHBTEBh8 .




```
#include <SanUSB.h>

#define esquerdavcc pin_b3
#define direitavcc pin_b4
#define esquerdagnd pin_b6
#define direitagnd pin_b7
#define led pin_b0

unsigned int16 i;
unsigned int ton,toff,incton,inctoff,guardaton,guardatoff;
int1 flag1, flag2, flag3;

main() {
OSCCON=0B01100110;

port_b_pullups(true);
incton=2; inctoff=18; //Período de 20ms - Passo mínimo de tempo = 1ms (5% (1/20) do duty cicle )

guardaton=read_eeprom(10);guardatoff=read_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;output_high(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
```

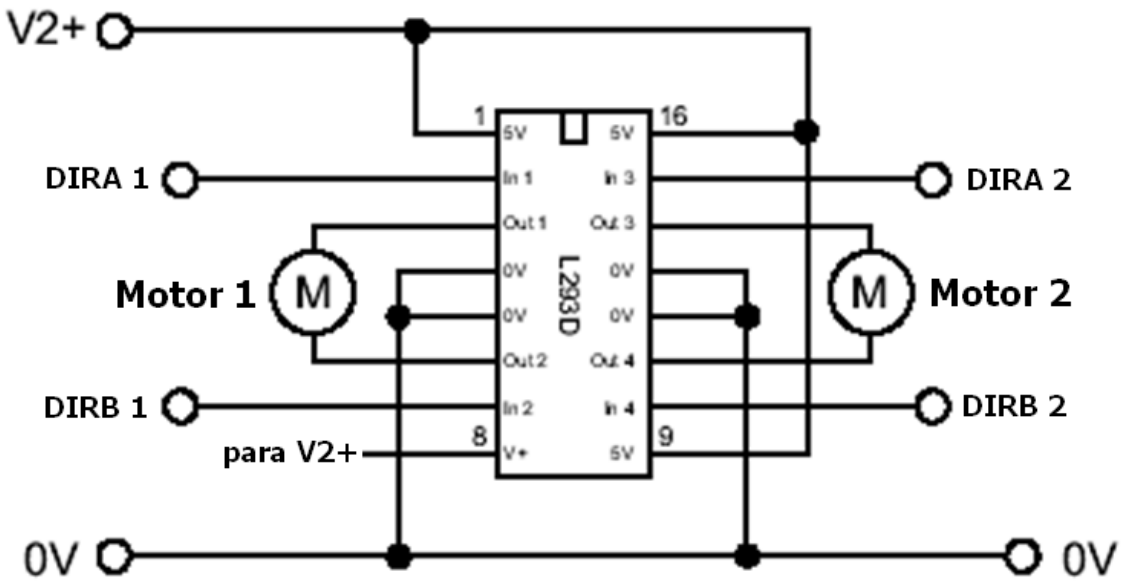
```
if (!input(pin_b2)) {flag2=1;}
if (inctoff<20 && flag2==1 && input(pin_b2) ) {flag2=0;++inctoff;--incton;output_low(led);
write_eeprom(10,incton);write_eeprom(11,inctoff);
                                    }
if (!input(pin_b5)) { // Bateu recuou
              for(i=0; i<400; ++i) //Volta 400 períodos de 20ms
                {  ton=incton; toff=inctoff;

                output_high(esquerdavcc);output_low(direitagnd);
                output_low(direitavcc);output_high(esquerdagnd);
                while(ton) {--ton;delay_ms(1); } //Parte alta da onda quadrada

                output_high(direitavcc);output_low(esquerdagnd);
                while(toff) {--toff;delay_ms(1); }
                                                  }}


        output_high(direitavcc);output_low(esquerdagnd); //Bloqueia o outro  lado
        output_low(esquerdavcc);output_high(direitagnd);
        while(ton) {--ton;delay_ms(1); } //Parte alta da onda quadrada

        output_high(esquerdavcc);output_low(direitagnd);
        while(toff) {--toff;delay_ms(1); } //Parte baixa da onda quadrada
                                                  }}
```

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. 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.



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. A Tabela permite programar o movimento em qualquer direção (conjugando 2 motores).

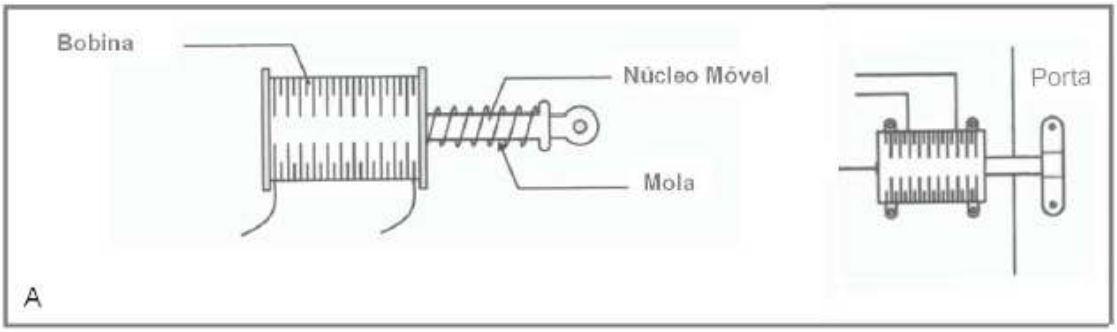
ENABLE	DIRA	DIRB	FUNÇÃO
H	H	L	Para Frente
H	L	H	Para trás
H	L/H	L/H	Stop Rápido
L	X	X	Stop Lento

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.

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 como é

mostrado na figura abaixo. No caso de um relé, fecha um contato para circulação de outro nivel maior de corrente.



Os relés são dispositivos comutadores eletromecânicos (Figura 2.14). A estrutura simplificada de um relé é mostrada na figura abaixo e a partir dela é explicado o seu princípio de funcionamento.

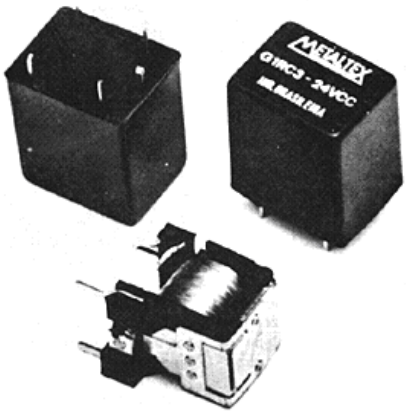
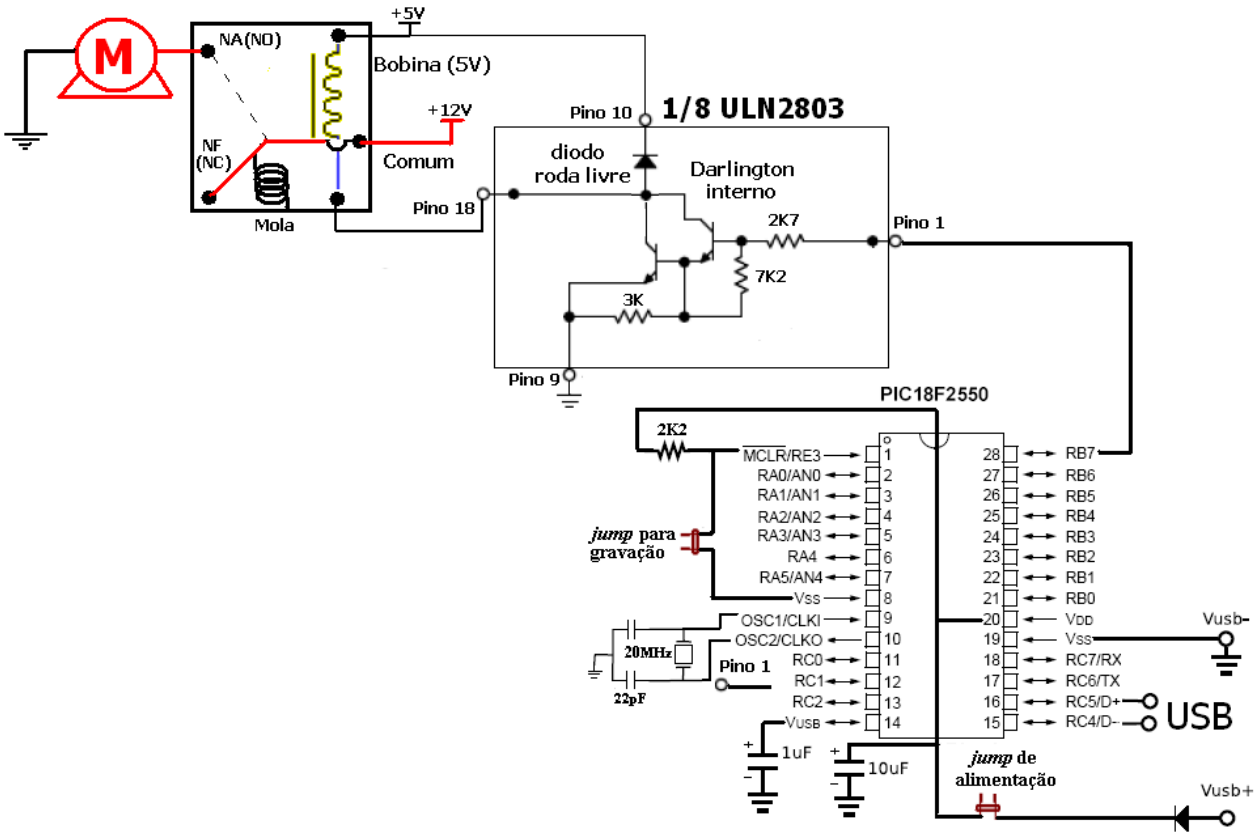


Figura 2.14 - Relé

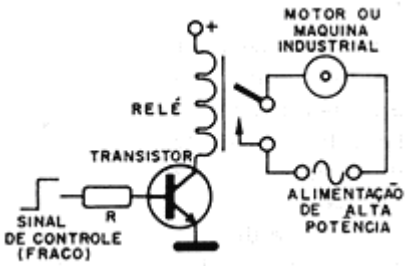
O controle de uma solenóide ou relé pode ser feito pelo chaveamento de um transistor Darlington ou de um MOSFET mostrado abaixo.

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 eletromagnética, 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 abaixo. 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.



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.

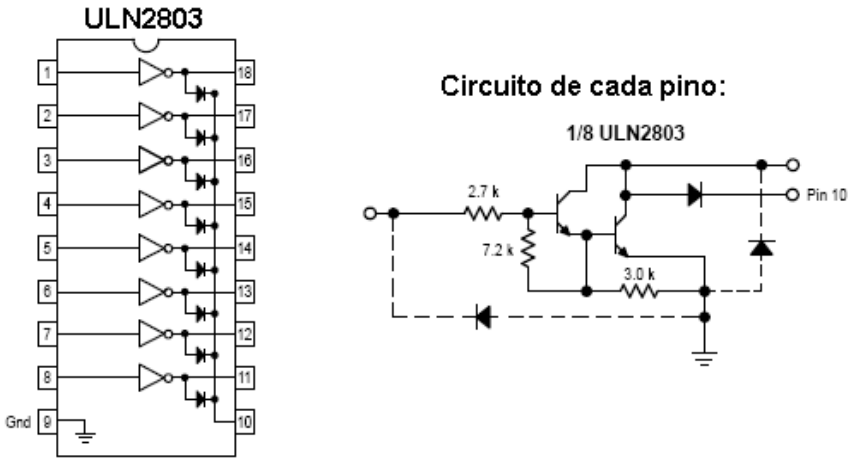


Aplicação de um relé

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.

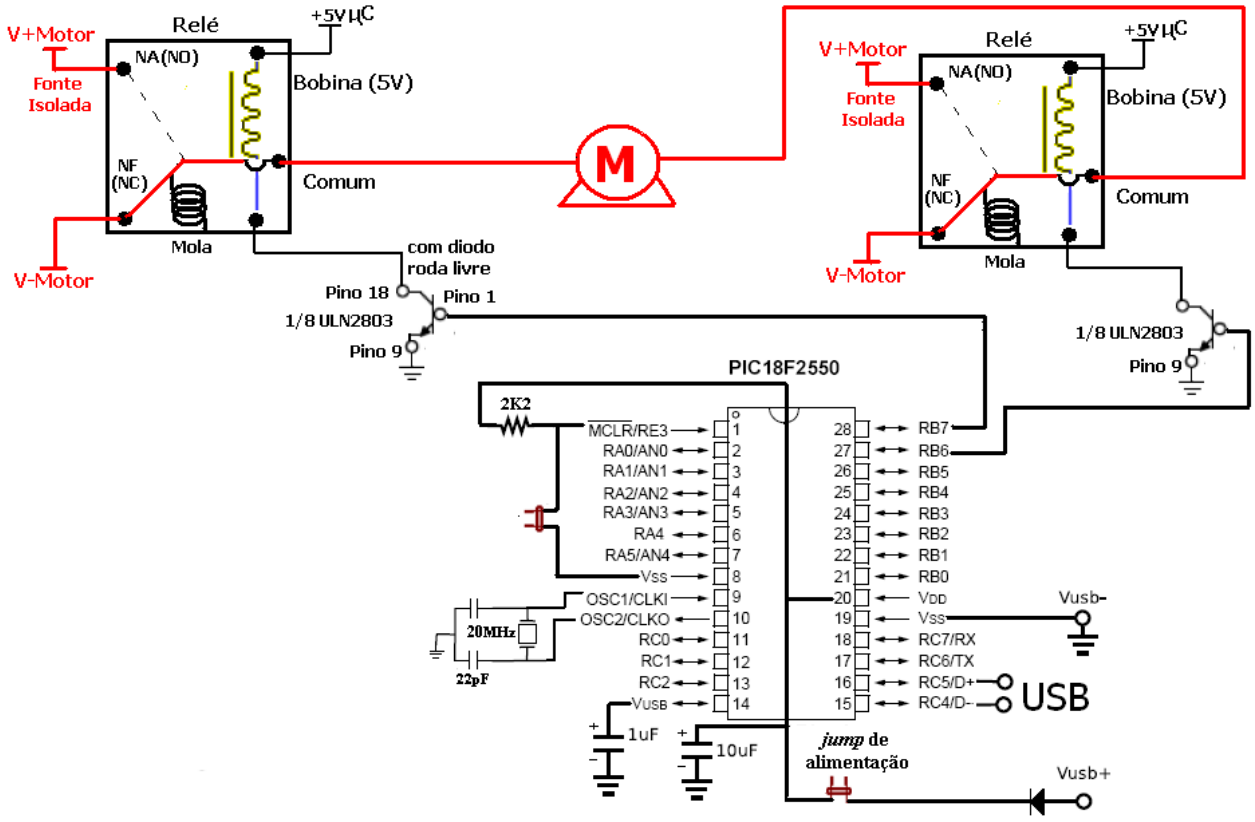
Quando o microcontrolador coloca +5V (nível lógico 1) no pino 1 do driver ULN2803, ele conecta o pino 18 do outro lado, onde está liga um pólo do motor, ao Gnd (nível lógico 0, por isso a simbologia de porta inversora na figura abaixo). Como o outro lado da bobina (o comum) ou do motor deve estar ligado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



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 abaixo, 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.



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.

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.

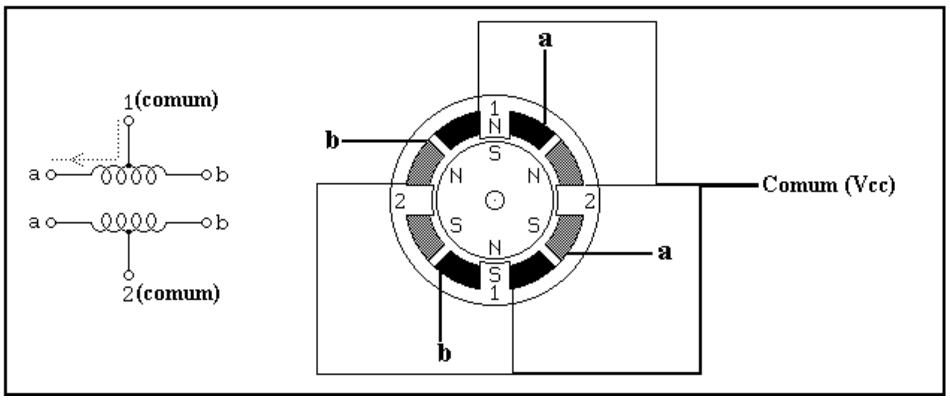
MOTORES DE PASSO UNIPOLARES

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).

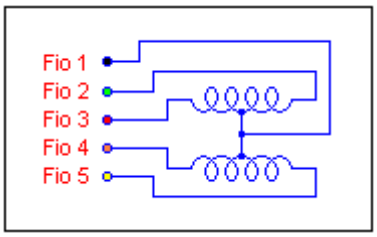
Normalmente, a derivação central das bobinas está ligada ao terminal positivo da fonte de alimentação (Vcc) e os terminais de cada bobina são ligados alternadamente à terra através de chaveamento eletrônico produzindo movimento.

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.

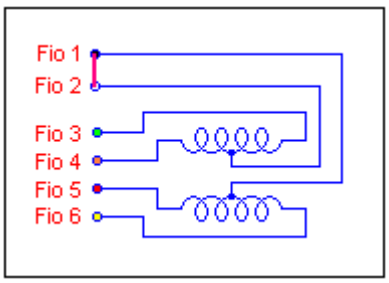


Os motores de passo unipolares mais encontrados possuem 5 ou 6 fios. Os motores de passo unipolares de 6 fios possuem dois **fios comuns (derivação central)**. Para o acionamento do motor de passo, estes fio comuns devem ser ligados à fonte de alimentação (+5V ou +12V) e os terminais da bobina ligados ao controle de chaveamento do motor de passo.

Motor de 5 fios



Motor de 6 fios



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.

MODOS DE OPERAÇÃO DE UM MOTOR DE PASSO UNIPOLAR

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;

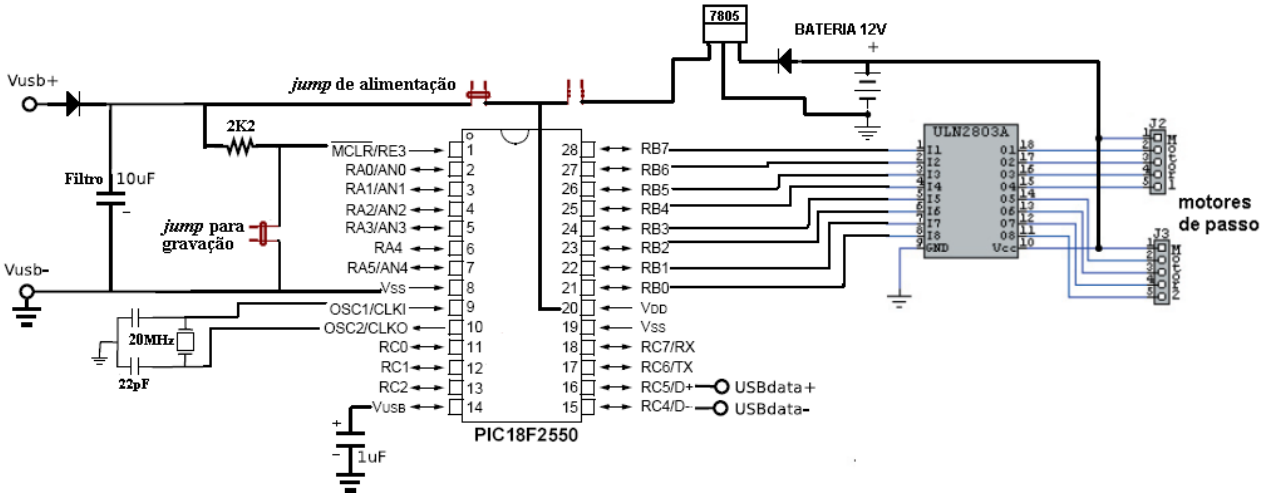
-Maior torque;

-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

ACIONAMENTO BIDIRECIONAL DE DOIS MOTORES DE PASSO

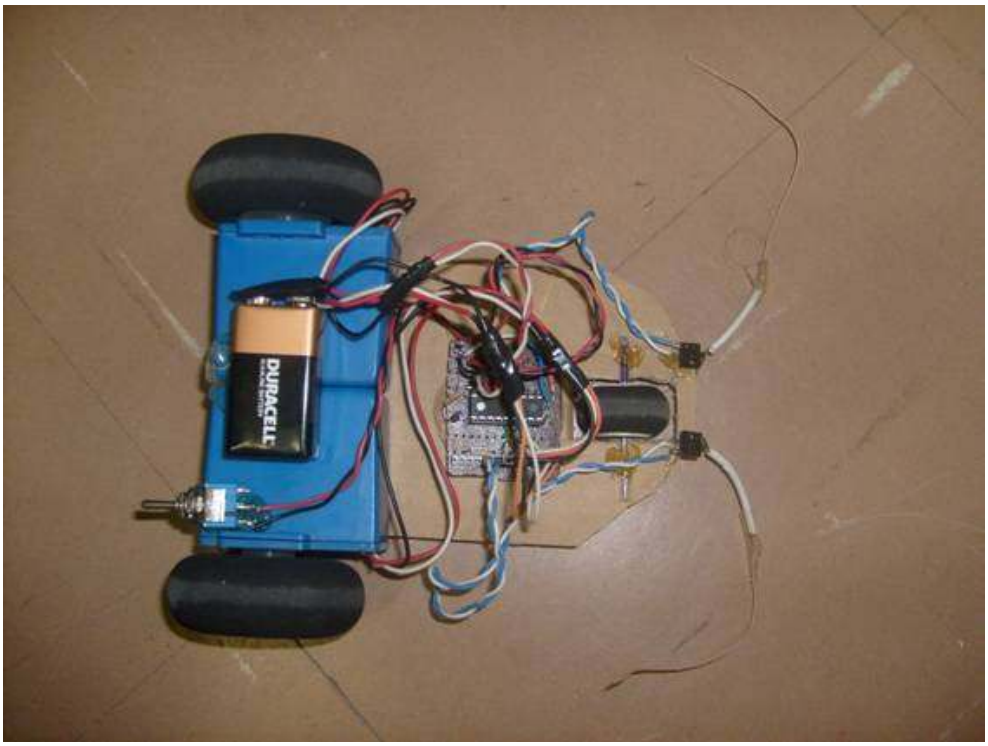
Como o driver de potência ULN2803 ou ULN2804 possui internamente 8 transistores de potência ele é capaz de manipular dois motores de passo ao mesmo tempo. Ele contém internamente oito diodos de roda livre e oito resistores de base dos transistores, o que possibilita a ligação direta ao microcontrolador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



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 link abaixo mostra esse motor utilizando a ferramenta SanUSB http://www.youtube.com/watch?v=vaegfA65Hn8.

SERVO-MOTORES

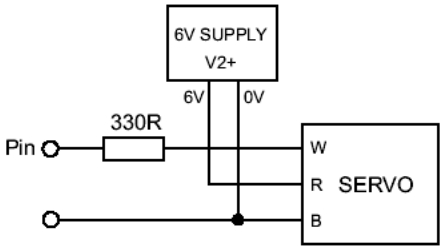
Os servos são muito utilizados em parabólicas, em carros e aviões radiocontrolados. Trata-se de dispositivos muito precisos que giram sempre o mesmo ângulo para um dado sinal. Um servo típico possui três condutores de ligação, normalmente preto, vermelho e branco (ou amarelo). O condutor preto é a referência 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 abaixo que é um carro para desvio de obstáculos, acionado por dois servo-motores. Este sinal de posição é normalmente um impulso de 1 a 2 milisegundos (ms), repetido depois de um pulso de 10 a 20ms. Com o pulso de aproximadamente 0,75 ms o servo move-se para um sentido e com o impulso de 2,25 ms para o sentido oposto. Desse modo, com um impulso de 1,5 ms, o servo roda para a posição central.

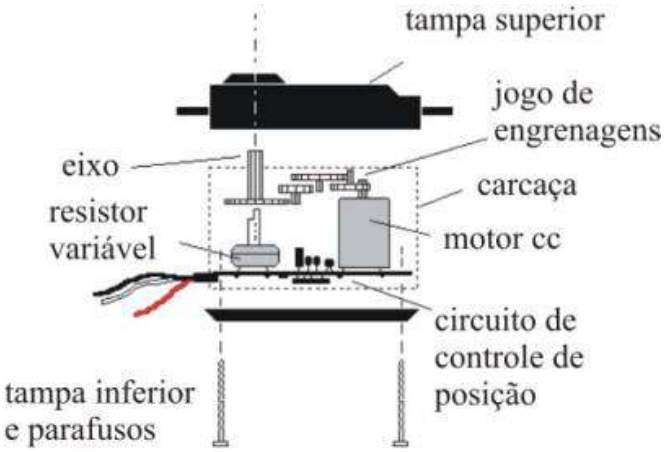


A tensão de alimentação do servomotor é tipicamente de 5V, podendo variar entre 4,5V e 6V. Devido à alta redução do jogo de engrenagens, o torque que se obtém de um servo é bastante alto, considerando o seu tamanho reduzido. Lamentavelmente, os servos consomem correntes elevadas (de 200 mA a 1 A) e introduzem ruído elétrico nos condutores de alimentação, necessitando a aplicação de capacitores de filtro ou uso de fontes isoladas, uma para o controle e outra para o motor. Lembre-se que o condutor de massa (referência) deve ser comum às duas alimentações. O programa abaixo move um servo-motor para frente e outro para trás. Note que essa operação é utilizada por robôs móveis que possuem dois motores em anti-paralelo.

```
#include <SanUSB.h>

#define motor1 pin_b5
#define motor2 pin_b6

int16 frente=50;
short int led;
main(){
clock_int_4MHz();

while (TRUE)
{
```

```
while (frente>0)
        {  output_high(motor2);          //Inicializa o pulso do motor 1
           output_high(motor1);          //Inicializa o pulso do motor 2
           delay_ms(1);
           output_low(motor1);           //Termina o pulso de 1ms do motor1– sentido horário
           delay_ms(1);

           output_low(motor1);
           output_low(motor2);           //Termina o pulso de 2ms do motor2 – sentido anti-horário
           delay_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);
}}
```

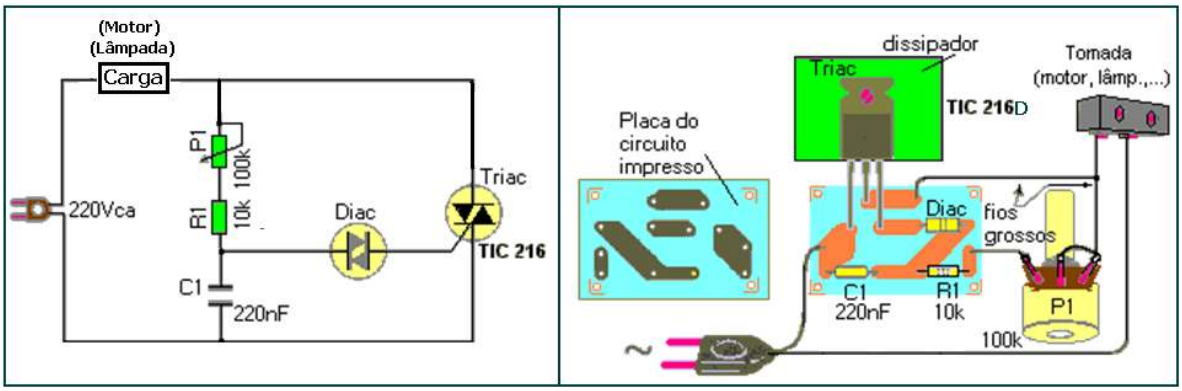




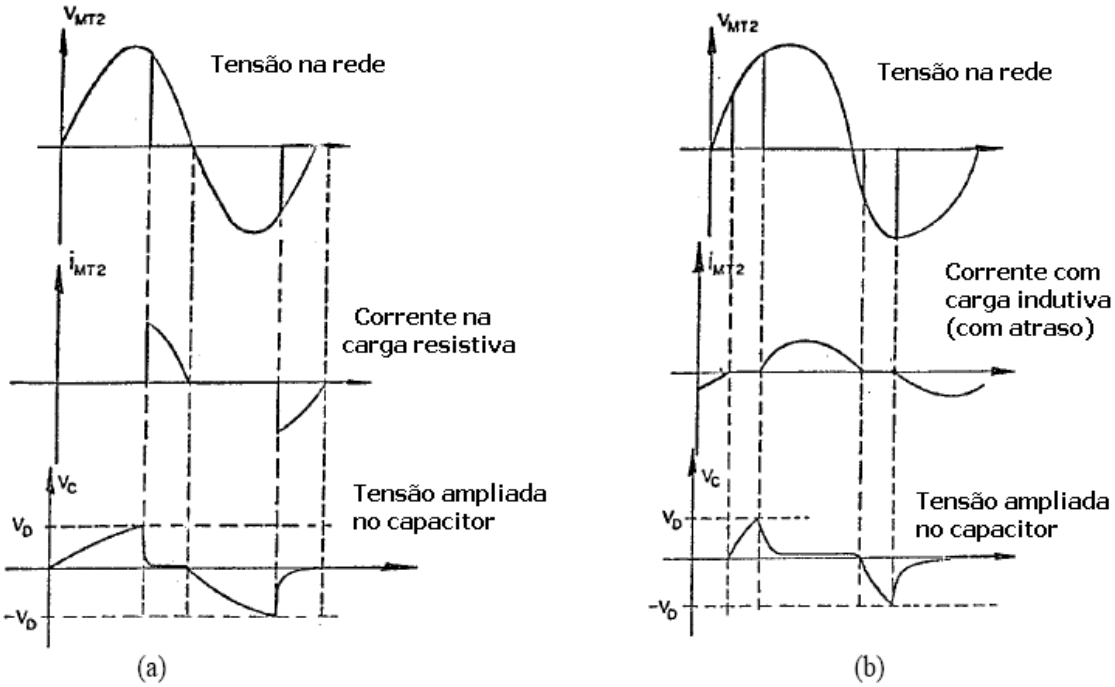
ACIONAMENTO DE CARGAS CA COM TRIAC

Como a maioria dos sistemas automáticos é acionada em corrente alternada, para o controle de fluxo de energia de tais sistemas é muito utilizada uma chave eletrônica, chamada TRIAC (*triode for Alternating Current*) da família dos tiristores, ou seja, um componente eletrônico equivalente a dois retificadores controlados de silício (SCR) ligados em antiparalelo que permite conduzir a corrente elétrica nos dois sentidos quando recebe um pulso na base (disparo). Durante a variação da onda senoidal, esse dispositivo apresenta um momento zero volts

e, conseqüentemente, corrente nula passando pelo triac, que o trava novamente até o próximo disparo.



As figuras abaixo mostram o comportamento das correntes em cargas resistivas e indutivas após o chaveamento do triac.



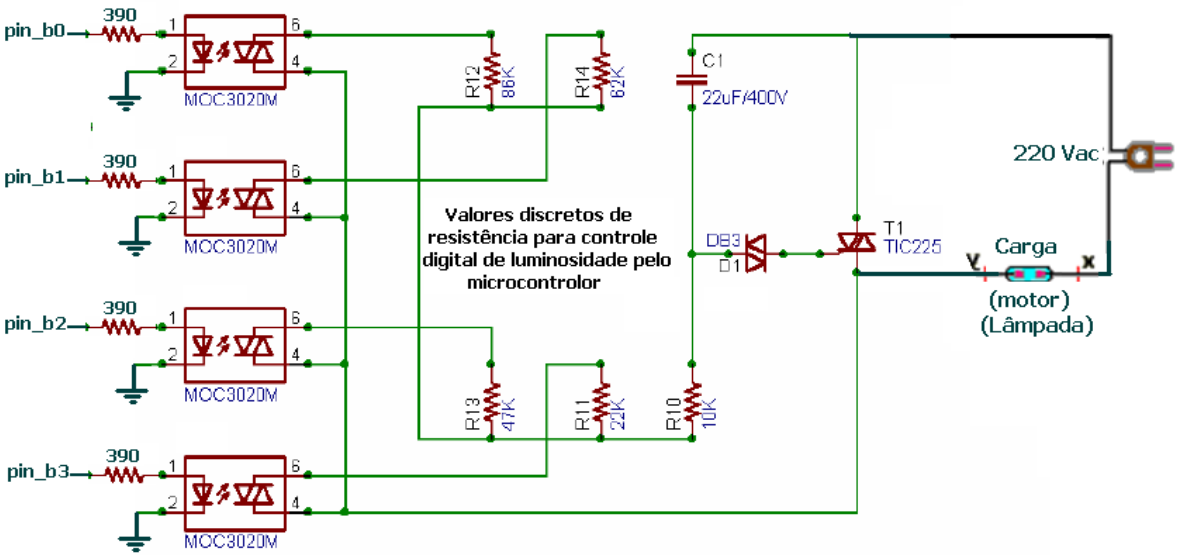
Suponha o TRIAC conduzindo. Quando a corrente inverte, passando pelo zero, o triac corta a alimentação da carga pela fonte CA e a partir daí a tensão sobre o capacitor começa a

crescer novamente em função do valor da capacitância e da resistência em série, até atingir a tensão de disparo (V_D) no DIAC (exemplos: 1N5411 e 40583) que dispara o gatilho do triac. O disparo provoca a descarga do capacitor e corta novamente o triac.

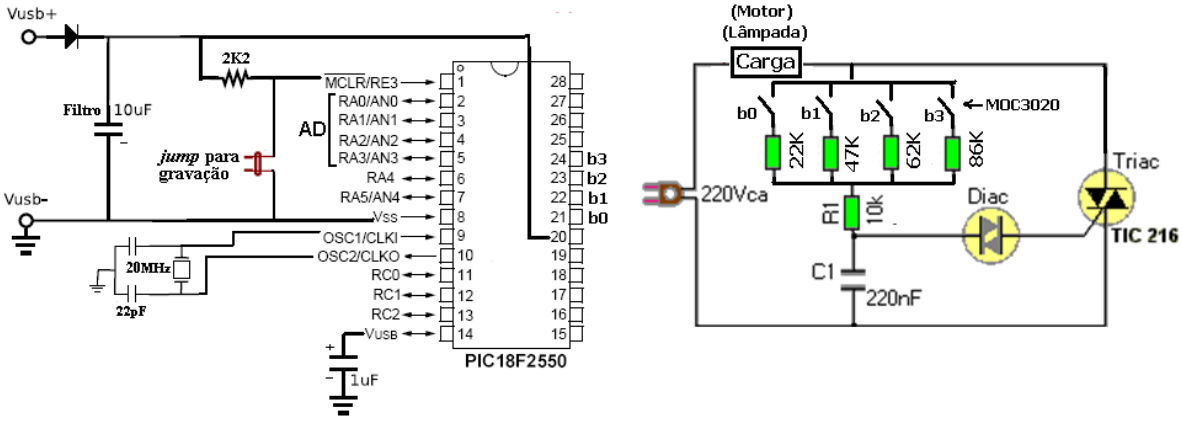
Note que sempre que a corrente passa pelo zero, o triac corta a alimentação da carga e só volta a conduzir quando o capacitor atinge a tensão de disparo, gerando uma onda senoidal "fatiada". O vídeo no link http://www.youtube.com/watch?v=q6QoMAAHZZw mostra a operação de um triac.

A figura abaixo apresenta um um circuito de controle digital de luminosidade por um microcontrolador, onde cada resistência conectada ao pino 6 do optoisolador MOC3021 altera o ângulo de disparo e, conseqüentemente, o tempo de condução do TRIAC BT139, que conduz até o final do semiciclo da onda senoidal.

.



Note que a figura abaixo representa a mesma disposição dos componentes do circuito acima.



Perceba que neste circuito, o tempo para carregar a tensão de disparo do capacitor C1 depende do valor das resitência em série associada. Como existem quatro resistores em paralelo que podem ser associadas ao capacitor, pode-se fazer o controle digital do tempo de disparo comutando as resistências atravás das chaves (MOC3020) de forma individual ou em paralelo. Dessa forma, é possível controlar a intensidade de corrente que será entregue para a carga, através da comutação dos resistores em paralelo.

Alguns modelos comerciais de triac são o BT139, o TIC206D e o TIC226D. Com eles é possível construir dimmers (controladores de luminosidade de lâmpadas) e também controladores de velocidade de motores CA.

Essa chave possibilita uma também forma de economia de energia, pois, enquanto ele não recebe o disparo, não há condução de corrente para a carga. Outra característica importante é a causa pela qual passou a substituir o reostato original, que acompanha os motores das máquinas de costura. Como o controle de velocidade é feito pela parcela do semiciclo aplicado e não pela tensão, o torque se mantém mesmo em baixas velocidades.

A outra série de MOC3041, já é projetada com circuito interno fixo para detecção de passagem por zero, assim o chaveamento é automático e não é possível nenhum controle do ângulo de disparo.

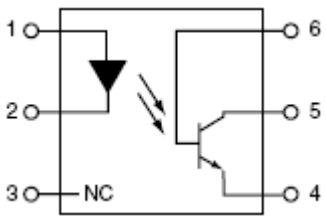
Também é possível controlar um motor CA de forma simples com o PWM em um período de 16ms, como mostra o video: http://www.youtube.com/watch?v=ZzDcUoYUnuk

É só utilizar como interface um optoacoplador MOC3021 (que recebe os pulsos PWM do PIC) e então comanda o chaveamento de um TRIAC TIC226D para controlar a onda senoidal da carga CA.

FOTOACOPLADORES

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:

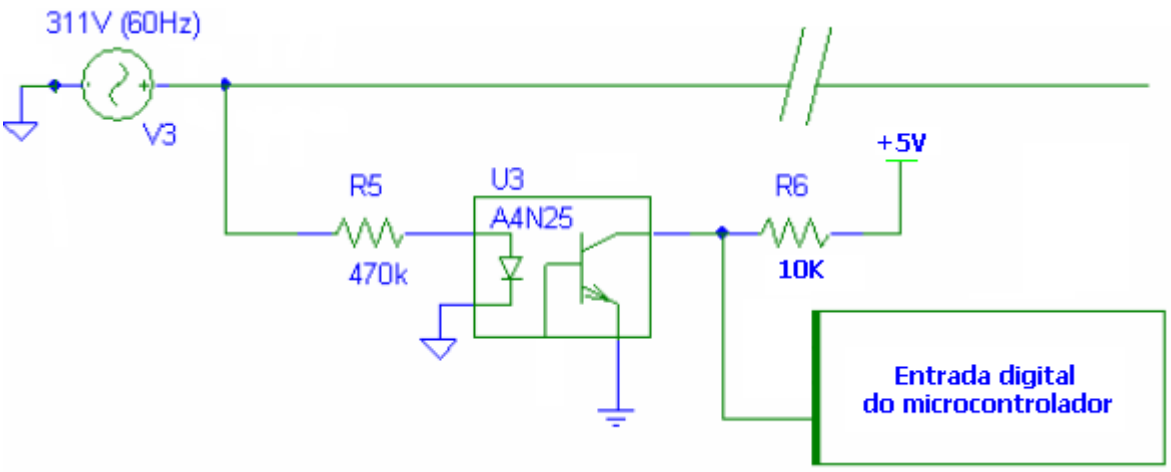


Esse dispositivo pode ser utilizado por um microcontrolador para identificar a presença de tensão 220VAC em um determinado ponto. A potência máxima dissipada por esse componente é de 250mW em 25 graus Celsius. Dessa forma, deve-se dimensionar um resistor em série com o foto-diodo interno para protegê-lo.

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 (6.1) é calculado o resistor que será utilizado no circuito, considerando a tensão de pico. Considere 311V como a tensão de pico.

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

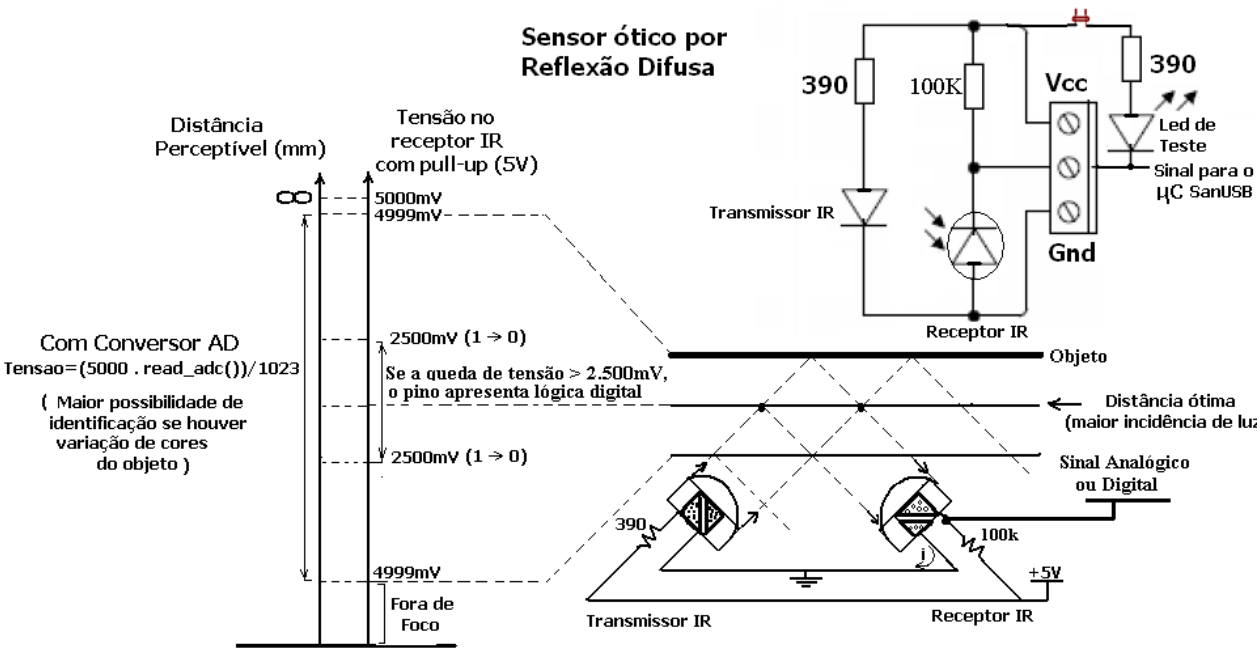
O resistor comercial mais próximo desse valor é 470KΩ.



Transmissor e Receptor IR

Os transmissores e receptores IR (*infra-red* ou Infravermellhos) são muito utilizados como sensor ótico por reflexão difusa para registro de posição. A figura abaixo 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). O vídeo

XXX mostra essa variação, com o acionamento de um led pela queda do sinal analógico atarvés da condução 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 ou 2.500mV (nível lógico baixo), isso é facilmente conseguido com um resistor de 100K em série com o receptor ótico IR.

Utilizando o princípio analógico, 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 e o valor do resistor em série com o receptor ótico IR foi de 2K2. 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. Mais detalhes em http://www.youtube.com/watch?v=18w0Oeaco4U. O programa abaixo mostra a leitura analógico em mV do sensor ótico via emulação serial.

```
#include <SanUSB.h> //Leitura de tensão em mV com variação de um potenciômetro
#include <usb_san_cdc.h>// Biblioteca para comunicação serial virtual

int32 tensao;

main() {
clock_int_4MHz();

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

output_high(pin_b7);
delay_ms(500);
output_low(pin_b7);
delay_ms(500);             }}
```

AUTOMAÇÃO E DOMÓTICA COM CONTROLE REMOTO UNIVERSAL

A comunicação entre a unidade remota e um eletrodoméstico, como uma TV, se dá geralmente por emissão de radiação infravermelha modulada por pulsos.

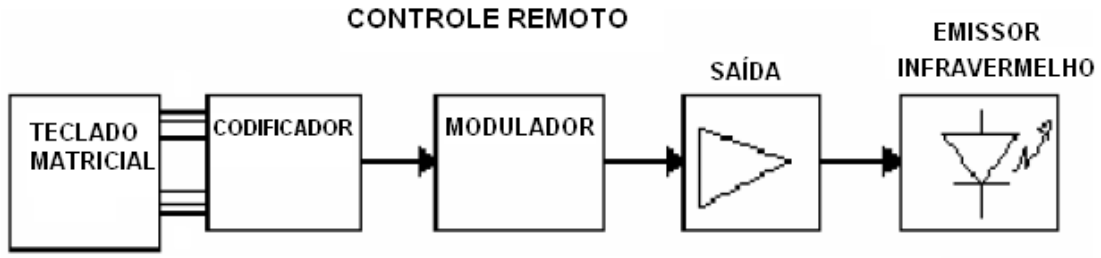


Diagrama em blocos da unidade transmissora remota

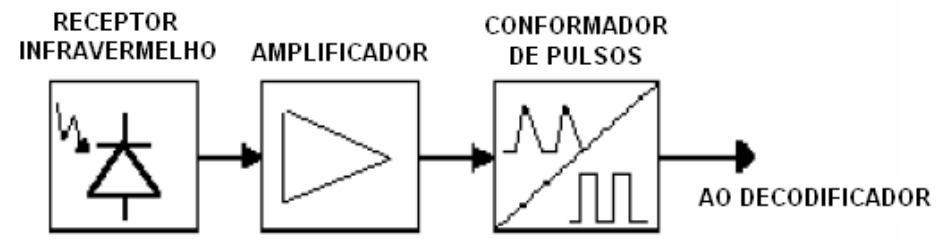


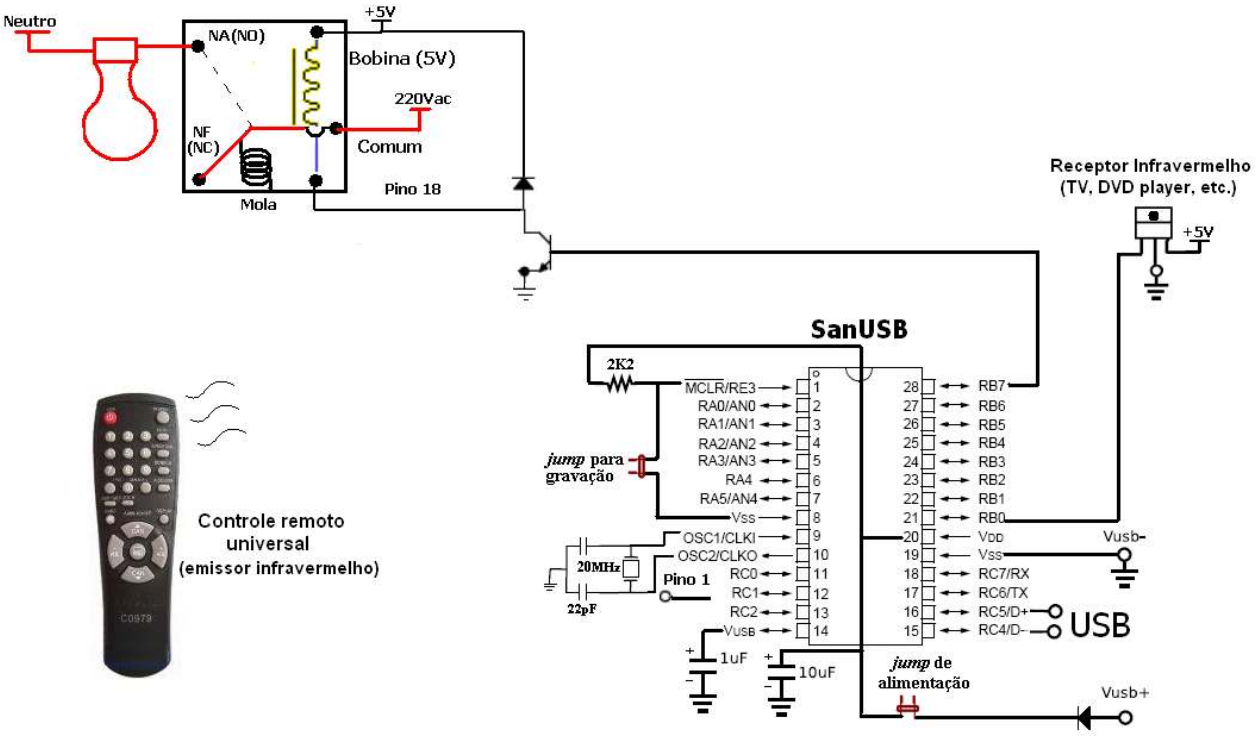
Diagrama em blocos da unidade receptora localizada no televisor

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 e etc.

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 freqüência 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 nível 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. Um exemplo de circuito para acionamento de cargas remotas com controle universal pode ser vista abaixo e em http://www.youtube.com/watch?v=1l6s9xtrJl0 .



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 <SanUSB.h>

short int rele;

#int_ext
void bot_ext()
{
rele=!rele;
output_bit(pin_b5,rele);
delay_ms(1000);  //Tempo para deixar o receptor cego por um segundo após a primeira atuação da interrupção
}

main() {
clock_int_4MHz();
enable_interrupts (global); // Possibilita todas interrupcoes
enable_interrupts (int_ext); // Habilita interrupcao externa 0 no pino B0 onde está ligado o receptor infravermelho

while (TRUE)
{
output_high(pin_B7);
delay_ms(500);
output_low(pin_B7);
delay_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 abaixo, para receber somente a luz IR direcional.

LCD (DISPLAY DE CRISTAL LÍQUIDO)

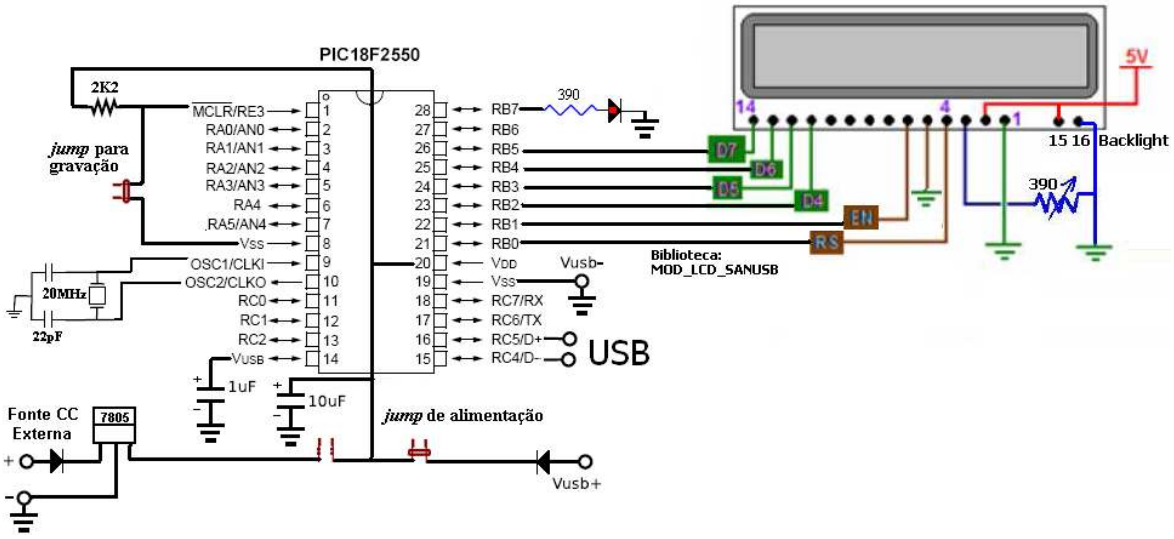
O LCD, ou seja, display de cristal líquido, é um dos periféricos mais utilizados como dispositivo de saída em sistemas eletrônicos. Ele contém um microprocessador de controle, uma RAM interna que mantém *escritos* no display (DDRAM) os dados enviados pelo microcontrolador e uma RAM de construção de caracteres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

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

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 este periférico que utiliza **somente o *nibble* superior do barramento de dados (D7, D6, D5 e D4)**, como é o caso da biblioteca mod_lcd_sanusb.c com a seguinte configuração:



A Tabela abaixo traz um resumo das instruções mais usadas na comunicação com os módulos LCD.

Tabela 15.4 - Instruções mais comuns

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 cursor ao entrar com caractere	Para a esquerda	0	0	**04**
	Para a direita	0	0	**06**
Deslocamento da mensagem ao entrar com caractere	Para a esquerda	0	0	**07**
	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**

Utilizando as instruções do LCD:

Para rolar o conteúdo do LCD um caractere para a direita, utilize o comando **lcd_envia_byte(0, instrução),** por exemplo, **lcd_envia_byte(0,0x1C**) e para rolar o conteúdo do LCD um caractere para a esquerda, utilize o comando **lcd_envia_byte(0,0x18).**

Exemplo de uso do recurso de rolagem do display:

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.

```
////////////////////////////Cabeçalho Padrão//////////////////////////////////
#include <SanUSB.h>
#include <MOD_LCD_SANUSB.c> // RB0-RS, RB1-EN, RB2-D4, RB3-D5, RB4-D6, RB5-D7
```

```
int16 temperatura;
int8 x;
int1 led;

main() {
clock_int_4MHz();
lcd_ini(); // Configuração inicial do LCD

setup_adc_ports(AN0);  //Habilita entradas analógicas - A0
setup_adc(ADC_CLOCK_INTERNAL);
delay_ms(100);
printf(lcd_escreve,"AUTOMACAO FERRAMENTA SanUSB ");

while (1) {
set_adc_channel(0);
delay_ms(10);
temperatura=500*read_adc()/1023;

lcd_pos_xy(1,2); // Posiciona segunda linha
printf(lcd_escreve,"TEMPERATURA ATUAL=%lu C",temperatura); //Com LM35

for (x = 0; x < 15; x ++) // repete o bloco abaixo por 15 vezes
{
lcd_envia_byte(0,0x18); // rola display um caractere para esquerda
lcd_envia_byte(0,0x0C); //Apaga o cursor

led=!led; output_bit(pin_b7,led);
delay_ms(500);
}
                                }}
```

Para ativar o cursor, utilize o comando **lcd_envia_byte(0,0x0E).** Para ativar o cursor piscante, utilize o comando **lcd_envia_byte(0,0x0F),** e para desativar o cursor, use **lcd_envia_byte(0,0x0C);**

Posicionando o cursor:

Para posicionar o cursor no LCD, podemos usar a função **lcd_pos_xy(x,y)**, onde x e y são, respectivamente, a coluna e a linha onde o cursor deve ser reposicionado.

Desta forma, caso deseje escrever algo na primeira linha do display, sem apagar a segunda linha, basta inserir o comando **lcd_pos_xy(1,1)**. Isso irá posicionar o cursor na primeira linha, e

primeira coluna. No entanto, tome cuidado, pois uma vez que o display não foi apagado, as informações antigas permanecerão na primeira linha, a menos que você as sobrescreva.

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.

Observe a seguinte tabela :

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)

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 abaixo.

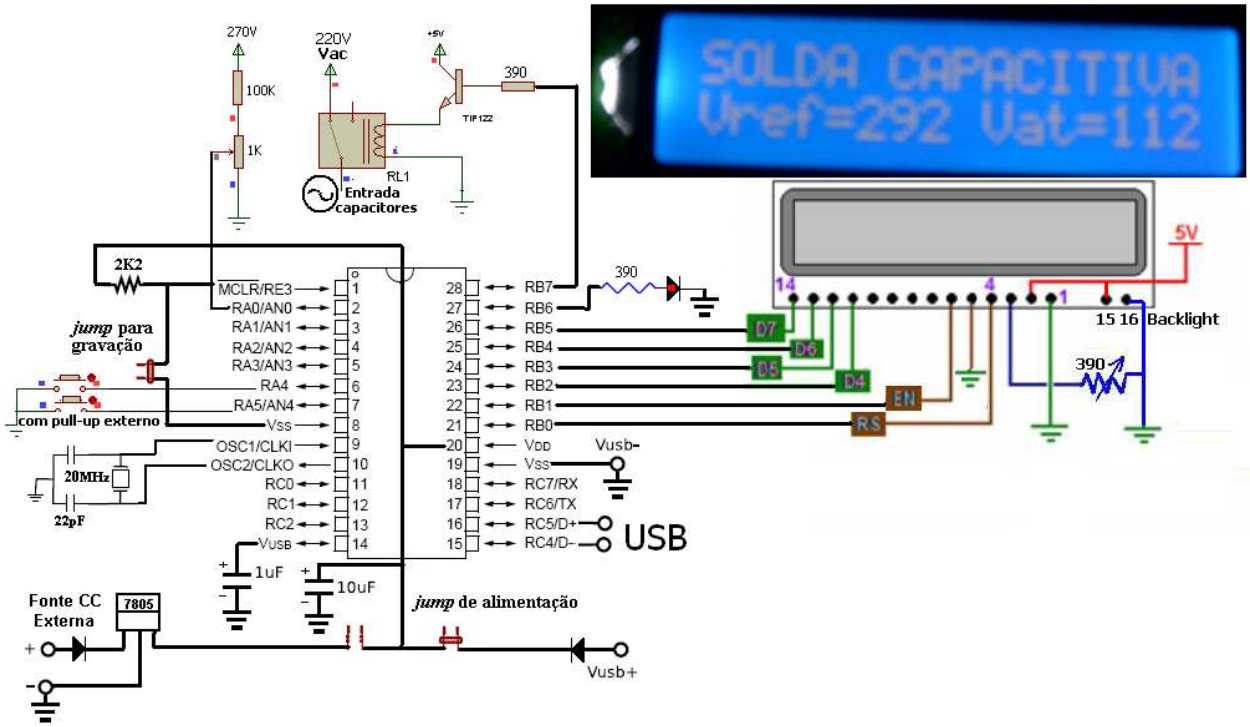


Figura 4: circuito do sistema de controle de tensão

```
////////////////////////////Cabeçalho Padrão//////////////////////////////////
#include <SanUSB.h>

#include <MOD_LCD_SANUSB.c> // RB0-RS, RB1-E, RB2-D4, RB3-D5, RB4-D6, RB5-D7

#define botaoinc pin_a4
#define botaodec pin_a5
#define rele pin_b7
#define ledrele pin_b6


unsigned int16 vref=270, guardavref, constante=100;
unsigned int32 vatual, valorAD;//Deve ser de 32 bits devido ao cálculo do AD que esoura 65536
unsigned int8 baixovref, altovref; // Como vref> 256 guardar o valor em 2 bytes, posições 10 e 11  da EEPROM interna
int1 flag1, flag2;

main() {
clock_int_4MHz();
lcd_ini(); // Configuração inicial do LCD
output_low(rele);
output_low(ledrele);

guardavref=(256*read_eeprom(10))+read_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)
delay_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) {output_high(rele); output_high(ledrele); } //Abre o relé, avisa com led
if (vatual<=20) {output_low(rele); output_low(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);


delay_ms(300);
                                        }}
```

LDR

LDR significa *Light Dependent 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.

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 tensão estabilizada, como mostra a figura 4.

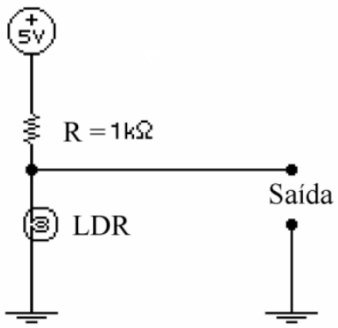


fig 4 - circuito sensor

Para obter este circuito e os valores de tensão na saída para as diferentes luminosidades, forma tidos por ANTONIETI, B. Em que as medições da tensão de saída foram feitas e colocadas em uma tabela juntamente com as iluminâncias medidas por um luxímetro comercial da marca MINIPA, modelo MLM-1010, de 3 ½ dígitos , com precisão de 4% da leitura + 0.5% do fundo de escala, na 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.

Tabela 2 - correspondência entre a tensão da saída e a iluminância

Lux	2	5	12	20	36	60	94	130	**180**	240	338	430	530	**674**	827	1000	1183	1404	1651	1923
Volt	4,9	4,9	4,8	4,7	4,5	4,3	4,1	4	**3,8**	3,6	3,3	3,1	3	**2,8**	2,7	2,5	2,4	2,3	2,1	2

Com base na tabela, foi construído o gráfico da figura abaixo.

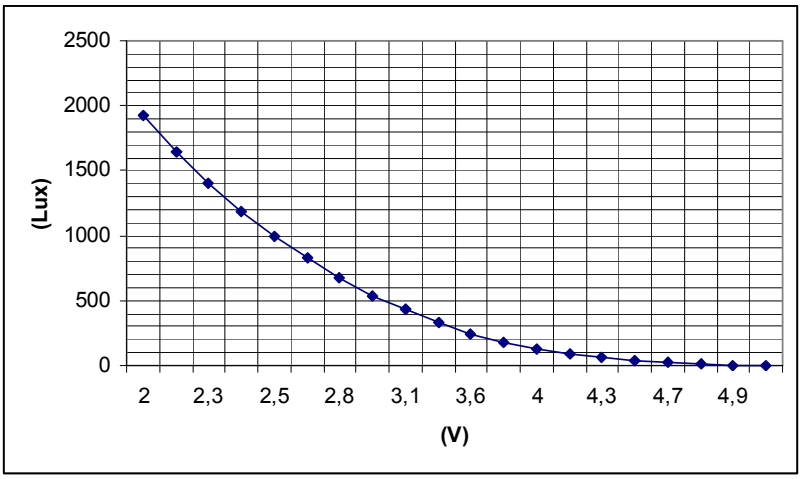


fig. 5 – gráfico lux X volt

Para simplificar o programa do PIC, foi modelado a curva do gráfico acima, dividindo-a em três retas como mostra a figura 6.

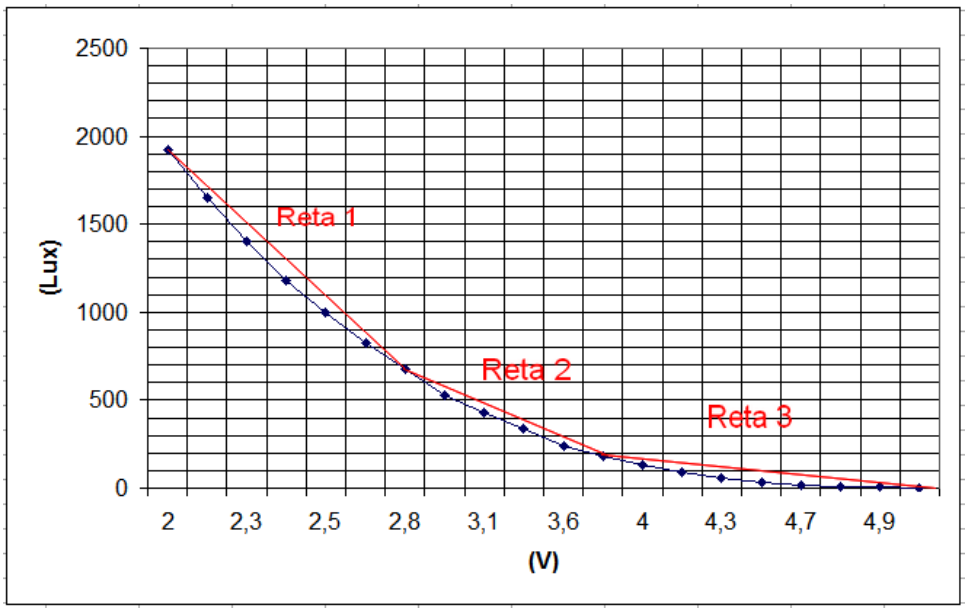


fig. 6 – divisão da curva em três retas

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 7 mostra o novo gráfico lux versus volts, utilizando as equações 03, 04 e 05.

Os cálculos da equação geral de cada reta são mostrados a seguir:

Reta 1: para $(V)<2,8$ e $(V)\geq 2,0$

$$\begin{vmatrix} x & y & 1 \\ 2,8 & 674 & 1 \\ 2 & 1923 & 1 \end{vmatrix} = 0 \rightarrow 2y + 674x + (2,8.1923) - 2,8y - (674.2) - 1923x = 0$$

$$2y - 2,8y + 674x - 1923x + (2,8.1923) - (674.2) = 0$$

$$-0,8y - 1249x + 4036,4 = 0 \rightarrow 0,8y = -1249x + 4036,4$$

$$y = \frac{(4036,4 - 1249x)}{0,8} \quad \text{[Eq. 03]}$$

Reta 2: para $(V)\geq 2,8$ e $(V)\leq 3,8$

$$\begin{vmatrix} x & y & 1 \\ 3,8 & 180 & 1 \\ 2,8 & 674 & 1 \end{vmatrix} = 0 \rightarrow 2,8y + 180x + (3,8.674) - 3,8y - (180.2,8) - 674x = 0$$

$$-y - 494x + 2057,2 = 0 \rightarrow \quad y = 2057,2 - 494x \quad \text{[Eq. 04]}$$

Reta 3: para $(V)>3,8$ e $(V)\leq 5,0$

$$\begin{vmatrix} x & y & 1 \\ 5 & 0 & 1 \\ 3,8 & 180 & 1 \end{vmatrix} = 0 \rightarrow 3,8y + (180.5) - 5y - 180x = 0 \rightarrow 3,8y - 5y - 180x + (180.5) = 0$$

$$1,2y + 180x - 900 = 0 \rightarrow 1,2y = 900 - 180x$$

$$y = \frac{900 - 180x}{1,2} \quad \text{[Eq. 05]}$$

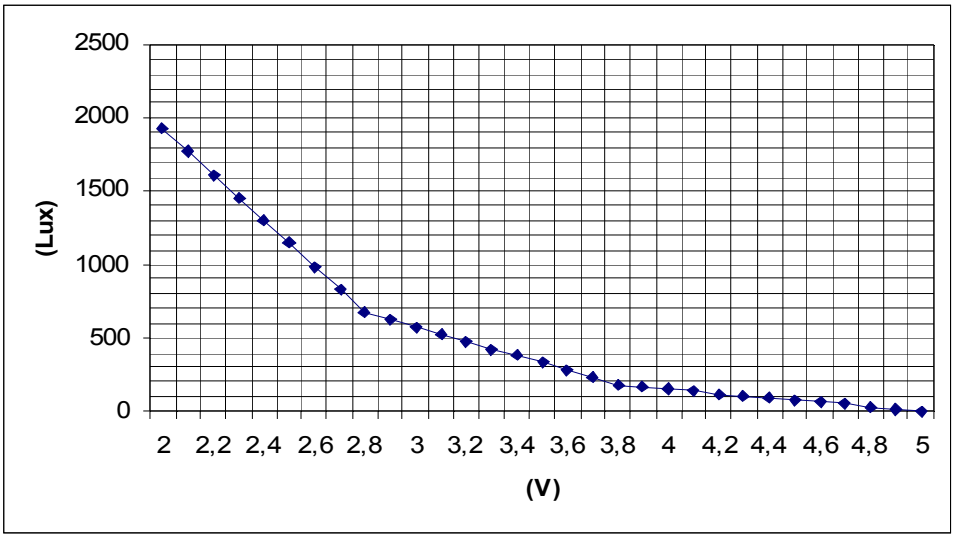


fig. 7 - gráfico lux *versus* tensão utilizando as equações 3, 4 e 5

SUPERVISÓRIO

Esta interface foi desenvolvida utilizando ambiente de programação Delphi® e através da emulação via USB de um canal serial COM virtual. A figura 8 mostra a tela do supervisório para iluminância e temperatura.

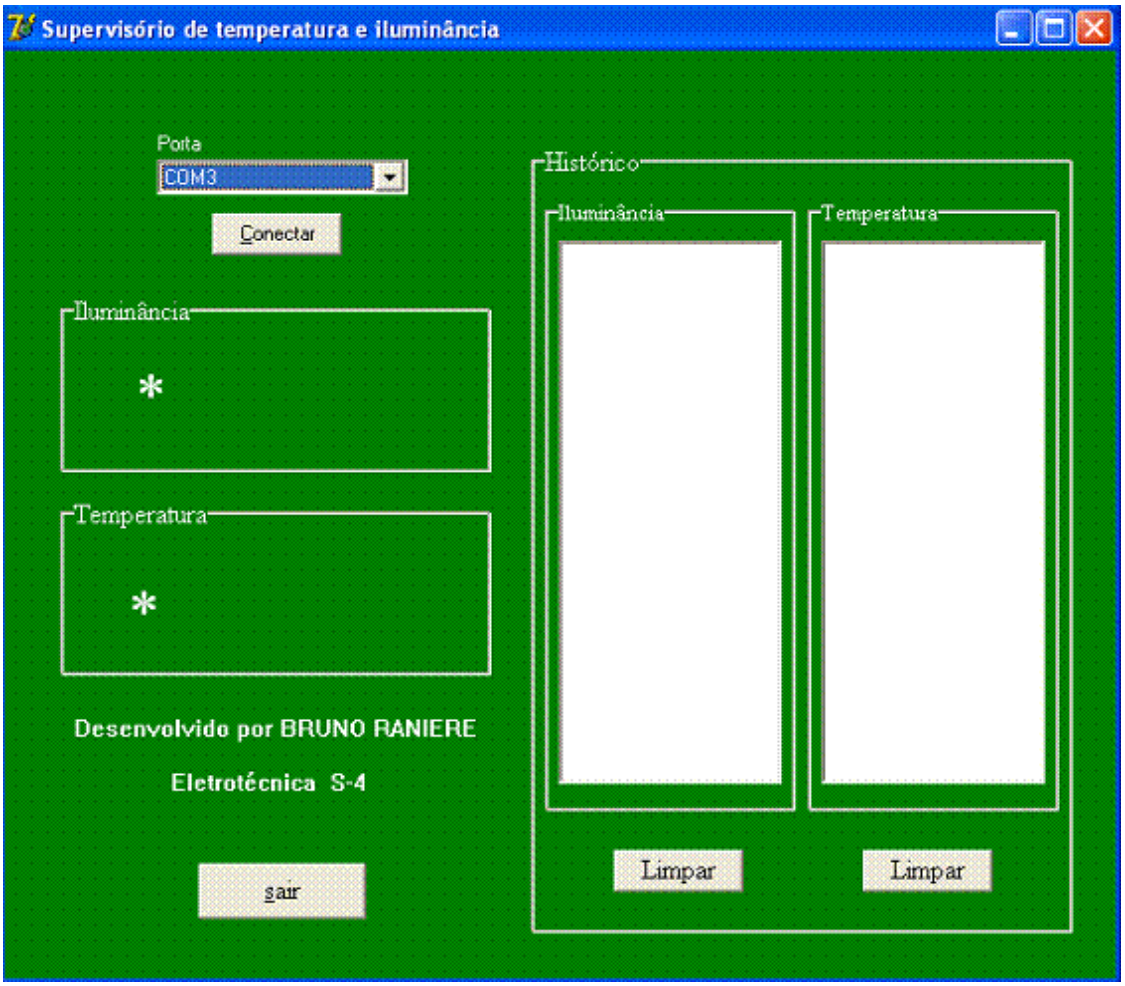


fig. 8 – Figura da tela do supervisório para Iluminância e Temperatura

Veja abaixo, na figura 9, o esquema circuito eletrônico montado e a na figura 10, 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.

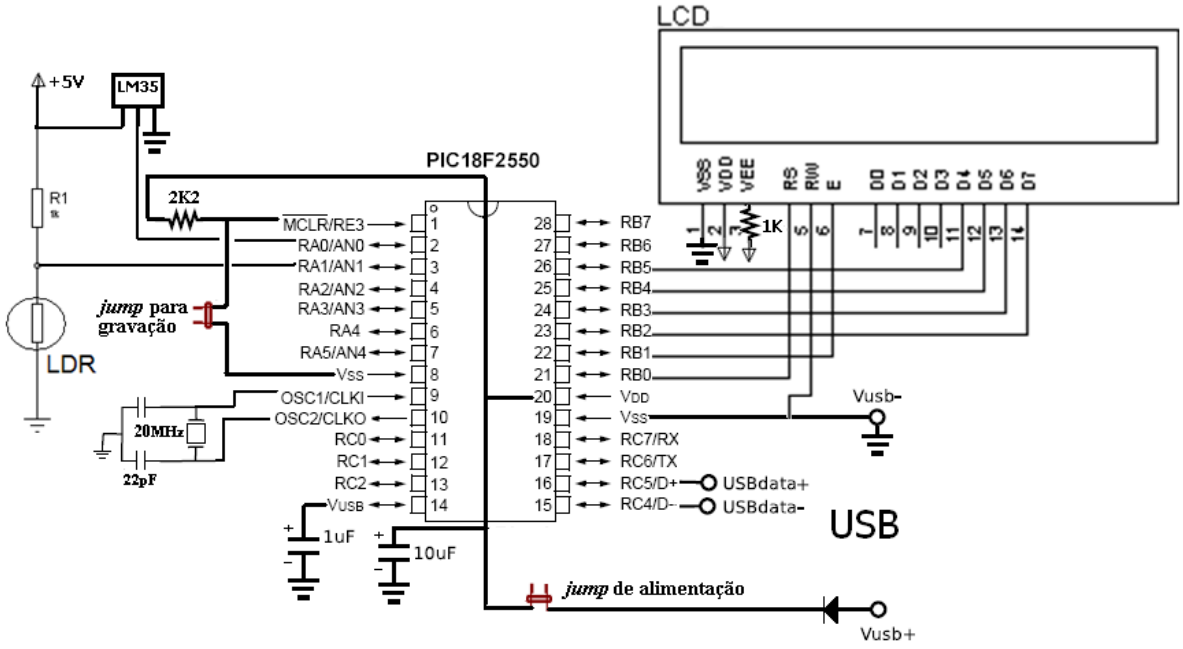


fig. 9

fig. 9 – – Esquema eletrônico do circuito montado

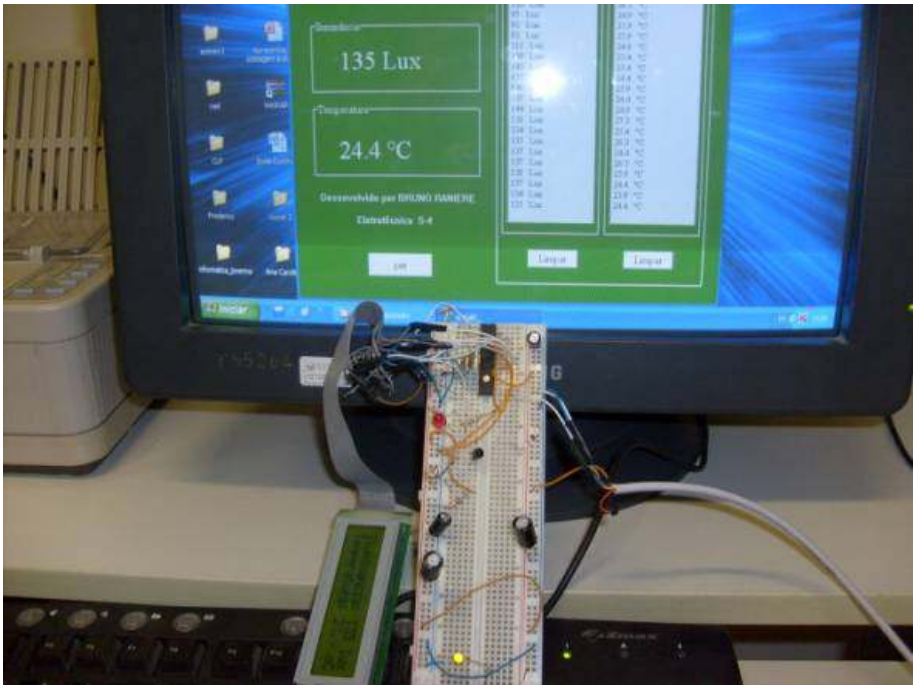


fig. 10 – Foto do circuito montado

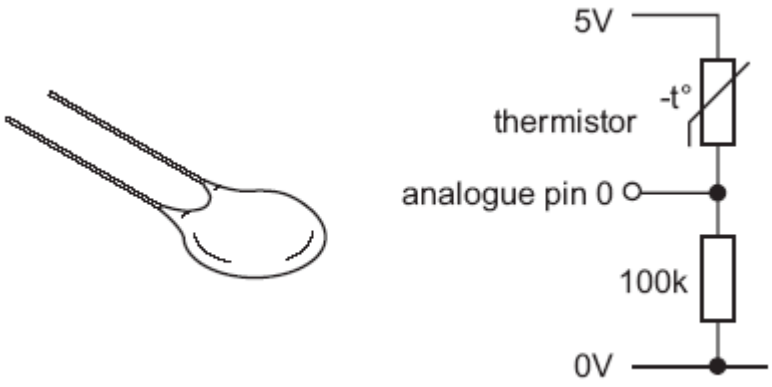
O luxímetro mostrado neste trabalho apresenta como uma solução de baixo custo para aplicações onde não é necessário haver uma grande precisão nas medições. O método do modelagem de curva pode ser aplicado em várias ocasiões onde não se sabe a equação que gerou o gráfico proposto. Isto ratifica a versatilidade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

```
// Programa Luxímetro digital + termômetro digital c/ comunicação via USB//

#include <SanUSB.h>
float tens,lux,temp;

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);
output_low(pin_b6);
printf (lcd_escreve," \f ");

while(1)
{
   set_adc_channel(1);
   delay_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 (usb_cdc_putc,"%.0f",lux);
      delay_ms(50);
      printf (usb_cdc_putc,"L");
      printf (lcd_escreve,"Iluminancia: %.0f lux     ",lux );
```

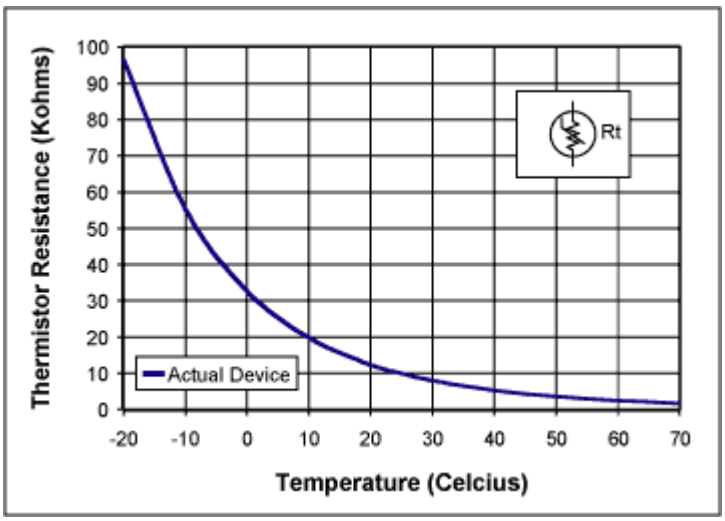
```
      lcd_envia_byte(0,0x0C); //Apaga o cursor    }

   if (tens<=2)        //Leitura não válida
   {
      lcd_pos_xy(1,1);
      printf (usb_cdc_putc,"Erro");
      delay_ms(50);
      printf (usb_cdc_putc,"L");
      printf (lcd_escreve,"valor fora da faixa!  ");
      lcd_envia_byte(0,0x0C); //Apaga o cursor
   }
       delay_ms(30);
      set_adc_channel(0);
      delay_ms(20);
      temp=500*(float)read_adc()/1023;
      lcd_pos_xy(1,2);
      printf (usb_cdc_putc,"%.1f",temp);
      delay_ms(50);
      printf (usb_cdc_putc,"T");
      printf (lcd_escreve,"Temperatura: %.1f oC  ",temp);
      lcd_envia_byte(0,0x0C); //Apaga o cursor
       delay_ms(800);
      output_high(pin_b6);
      delay_ms(200);
      output_low(pin_b6);         }
}
```

Termistor

Um termistor 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 do termistor ao microcontolador é muito simples, como mostra a figura baixo.



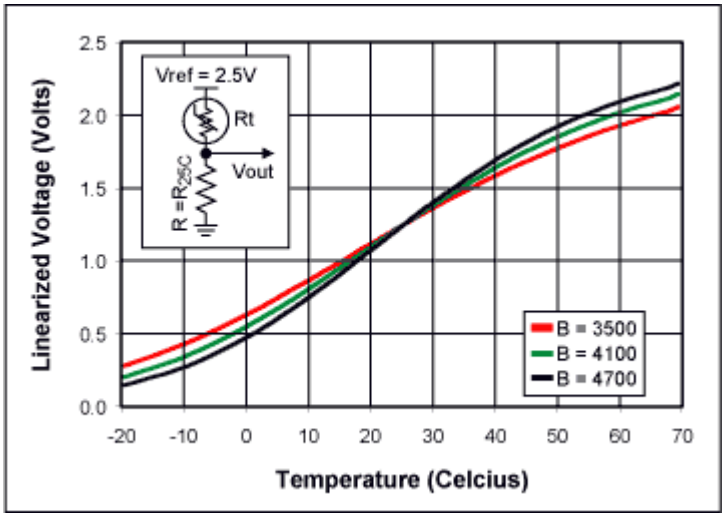
Convém lembrar que a resposta de um termistor não é linear, como mostra a figura abaixo.



LINEARIZAÇÃO

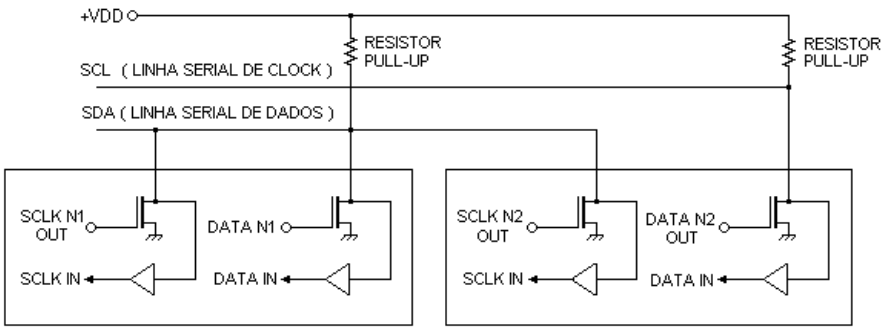
Um forma muito comum de linearização do termistor é por modalidade da tensão, onde um termistor NTC é conectado em série com um resistor normal formando um divisor de tensão. O circuito do divisor contém uma fonte de tensão de referência (Vref) igual a 2,5V. Isto tem o efeito de produzir uma tensão na 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 (25^0 C), então a região de tensão linear será simétrica em torno da temperatura ambiente (como visto em figura abaixo).



INTERFACE I^2C

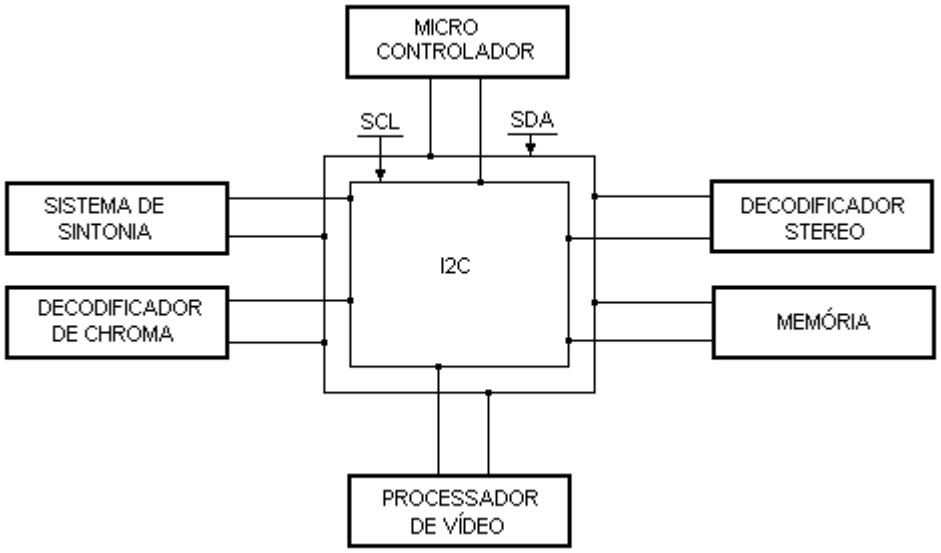
I2C significa Inter-IC (Integrated Circuit). Este barramento serial foi desenvolvido pela Philips como o objetivo de conectar CIs e periféricos de diferentes fabricantes em um mesmo circuito, como microcontroladores, memórias externas e relógio em tempo real, usando o menor número de pinos possível. 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 nível 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.

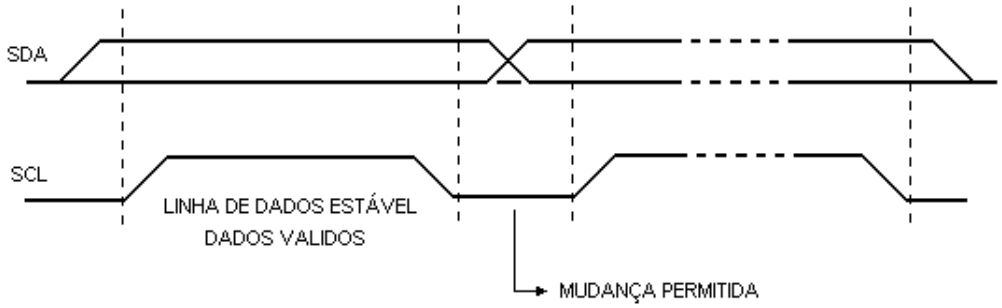
Esta interface apresenta a filosofia *multi-master* onde todo CI da rede pode transmitir ou receber um dado, e o transmissor gera o seu próprio clock de transmissão. O número máximo de CIs que podem ser conectados é limitado apenas pela capacitância máxima do barramento de 400pF.

Um exemplo típico de configuração I2C em TVs é mostrado na figura abaixo:



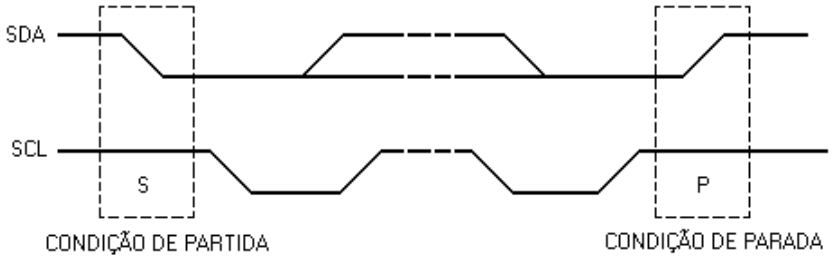
REGRAS PARA TRANSFERÊNCIA DE DADOS

Cada bit da linha de dados só é lido quando o nível da linha de clock está em nível alto.

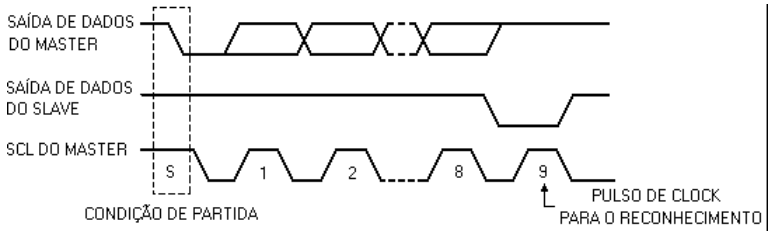


As condições de partida e parada de transmissão são sempre geradas pelo MASTER. O barramento é considerado como ocupado após a condição de partida, e livre um certo período de tempo após a condição de parada.

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**.



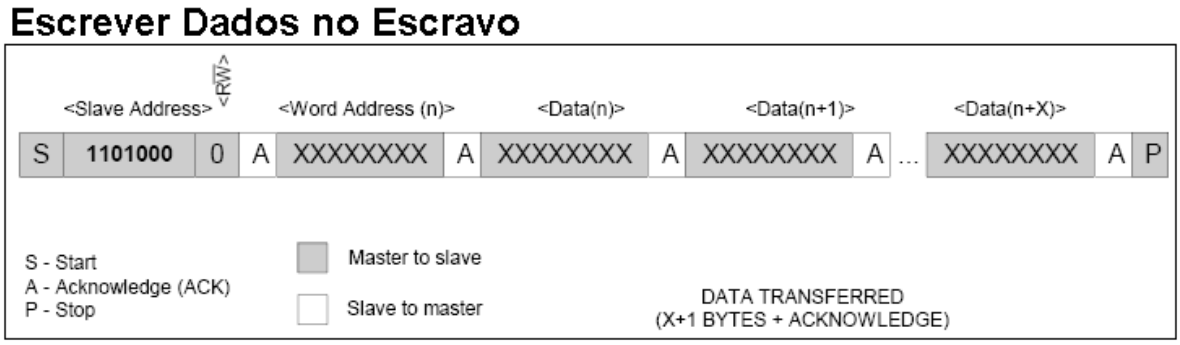
Cada byte é acompanhado de um bit de reconhecimento obrigatório. O reconhecimento é gerado pelo MASTER no ***décimo bit*** liberando a linha SDA (nível alto) durante a ocorrência do pulso de clock de reconhecimento. Por sua vez, o CI receptor (SLAVE) é obrigado a levar a linha SDA a nível baixo durante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



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 nível 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 nível L, forçando o MASTER a entrar em um modo de espera.

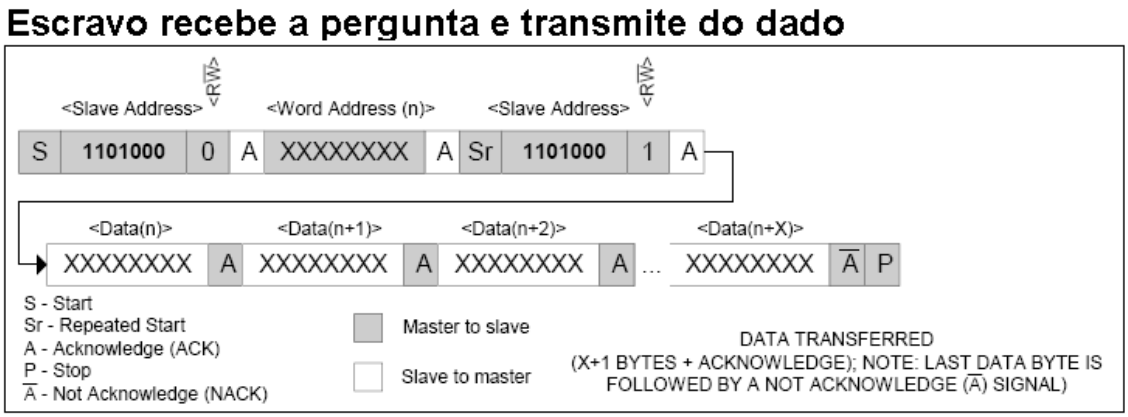
Para escrever um dado nos escravos é necessário enviar um byte de endereço do escravo, onde os **4 bits mais significativos** identificam o tipo de escravo (por exemplo, memórias EEPROM é **1010 ou 0xa0** e RTC é **1101 ou 0xd0** (com exceção do RTC PCF8583 cujo endereço também é **0xa0**). Os **3 bits 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();
  delay_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*).



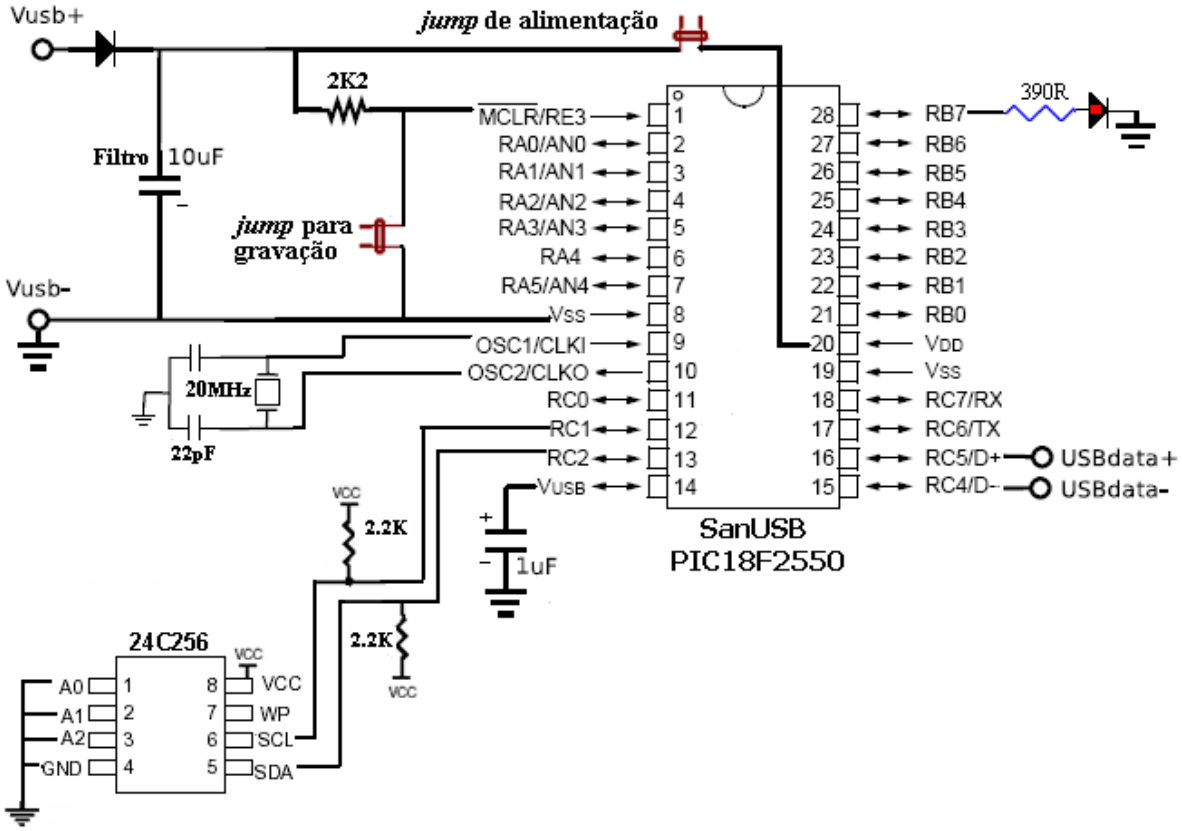
A Função da biblioteca I2C que descreve este protocolo de operação de leitura de memória EEPROM é a seguinte:

```
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;
```

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 abaixo mostra o circuito simples de uma EEPROM I2C ligada nn ferramenta SanUSB.



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 emulação serial somente quando a tecla ‘L’ é pressionada.

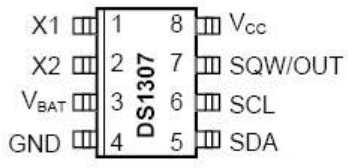
```
#include <SanUSB.h>
#include <usb_san_cdc.h>// Biblioteca para emulação da comunicação serial
#include <i2c_sanusb.c>
```

```
unsigned int16  i,j,endereco=0, posicao=0, valorgravado;
int32 tensao_lida32; //Varia de 0 a 5000mV (16bits), mas é int32 porque o cãlculo ultrapassa 65536
int8 byte1,byte2,byte3,byte4; // 4 Partes do valor int32 tensao_lida32
char comando;

int conv_dec_4bytes(int32 valor32) //Converte decimal de 32 bits em 4 bytes
    {
      int32 resultado1=0,resultado2=0;
      resultado1 = valor32/256; byte1 = valor32%256; //o que for resto (%) é menos significativo
      resultado2 = resultado1/256; byte2= resultado1%256;
      byte3= resultado2%256;  byte4 = resultado2/256;
      return(byte4,byte3,byte2,byte1);
    }

main() {
clock_int_4MHz();

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 entradas analógicas - A0
setup_adc(ADC_CLOCK_INTERNAL);

while(1){

set_adc_channel(0);
delay_ms(10);   //Tensão é 32bis porque o produto 5000* read_adc() de 10 bits (1023) pode ser maior que 65536
tensao_lida32=(5000*(int32)read_adc())/1023; //Calcula a tensão Trimpot em mV:   5000mV      -  1023 (10bis)
printf (usb_cdc_putc,"\r\n tensao do Trimpot = %lu mV\r\n",tensao_lida32);//             tensao_lida -  read_adc();

conv_dec_4bytes(tensao_lida32);
//printf (usb_cdc_putc,"\r\nVariavel em Hexadecimal = %x %x %x %x\r\n",byte4,byte3,byte2,byte1);//Debug

posicao=2*endereco; //Endereço é o ponteiro de cada valor de 2 bytes (16 bits)
escreve_eeprom( 0, posicao, byte2); //segundo Byte menos significativo do int32
escreve_eeprom( 0, posicao+1, byte1 ); //Byte menos significativo do int32
//printf (usb_cdc_putc,"\r\nEndereco = %lu   Posicao = %lu\r\n",endereco,posicao);//Debug

++endereco;    if (endereco>=150){endereco=0;} //Buffer de 300 bytes posicao<=300
//****************LEITURA DO BUFFER DA EEPROM EXTERNA**************************
if (usb_cdc_kbhit(1)) //Se existe carater digitado entre na função
      {comando=usb_cdc_getc();
       if (comando=='L'){
printf(usb_cdc_putc, "\r\n\nEEPROM:\r\n"); // Display contém os primeiros 64 bytes em hex da eeprom externa i2c
  for(i=0; i<10; ++i) {   //10linhas * 30colunas = 300 bytes (int16 i,j)
         for(j=0; j<15; ++j) {

valorgravado= 256*le_eeprom(0,(i*30)+2*j) + le_eeprom(0,(i*30)+2*j+1); //150 Valores de 16 bits ou 300 de 8 bits.
            printf(usb_cdc_putc, "%lu ", valorgravado );
                        }
            printf(usb_cdc_putc, "\n\r");
                         }}}
//*****************************************************************************
output_high(pin_b7);
delay_ms(1000);
output_low(pin_b7);
delay_ms(1000);             }}
```

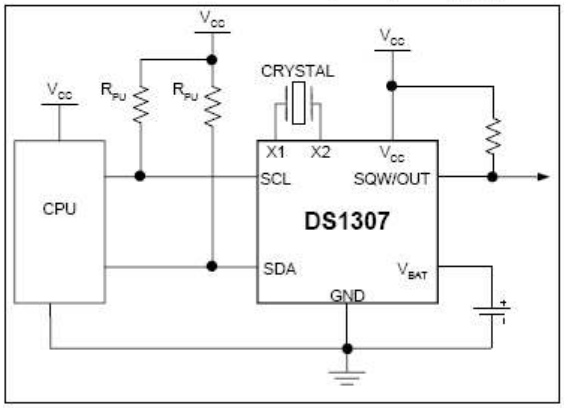
```
\0A tensao do Trimpot = 1158 mV\0D
\0A\0D
   \0A
\0A   EEPROM:\0D
   2662 2135 2150 2135 2160 2170 1148 1148 1129 1148 1158 1158 2145 2145 2130 \0A
2155 2150 2140 2111 2126 2130 2145 2130 2126 2130 2145 2135 2130 2145 2145 \0A   \0D
2135 2135 2145 2150 2140 2116 2130 2140 2150 2140 2130 2150 2140 2126 2160 \0A\0D
2145 2145 2150 2140 2135 2145 2135 2150 2135 2145 2150 2145 2145 2135 2145 \0A\0D
2145 2135 2160 2145 2145 2150 2140 2150 2140 2140 2140 2155 2140 2140 2140 \0A\0D
2140 2160 2145 2145 2155 2130 2170 2150 2135 2155 2126 2145 2096 2106 2145 \0A\0D
2091 2140 2155 2145 2145 2145 2145 2135 2150 2130 2155 2145 2145 2170 2135 \0A\0D
2140 2130 2140 2160 2145 2091 3831 3822 3826 3836 3826 3831 3822 3831 3822 \0A\0D
3826 3817 3040 3025 3015 3020 3030 2991 3025 3015 3020 3015 2996 3040 2991 \0A\0D
3005 3059 3015 2996 3020 3015 2634 2649 2614 2605 2683 2639 2653 2639 2619 \0A\0D
```

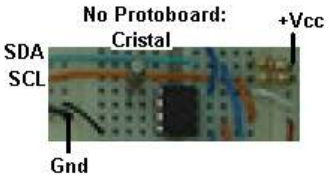
RTC (RELÓGIO EM TEMPO REAL)

O *Real Time Clock I2C* DS1307 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 I^2C (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.

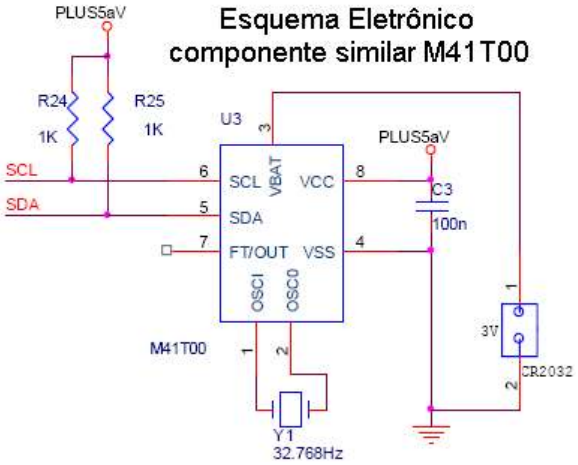


Circuito comum de operação RTC





Esquema Eletrônico
componente similar M41T00



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 PIC16F877A. Testes indicaram que a bateria suportou o processamento e incremento automático do relógio por cerca de sete horas sem alimentação externa.

REGISTROS RETENTORES DE TEMPO DO DS1307

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00H	CH	10 Seconds			Seconds (0-9)				Seconds	00–59
01H	0	10 Minutes			Minutes (0-9)				Minutes	00–59
02H	0	12	10 Hour (0-2)	10 Hour	Hours (0-9)				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03H	0	0	0	0	0	DAY			Day	01–07
04H	0	0	10 Date		Date (1-9)				Date	01–31
05H	0	0	0	10 Month	Month (1-9)				Month	01–12
06H	10 Year				Year (0-9)				Year	00–99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08H-3FH									RAM 56 x 8	00H–FFH

//PROGRAMA PARA CONFIGURAR E LER UM RELÓGIO RTC I2C VIA MONITOR SERIAL//////////////////////

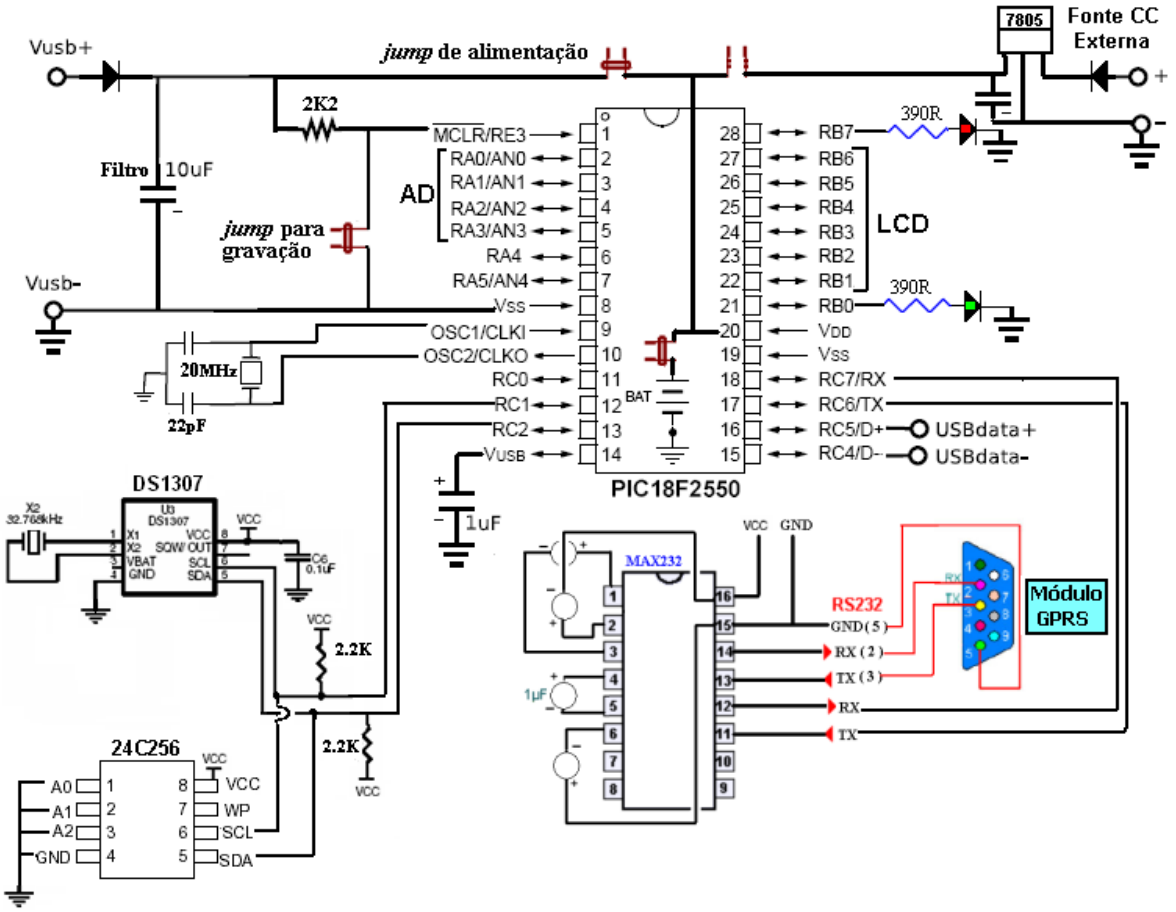
```
//PROGRAMA PARA CONFIGURAR E LER UM RELÓGIO RTC I2C VIA MONITOR SERIAL//////////////////////
#include <SanUSB.h>
#include <usb_san_cdc.h>// Biblioteca para emulação da comunicação serial
#include <i2c_sanusb.c>

Char entrada,funcao,endrtc,valorrtc1,valorrtc2;
unsigned int endereco, valor,valorbcd, numquant;
boolean led;
/*****************************************************************************************************
* 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
}
////////////////////////////////////////////////////////////////////////////////
main() {
clock_int_4MHz();
usb_cdc_init(); // Inicializa o protocolo CDC
usb_init(); // Inicializa o protocolo USB
usb_task(); // Une o periférico com a usb do PC

while (1) {

//*****************************************************************************
if (usb_cdc_kbhit(1)) { //verifica se acabou de chegar um novo dado no buffer de recepção, depois o kbhit é zerado

entrada=usb_cdc_getc(); //comando é o Byte recebido pela serial usb_cdc_putc,
if (entrada=='A')
{ funcao=usb_cdc_getc();
switch (funcao) //UTILIZAR VALORES DECIMAIS EM DOIS DIGITOS. ex:06 ou 23 ou 15
{
////////////////////////FUNCAO 4: CONFIGURA RELÓGIO////////////////////////////////Ex: A4H09
case '4':
{
endrtc=usb_cdc_getc();
valorrtc1=usb_cdc_getc();
valorrtc2=usb_cdc_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(usb_cdc_putc,"\r\nA5 %2x:%2x:%2x",le_rtc(2), le_rtc(1),le_rtc(0)); //BCD em hexadecimal representa o decimal
printf(usb_cdc_putc," %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));//BCD em hexadecimal representa o decimal
printf(usb_cdc_putc," %2x%2x%2x\r\n",le_rtc(4), le_rtc(5), le_rtc(6));
break;
}}}
//*****************************************************************************
led = !led; // inverte o led de teste
output_bit (pin_b7,led);
delay_ms(500);
}
}
```

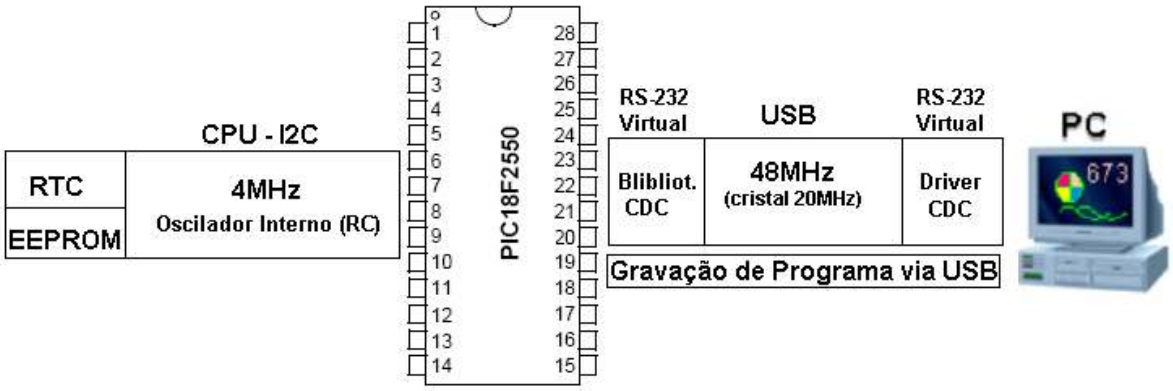
PROTÓTIPO DATALOGGER USB DE BAIXO CUSTO

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.

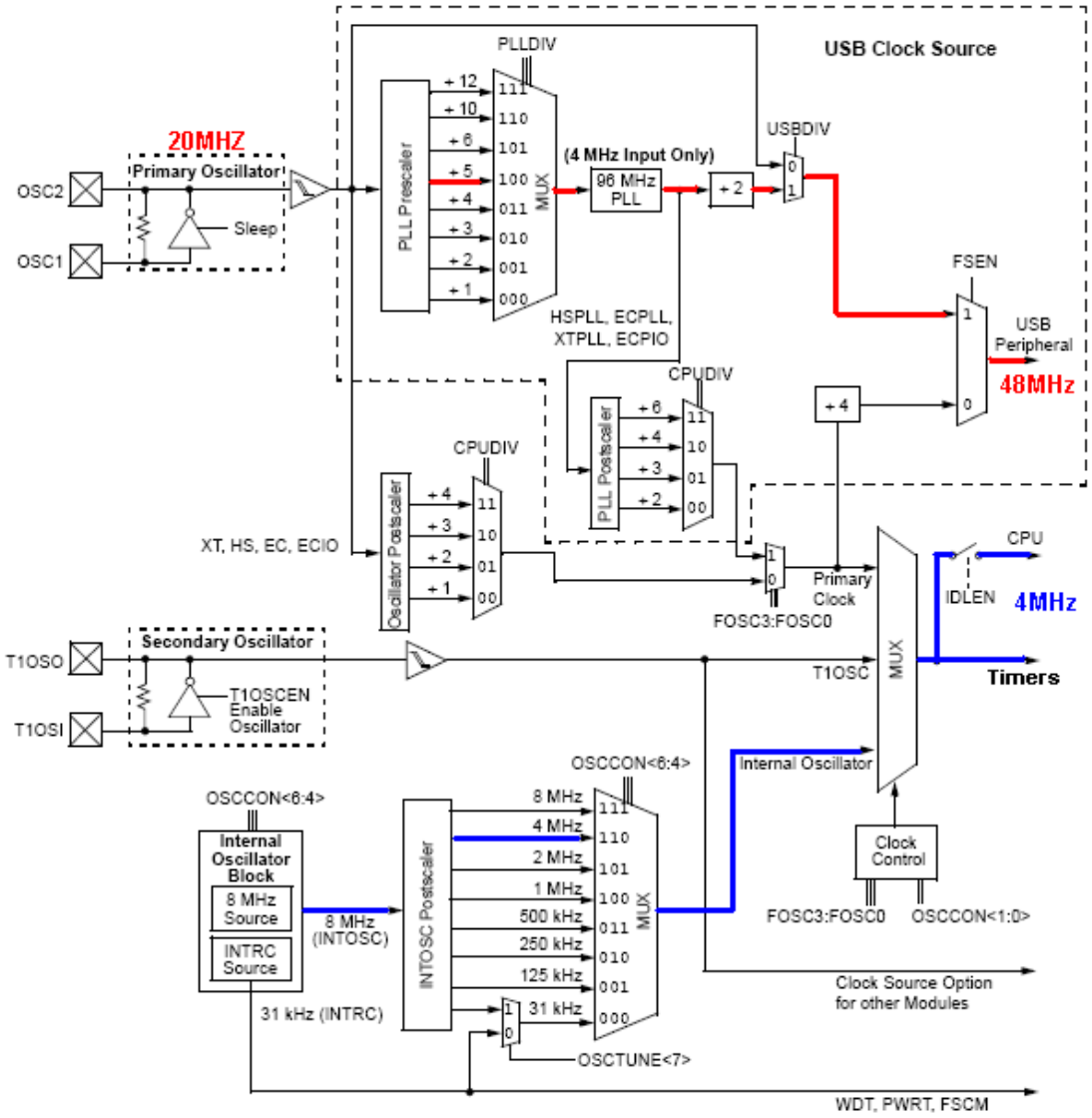
O conversor TTL/EIA-232 Max232 é utilizado para conexão do módulo GPRS ao sistema, cujos comandos AT são descritos no próximo tópico.



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;).



A configuração dos fusíveis no sistema de clock seguem o caminho em vermelho para a USB e azul para o processador e para os t*imers*.



```
////////////////////////////////////////////////////////////////////
//// 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 <SanUSB.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,posquant1,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(hora);
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||minutodec==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(usb_cdc_putc,"\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


////////////////////////////////////////////////////////////////////////////
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
usb_cdc_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);
         delay_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(usb_cdc_putc,"\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
```

```
delay_ms(500);
//*****************************************************************************

if (usb_cdc_kbhit(1)) {
 //verifica se acabou de chegar um novo dado no buffer de recepção, depois o kbhit é zerado para próximo dado

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

    switch (funcao) //UTILIZAR VALORES DECIMAIS EM DOIS DIGITOS. ex:06 ou 23 ou 15
     {

//*****************************************************************************
case '4':
       {
         endrtc=usb_cdc_getc();
         valorrtc1=usb_cdc_getc();
         valorrtc2=usb_cdc_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(usb_cdc_putc,"\r\nVALOR ESCRITO = %2x\r\n",valorbcd);
           //printf(usb_cdc_putc,"\r\nPOSICAO = %2x\r\n",endereco);

           hora=le_rtc(2);minuto=le_rtc(1);segundo=le_rtc(0);

           printf(usb_cdc_putc,"\r\nA4%2x:%2x:%2x",hora, minuto,segundo);
           printf(usb_cdc_putc," %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=usb_cdc_getc();
        posmeme2=usb_cdc_getc();
        sensor=usb_cdc_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);
        delay_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)) );}
            delay_ms(15);
          }
          printf(usb_cdc_putc,"\r\n"); //posiciona próxima linha
              }
      break;

   }}}
   }
 }
```

TRASNMISSÃO 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.

COMANDOS AT PARA ENVIAR MENSAGENS SMS DE UM COMPUTADOR PARA UM CELULAR OU MODEM GSM

A seguinte tabela lista os commandos AT para escrever e enivair mensagens SMS:

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

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.

COMANDOS AT PARA RECEBER MENSAGENS SMS EM UM COMPUTADOR ENVIADAS POR UM CELULAR OU MODEM GSM

A seguinte tabela lista os commandos AT para receber e enivair mensagens SMS:

Comando AT	Significado
+CNMI	New message indications
+CMGL	Lista mensagens
+CMGR	Lê menssagens
+CNMA	Reconhecimento de nova menssagem

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**

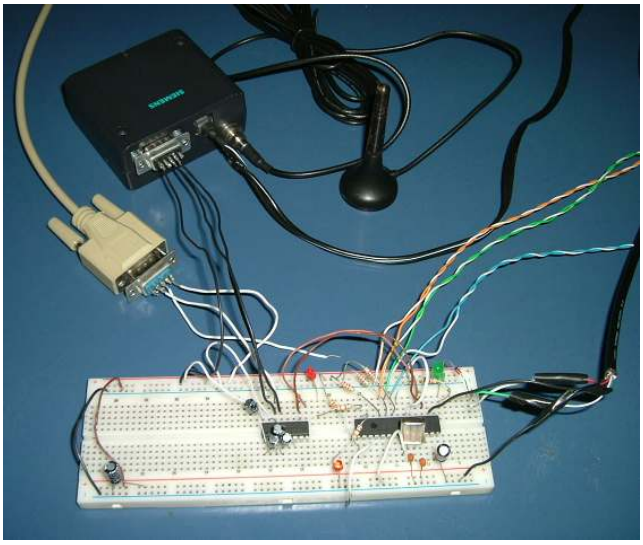
4- OK

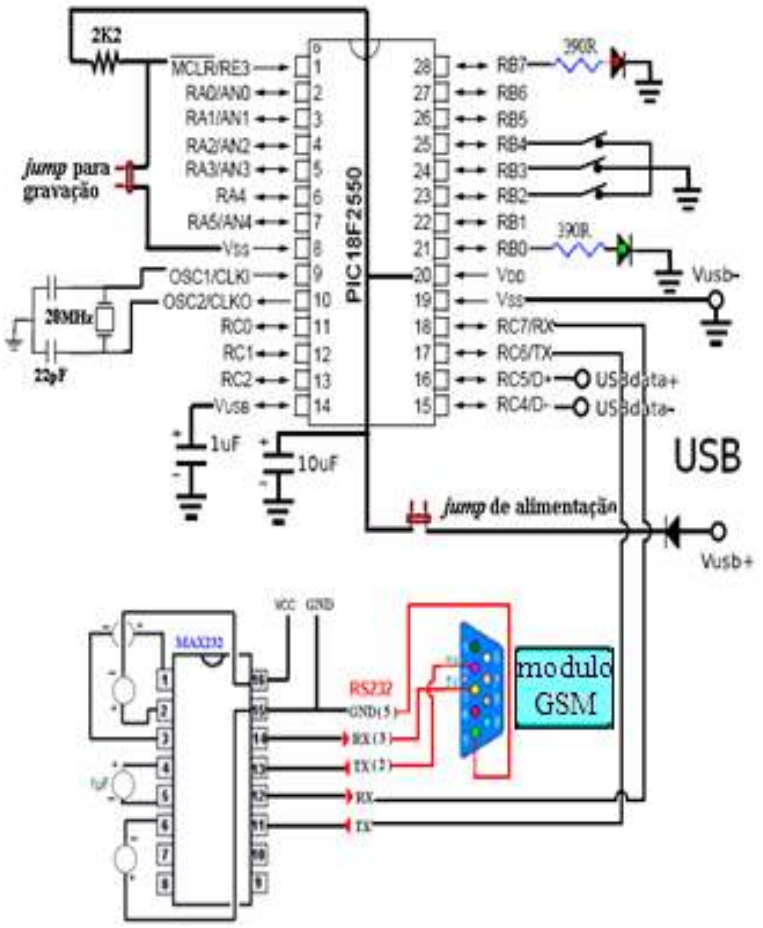
5-**AT+CMGS="+558588888888"**

6-**>Intrusão**

7- OK

As figuras abaixo apresentam 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*).






```
#include <SanUSB.h>
short int  ledpisca;

#USE RS232 (BAUD=9600,XMIT=PIN_C6,RCV=PIN_C7,stream=PC)

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
delay_ms(2000);

printf("AT+CMGS=\"+558588888888\"\r"); //5- Envia para o numero de destino

ledpisca=!ledpisca;
output_bit(pin_b7,ledpisca);
delay_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);
delay_ms(2000);
```

```
putc(0x0D);

while(TRUE){      output_high(pin_B7);
                  delay_ms(500);
                  output_low(pin_B7);
                  delay_ms(500);
}}
```

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.

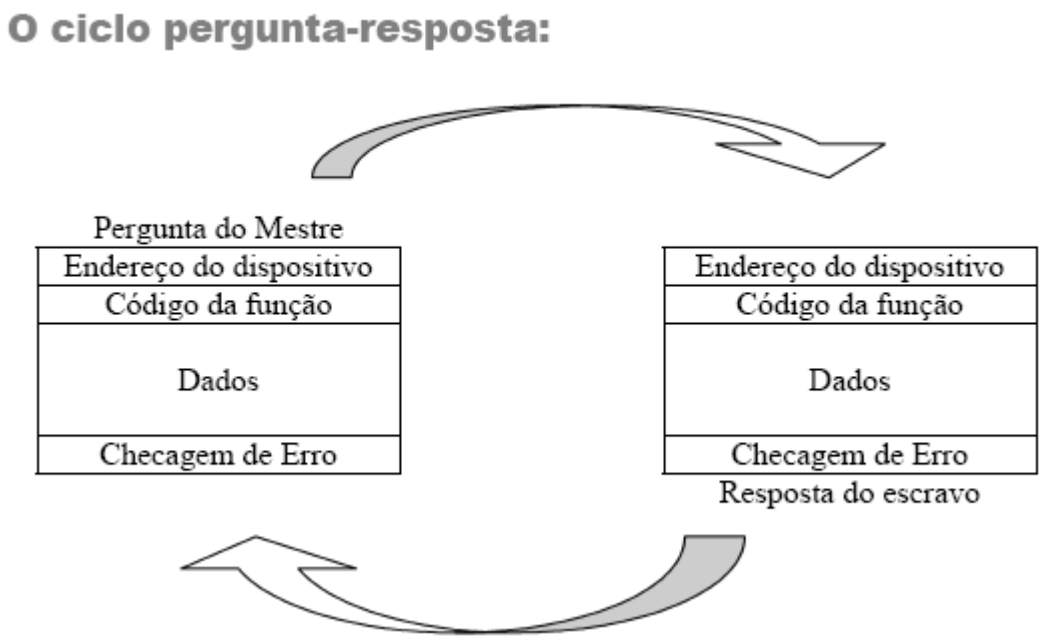
Este é talvez o protocolo de mais utilizado em automação industrial, pela sua simplicidade e facilidade de implementação.

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.

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.



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.

Modos de Transmissão

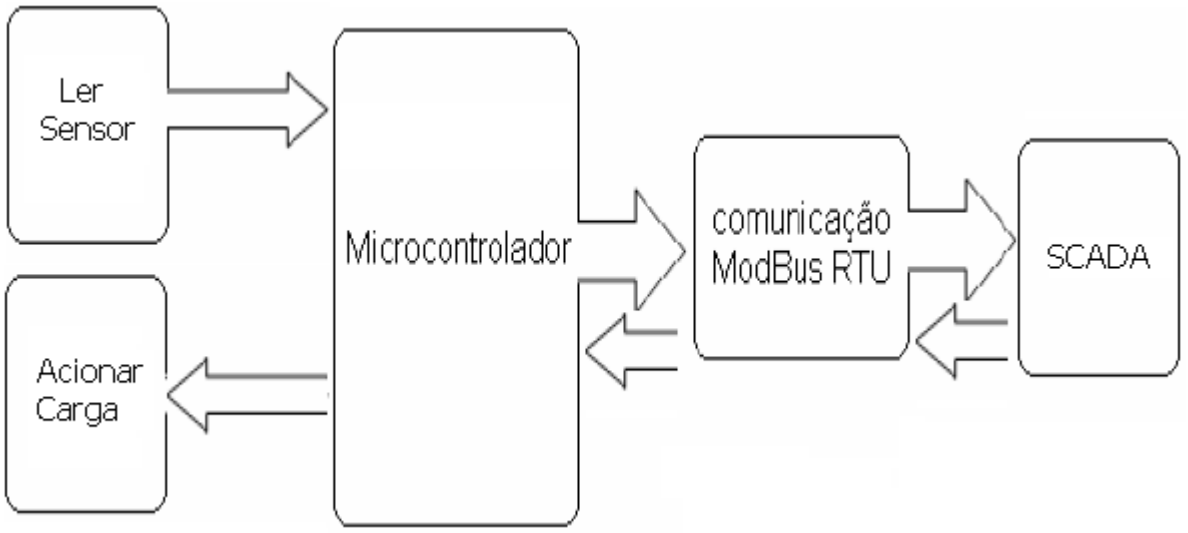
Existem dois modos de transmissão: ASCII (American Code for Information Interchange) e RTU (Remote Terminal Unit), que são selecionados durante a configuração dos parâmetros de comunicação.

Como a comunicação geralmente utilizada em automação industrial é em modo RTU, o projeto proposto foi desenvolvido nesta forma de comunicação.

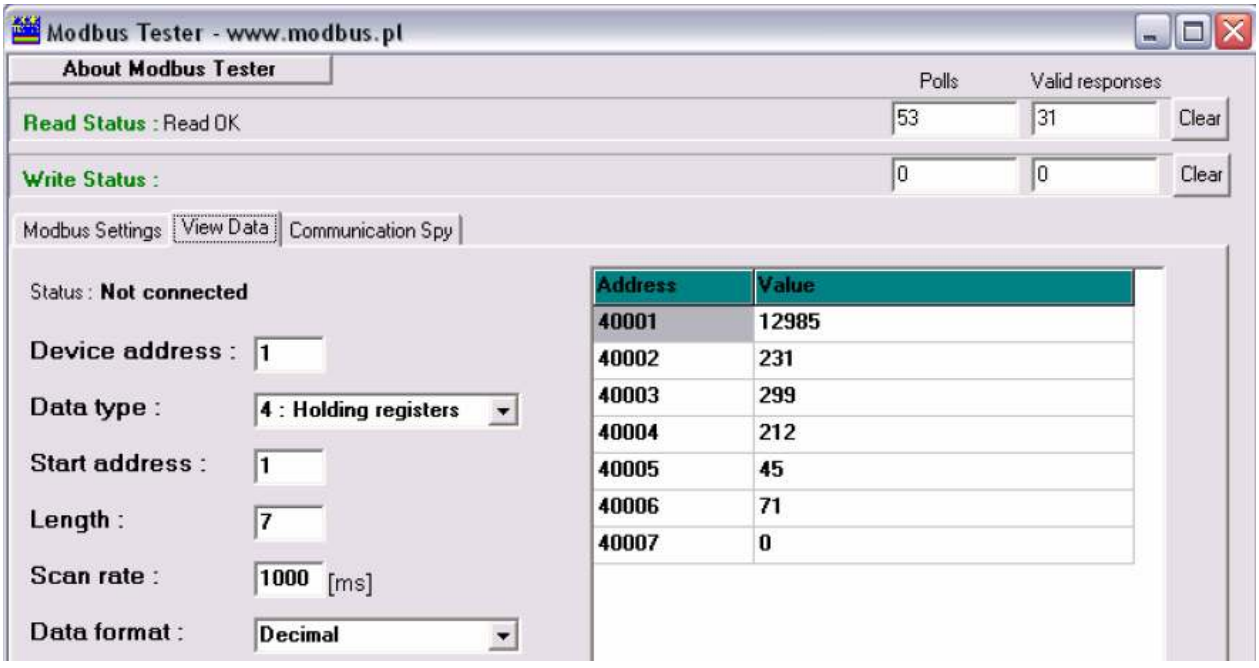
Modo RTU

Start	Endereço	Função	Dados	CRC	END
Silêncio 3..5 chars	← 8 bits→	← 8 bits→	←N x 8 bits→	←16 bits→	Silêncio 3..5 chars

A implementação prática de um projeto com o modbus embarcado é mostrada no diagrama de blocos abaixo.



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.

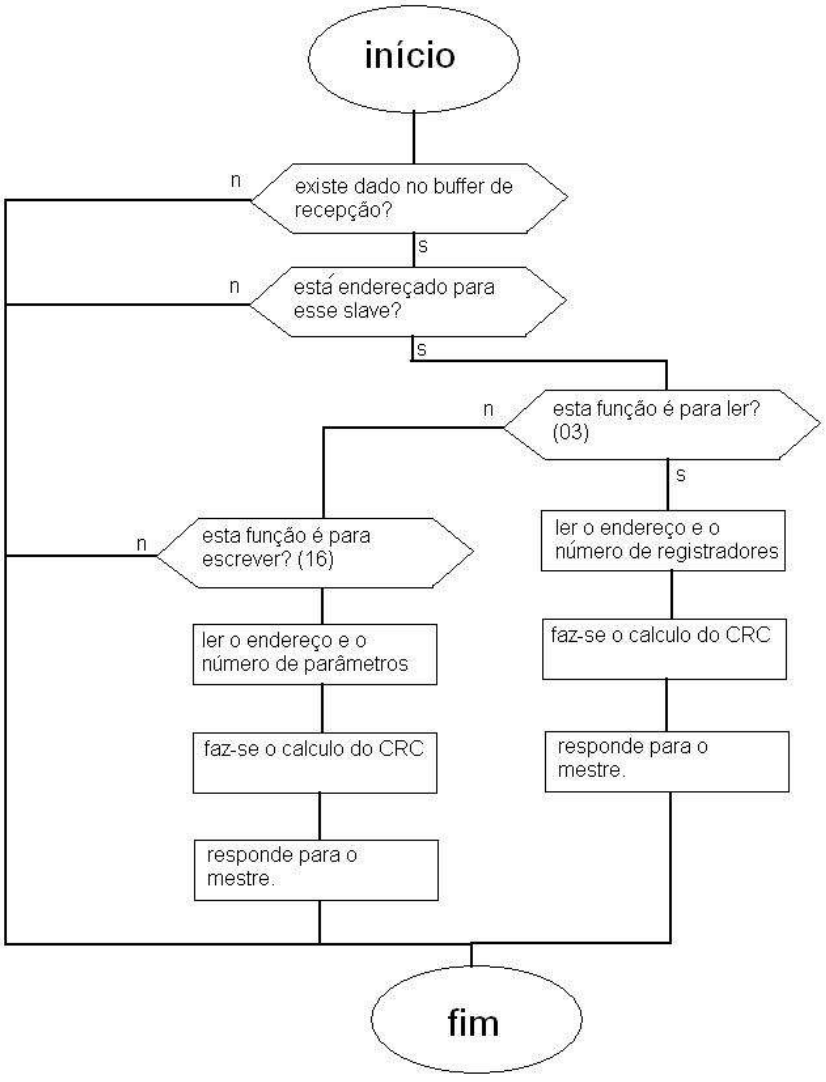
O modo Modbus RTU com microcontrolador PIC desse projeto, mostrado no link http://www.youtube.com/watch?v=KUd1JkwGJNk , suporta funções de leitura (3) e escrita (16).

```
#include <SanUSB.h>
#include <usb_san_cdc.h>// Biblioteca para comunicação serial
#byte port_a = 0xf80
#byte port_b = 0xf81

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]=usb_cdc_getc();
  buffer[3]=usb_cdc_getc();
  buffer[4]=usb_cdc_getc();
  buffer[5]=usb_cdc_getc();
  buffer[6]=usb_cdc_getc();
  buffer[7]=usb_cdc_getc();
  delay_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]=usb_cdc_getc();
  buffer[3]=usb_cdc_getc();
  buffer[4]=usb_cdc_getc();
  buffer[5]=usb_cdc_getc();
  buffer[6]=usb_cdc_getc();
  buffer[7]=usb_cdc_getc();
  buffer[8]=usb_cdc_getc();
  buffer[9]=usb_cdc_getc();
  buffer[10]=usb_cdc_getc();
  delay_ms(3);
  tamanhodata = 6;
  CRC16();
  port_b = 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],buffer[5],buffer[6],
buffer[7]); //7 bytes
}

main()
{
  clock_int_4MHz();
  usb_cdc_init(); // Inicializa o protocolo CDC
  usb_init(); // Inicializa o protocolo USB
  usb_task(); // Une o periférico com a usb do PC
  set_tris_b(0b00000000);
  while(1)
  {
    if (usb_cdc_kbhit(1))
    {
//verifica se acabou de chegar um novo dado no buffer USB, depois o kbhit é zerado para próximo dado

      buffer[0]=usb_cdc_getc();
      z = buffer[0];
      if (z==1) //verifica se o endereco do slave e igual a 1
      {
        buffer[1]=usb_cdc_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 abaixo:



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.

Introdução à *Multitasking* e Sistemas operacionais em tempo real (RTOS)

Um sistema operacional em tempo real (RTOS) é um programa (geralmente chamado de kernel), que controla a alocação de tarefas quando o processador está operando em um ambiente multitarefas (*multitasking*).

O RTOS decide, por exemplo, que tarefa executar em seguida, como coordenar a prioridades de tarefas e como transmitir dados e mensagens entre as tarefas. O compilador CCS dispõe de bibliotecas embutidas para executar multitarefas paralelas, ou seja, programação *multitasking*. Como exemplo, podemos citar a execução de três tarefas quaisquer como comutar três leds ou funções em freqüências diferentes, de forma independente e paralela. Algumas funções RTOS multitarefas do compilador CCS estão descritas abaixo:

rtos_run () inicia a operação de RTOS. Todas as operações de controle de tarefas são implementadas após chamar essa função.

rtos_terminate () termina a operação de RTOS. O controle retorna ao programa original sem RTOS. Na verdade, esta função é um retorno de rtos_run ().

rtos_enable () esta função ativa a tarefa para que ela possa ser chamado pelo rtos_run ().

rtos_disable () esta função desativa a tarefa para que ela não possa mais ser chamado pelo rtos_run (), a menos que seja reativada pela função rtos_enable ().

rtos_msg_send () recebe esta função envia um byte para a tarefa especificada, onde ele é colocado na tarefa mensagem.

rtos_msg_read () lê o byte localizado na tarefa mensagem.

rtos_msg_ poll () retorna true se houver dados na tarefa mensagem.

Preparação para RTOS Multitasking

Além das funções anteriores, o comando de pré-processamento # use rtos() deve ser especificado no início do programa, antes de chamar as funções RTOS. O formato deste comando é pré-processador:

#use rtos(timer=n, minor_cycle=m)

onde o timer está entre 0 e 4 e especifica o timer do microcontrolador que será usado pelo RTOS; e minor_cycle é o maior tempo de qualquer tarefa executada. O número deve ser seguido por s, ms, us ou ns.

Declaração de uma tarefa

A tarefa é declarada como qualquer outra função em C, mas as tarefas de uma aplicação multi-tasking não tem argumentos e não retornam nenhum valor. Antes de uma tarefa ser declarada, é especificado o comando de pré-processamento de tarefas. O formato deste comando de pré-processamento é:

#task(rate=n, max=m, queue=p)

onde **rate** especifica em quanto tempo a tarefa deve ser chamada. O número especificado deve ser seguido por por s, ms, us ou ns; **max** especifica o tempo máximo em que o processador pode executar a tarefa. O tempo especificado aqui deve ser igual ou inferior a o tempo especificado por minor_cycle. **queue** é opcional e especifica o número de bytes a serem reservados para a tarefa receber mensagens de outras tarefas. O programa *multitasking* abaixo comuta três leds em freqüências diferentes, de forma independente e paralela.

```
#include <SanUSB.h>

//O firmware abaixo executa uma multi-tarefa (multi-tasking) onde 3 LEDs comutam
//independentes e simultaneamente

// Define qual o timer utilizado para o multitasking e o maior tempo de cada tarefa (minor_cycle) do RTOS
#use rtos(timer=0, minor_cycle=10ms)

#task(rate=250ms, max=10ms) // Declara TAREFA 1 - chamada a cada 250ms
void task_B7()
{output_toggle(PIN_B7); }// comuta B7 – inverte o estado de B7

#task(rate=500ms, max=10ms) // Declara TAREFA 2 - chamada a cada 500ms
void task_B6()
{output_toggle(PIN_B6);} // comuta  B6

#task(rate=1s, max=10ms) // Declara TAREFA 3 - chamada a cada segundo
void task_B0()
{output_toggle(PIN_B0); } // comuta  B0

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

```
set_tris_b(0); // Configura PORTB como saída

setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1); //ciclo de máquina de 1us
rtos_run(); // inicia o RTOS (sistema operacional em tempo real)
}
```

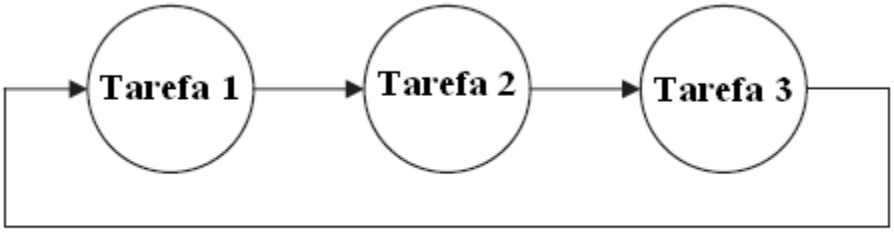
Em um sistema multitarefa, inúmeras tarefas exigem tempo da CPU, e uma vez que existe apenas uma CPU, é necessária alguma forma de organização e coordenação para cada tarefa tenha o tempo que necessita. Na prática, cada tarefa tem um intervalo de tempo muito curto, assim parece que as tarefas são executadas 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;

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 abaixo.



Implementação de máquina de estado

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.

```
for(;;)
{
        state = 1;
        switch (state)
        {
                CASE 1:
                        implement TASK 1
                        state++;
                        break;
                CASE 2:
                        implement TASK 2
                        state++;
                        break;
                CASE 3:
                        implement TASK 3
                        state = 1;
                        break;
        }
        Delay_ms(n);
}
```

Máquina de Estado implementa em linguagem C

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.

```
for(;;)
{
        state = 1;
        switch (state)
        {
                CASE 1:
                        implement TASK 1
                        state = 2;
                        break;
                CASE 2:
                        implement TASK 2
                        state = 3;
                        break;
                CASE 3:
                        implement TASK 3
                        state = 1;
                        break;
        }
        Delay_ms(n);
}
```

Selecionando o próximo estado a partir do estado atual

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.

OUTROS MODELOS DE PIC

Neste tópico são descritos alguns modelos das famílias PIC16F e 18F (F de memória de programa *flash*, ou seja, pode ser apagada e gravada por completo diversas vezes).

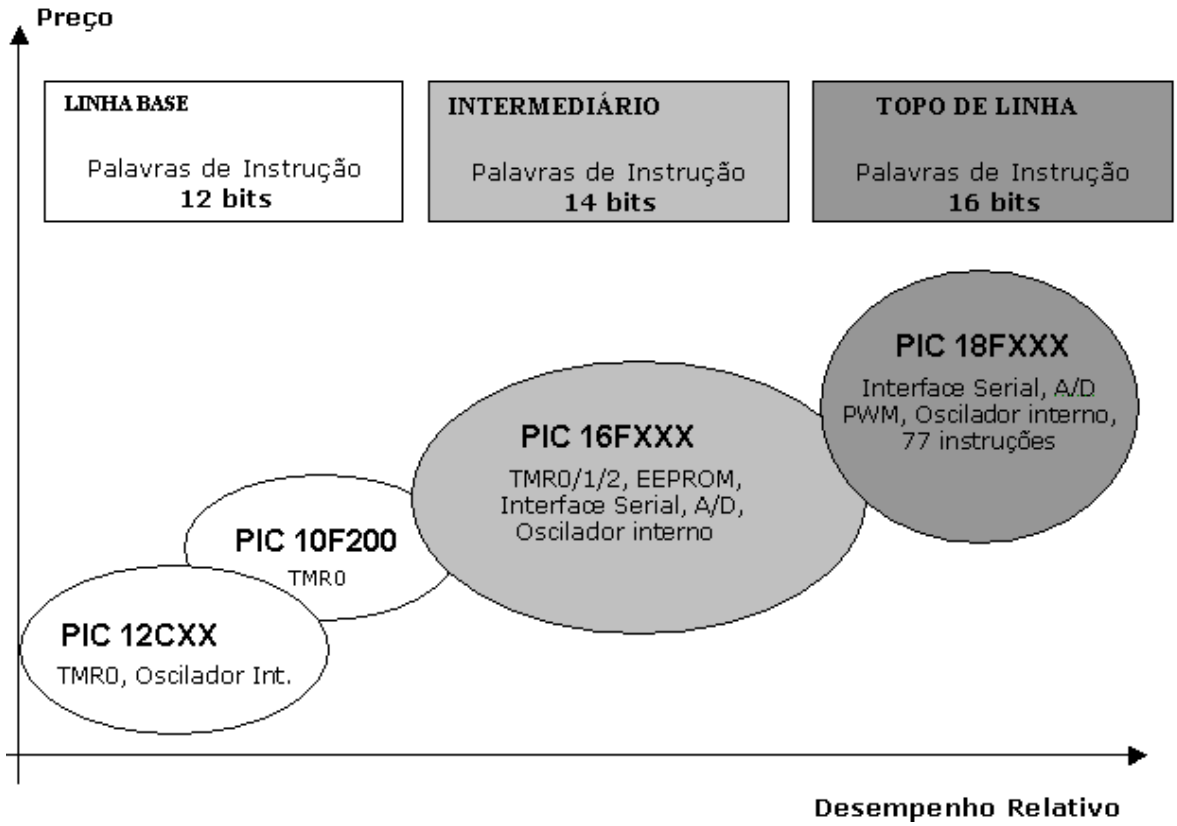


Figura 1 –Relação custo x Desempenho dos Microcontroladores PIC

O **PIC16F84**, que foi um dos primeiros a usar memória de programa flash com recurso de gravação serial (ICSP) permitindo a divulgação de projetos simples de gravadores para PIC e que ajudou a disseminar a utilização dos microcontroladores Microchip.

Dentre os modelos mais difundidos dos microcontroladores intermediários depois do PIC16F84, estão o PIC16F628A (sem conversor AD interno) e o PIC16F877A (40 pinos e sem oscilador interno).

Devido à melhor relação custo/desempenho na maioria das práticas propostas, os modelos mais recomendados nesta apostila da família PIC16F e descritos abaixo são o PIC16F628A, PIC16F688, o PIC16F819, o PIC16F873A e o PIC16F877A. As características destes cinco modelos são descritas abaixo:

COMPONENTE	Flash Words (14 bits)	RAM (Bytes)	EEPROM (Bytes)	I/O	USART	Oscilador interno	Bootloader disponível	AD (10 bits)	Timers 8/16- bit
PIC16F628A	2K	224	128	16	S	S	N	-	2/1
PIC16F688	4K	256	256	12	S	S	N	8	1/1
PIC16F819	2K	256	256	16	N	S	N	5	2/1
PIC16F873A	4K	256	256	22	S	N	S	8	2/1
PIC16F877A	8K	368	256	33	S	N	S	8	2/1

O **PIC16F628A** é visto como o sucessor do PIC16F84, com uma configuração estendida de RAM e encapsulamento compatível de 18 pinos, necessitando de pouca ou nenhuma alteração de hardware. Apresenta mais recursos e as seguintes características: CPU RISC de 35 instruções, até 20 MHz, ou 4MHz com ***oscilador RC interno***, 16 pinos de entrada e/ou saída, dois comparadores analógicos, gerador PWM, 3 Timers, interface de comunicação serial assíncrona (USART). Sua memória ROM é de 2KWords (permite até 2048 instruções em um programa), e memória RAM de 224 bytes, com EEPROM de 128 bytes. No PIC16F628A, a porta A é *tristate,* ou seja, para nível lógico alto na saída é necessário colocar resistores de pull-up externo e a porta B contém weak (fraco) pull-up interno habilitado por software (***port_b_pullups(true);***). Caso seja

necessário utilizar LCD, ele pode ser conectado em qualquer porta, porque ele apresenta pull-up interno nos seus pinos.

Os programas em C com o compilador CCS® podem ser feitos utilizando instruções do próprio compilador, como **output_high(pin_b0)** mostrado no primeiro programa para piscar um led, ou configurando diretamente os registros de controle do PIC após definir o seu endereço com a diretiva **#byte**, como por exemplo, **#byte port_b = 0x06** ou **#byte osccon = 0x8f**.

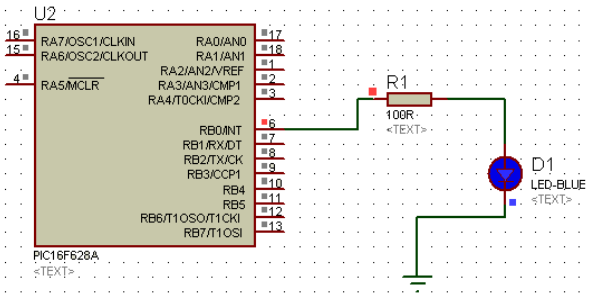
PROGRAMA PARA PISCAR UM LED EM B0 COM O 16F628A:

```
//Programa para piscar um led no pino B0 com comandos do CCS output_high e  output_low
#include <16f628A.h>
#fuses INTRC_IO, NOMCLR, NOWDT, NOLVP, NOBROWNOUT, PUT, NOPROTECT//sem cristal
#define LED PIN_B0
#use delay(clock=4000000)

main()
{
  while(true)//Ficará repetindo enquanto expressão entre parenteses for verdadeira
  {         // como a expressao é sempre verdadeira (true), o loop é infinito
  output_high(LED);
  delay_ms(500);
  output_low(LED);
  delay_ms(500);
  }
 }
```

OU

```
#include <16f628A.h>
#fuses INTRC_IO, NOMCLR, NOWDT, NOLVP, NOBROWNOUT, PUT, NOPROTECT
#use delay(clock=4000000)
#byte port_b = 0x06  // Aponta a posição 6h da RAM como PORT_B para o compilador CCS

main()
{
port_b_pullups(true);
set_tris_b(0b11111110);   //necessário definir: pin_b0 como saída e demais como entrada
inicio:
  bit_set(port_b,0);
  delay_ms(700);
  bit_clear(port_b,0);
  delay_ms(700);    // Outra forma de mover o valor para a port_b é:
            port_b=0b00000001;
            delay_ms(500);
            port_b=0b00000000;
```

```
        delay_ms(500);
goto inicio;
  }
```



Na verdade, o circuito de Hardware necessário para fazer o LED piscar com oscilador interno e o reset interno é apenas a conexão do pino V_{DD} em +5Ve V_{SS} em G_{ND}, e o LED conectado ao pino RB0 através de um resistor de 1K.

O **PIC16F688** é um modelo intermediário, de baixo custo, e apresenta todos os periféricos básicos internos como EEPROM, 8 conversores AD internos de 10 bits, oscilador RC interno e USART para comunicação serial. Ele apresenta duas portas bidirecionais que são a porta A, contém weak (fraco) pull-up interno em A0, A1, A2, A4 e A5 configurado por software (*port_a_pullups(true);*) e a porta C *tristate*, ou seja, utilizada geralmente como entrada. Para nível lógico alto na saída é necessário colocar resistores de pull-up externo, geralmente de 10K.

A porta A apresenta ainda um pino de interrupção externa (A2), habilitado com borda de descida quando o INTEDG (bit 6) do registro OPTION (81h) é zero (*default* é um), e interrupção por mudança de estado em todos pinos da porta A.

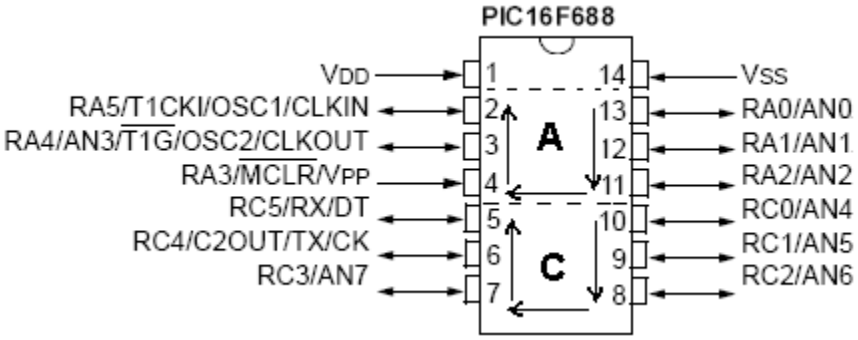
Uma desvantagem do PIC16F688 em relação aos outros modelos como o PIC16F628A e o 16F877A é que ele não apresenta os resistores de *pull-up* internos da porta A estáveis para entrada (somente para saída como, por exemplo, acionamento de LEDs), ou seja, sempre que um

pino for aterrado com um botão, deve-se colocar um resistor de pull-up externo para retornar o nível lógico alto quando o botão for liberado.

Para a simulação do PIC16F688 no Proteus®, embora se verifique que os modelos apresentam alguns erros em relação a operação real, pode-se utilizar o modelo PIC16F88 considerando a diferença das portas, ou o PIC16F870 trocando o fusível INTRC_IO (clock interno) por XT (clock externo até 4 MHz).

Para gravar a flash do PIC16F688 incluindo o oscilador RC interno, é recomendável configurar o registro OSCCON que apresenta oito freqüências de clock, para freqüências de 4 ou 8 MHz (*OSCCON=0b01100100;).* Um exemplo de programa para piscar alternadamente Leds na porta A, com oscilador interno e *reset* interno, é mostrado a seguir:

```
#include <16f688.h>  //pisca pinos alternados da porta A
#use delay (clock=4000000)
#fuses PUT,NOWDT,NOBROWNOUT,INTRC_IO,NOMCLR
#byte port_a = 0x05 // Aponta a posição 5h da RAM como PORT_A para o compilador CCS
#byte tris_a = 0x85
#BYTE OSCCON=0X8F  //Byte de controle do oscilador interno
main()
  {
   OSCCON=0b01100100; // Configura o oscilador como interno e com freqüência de 4MHz
   port_a_pullups(true); //Pull-up interno em A exceto A3
   set_tris_a(0b00000000);//Porta A como saída
   while(1) //Loop infinito
    {
    port_a=0b01010101;
    delay_ms(500);
    port_a=0b10101010;
    delay_ms(500);
    }
  }
```



Este modelo também pode utilizar o oscilador interno para gerar a taxa de comunicação serial, ou seja, não é necessário o uso de um cristal para esse fim, como mostra o exemplo abaixo:

```
#include <16F88.h>
#fuses  PUT,NOBROWNOUT,NOWDT,INTRC_IO,NOMCLR
#use delay(clock=8000000)
#use rs232(baud=38400,xmit=PIN_B5,rcv=PIN_B2) // no 16F688 xmit=PIN_C4,rcv=PIN_C5
#BYTE OSCCON=0X8F
main()
{
OSCCON=0B01110100; // oscilador interno com 8MHz
while(1)
  {
printf("AUTOMACAO INDUSTRIAL\r\n ");
getchar();
  }
}
```

No lugar de printf("AUTOMACAO INDUSTRIAL\r\n") também pode–se utilizar puts("AUTOMACAO INDUSTRIAL").

```
#include <16F688.h>
#fuses  PUT,NOBROWNOUT, NOWDT, INTRC_IO,NOMCLR
#use delay(clock=8000000)
#use rs232(baud=115200,xmit=PIN_C4,rcv=PIN_C5)
#BYTE OSCCON=0X8F
main()
{
OSCCON=0B01110100;
while(1)
        {
```

```
puts("AUTOMACAO INDUSTRIAL");
getchar();
        }
}
```

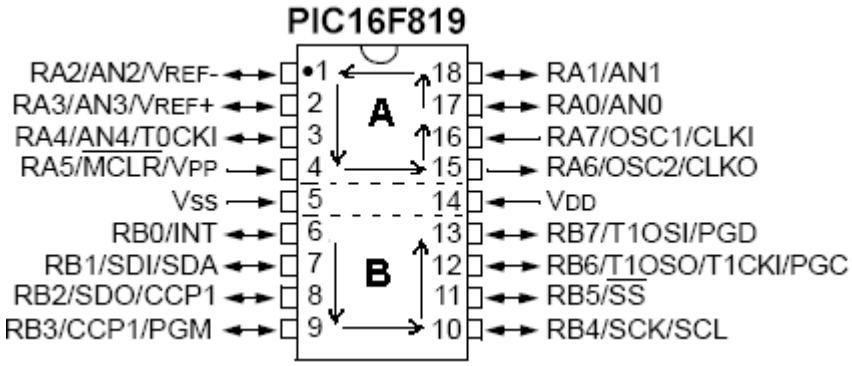
O compilador CCS® também apresenta uma interface de comunicação serial com mostra a figura abaixo que é o resultado do exemplo. Para acessá-lo basta acessar *tolls -> serial port monitor* e depois configure a taxa e a porta de comunicação serial em *configuration -> set port options*.

Configurando o PIC16F688 com uma freqüência no oscilador interno de 8MHz pode-se atingir na prática um *Baude Rate* (taxa de comunicação serial) de até 115200 bps.

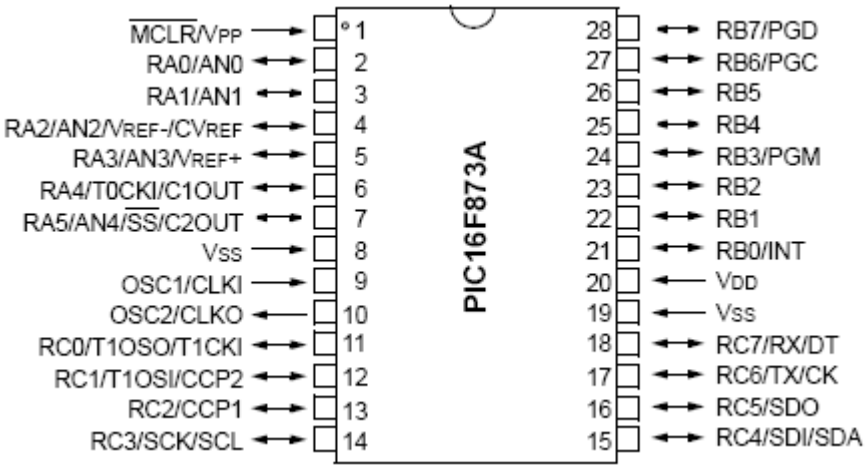
Como o erro de comunicação serial tende a crescer mediante o aumento da taxa de comunicação serial, o uso desta taxa elevada na prática comprova a estabilidade do oscilador interno.

As principais características de cada modelo podem ser vistas dentro da pasta *Devices* no compilador CCS® ou no próprio *Datasheet* com informações mais completas. Note que o registro 8Fh da RAM (que corresponde ao OSCCON no PIC16F688 e no PIC1F819) pode ser preenchido em qualquer outro modelo sem o OSCCON que não há problema, pois esta posição é livre.

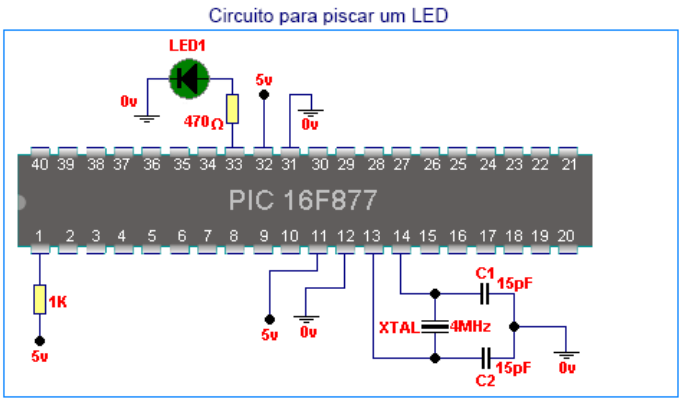
O **PIC16F819** é um microcontrolador intermediário da família 16F, também de 18 pinos, com inclusão de 5 conversores AD, oito velocidades de Clock RC interno, porta serial síncrona (SSP), interface I2C e até 16 pinos de Entrada/Saída. Neste modelo, a Porta A tem *pull-up* interno de 0 a 4 e os demais são *Tristate*) e a Porta B contém pull-up interno configurável por software).

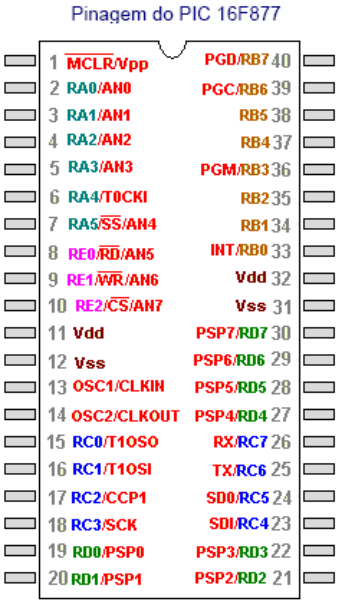


O **PIC16F873A** é um com 28 pinos, deste 22 para I/O com 8 conversores AD de 10 bits e USART. Pode ser considerado como a versão compacta do 16F877A com menor custo com bootloader disponível (*San Bootloader*).. Neste modelo, a Porta A tem *pull-up* interno de 0 a 3 e os demais são *Tristate*), a Porta B contém pull-up interno configurável por software e a Porta C apresenta *pull-up* interno.



O **PIC16F877A** é atualmente um dos mais difundidos da família 16F, apresentando mais recursos, embora maior custo. Dentre seus recursos, podemos citar como os mais importantes: 33 pinos de Entrada/Saída, CPU RISC de 35 instruções, com clock de até 20MHz (5 milhões de instruções por segundo). Até 8 KWords de memória de programa, 368 bytes de RAM, 256 bytes de EEPROM, dois comparadores e geradores PWM, 8 canais de conversão A/D de 10 bits, interface I2C e comunicação serial assíncrona com bootloader disponível (*San Bootloader*). Neste modelo, a Porta A tem *pull-up* interno de 0 a 3 e os demais são *Tristate*, a Porta B contém pull-up interno configurável por software), a Porta C apresenta *pull-up* interno e as portas D e E são *Tristate.* Não apresenta oscilador RC interno e deve ter um pino de reset externo, então o circuito básico para piscar um LED em B0 é:





Como pode ser visto, este modelo necessita de um pull-up externo no pino de reset e um circuito de clock externo.

Além do PIC18F2550 da família PIC18F, é importante destacar o modelo 18F452 difundido no mercado nacional.

Este modelo contém um contador de programa com 16 bits e, por isso, podem dar um pulo para chamada de uma função de até 2^{16}, ou seja, 64K (65536 posições), diferente da família 16F com contador de programa com 11 bits, ou seja , só é capaz de fazer uma chamada de uma posição distante somente a 2K (2048 posições). Assim, quando se trabalha com programas de alto nível em C, com várias aplicações na função principal *main* ou na função de interrupção serial, é necessário utilizar um modelo da família 18F, pois facilmente se tornam maiores que 2K e o contador de programa não consegue mais fazer a chamada destas funções. Além disso, em operações matemáticas ou com matrizes, o compilador utiliza a memória RAM para variáveis de

dados e, também, como memória *cash*, ou seja, ele guarda os valores intermediários das operações matemáticas complexas em outras posições da RAM. Assim, para programas de alto nível que utilizam operações matemáticas ou matrizes também é necessário modelos da família 18F. As características do modelo 18F452 são mostradas na tabela abaixo.

COMPONENTE	Flash Words (16 bits)	RAM (Bytes)	EEPROM (Bytes)	I/O	USART	Oscilador interno	Bootloader disponível	AD (10 bits)	Timers 8/16- bit
18F452	16K	1536	256	33	S	N	S	13	1/3

As principais diretivas de pré-processamento e os comandos do programa em C são comentados a seguir:

PRINCIPAIS DIRETIVAS DE PRÉ-PROCESSAMENTO(#)

São comandos utilizados para especificar determinados parâmetros internos utilizados pelo compilador no momento de compilar o código-fonte. O compilador CCS possui uma grande quantidade de diretivas, entre elas as mais comuns são:

#INCLUDE: Utiliza-se essa diretiva para inserir arquivos de biblioteca e funções no código do programa atual.

Exemplo:

```
#include <16f628A.h> // Inclui o arquivo da pasta padrão do compilador
```

#FUSES: Especifica o estado dos fusíveis de configuração do dispositivo. Lembrando que as opções nas especificadas são deixadas no padrão do dispositivo. Para verificar as opções

disponíveis pode-se verificar no *Data sheet* do componente ou em *View > Valid Fuses* do compilador CCS.

Opção	Descrição
LP	Oscilador LP
RC	Oscilador RC
XT	Oscilador XT menor ou igual a 4 MHz
HS	Oscilador HS
INTRC	Oscilador interno *
INTRC_OSCOUT	Oscilador interno com saída de clock *
INTRC_IO	Oscilador interno, OSC1 e OSC2 como I/O *
WDT	Watchdog habilitado
NOWDT	Watchdog desabilitado
PROTECT	Proteção do código habilitado
PROTECT_75%	Proteção habilitada para 75 % da memória *
PROTECT_50%	Proteção habilitada para 50 % da memória *
NOPROTECT	Proteção do código desabilitada
PUT	Temporizador de Power-up ligado
NOPUT	Temporizador de Power-up desligado
BROWNOUT	Reset por queda de tensão habilitada *
NOBROWNOUT	Reset por queda de tensão desabilitada *
MCLR	Pino MCLR utilizado para RESET *
NOMCLR	Pino MCLR utilizado como entrada *
LVP	Programação em baixa tensão habilitado *
NOLVP	Programação em baixa tensão desabilitado *

* Esta opção não está disponível em todos os dispositivos

Exemplo:

#fuses **INTRC_IO**, **NOMCLR**, **NOWDT**, **NOLVP**, **NOBROWNOUT**, PUT, NOPROTECT

Uma grande parte da família PIC16F como, por exemplo, o 16F628A, 16F688 e 16F819, possuem um circuito RC de clock interno configurável por software, transformando os pinos OSC1 e OSC2 em pinos de I/O sem saída de pulso de clock. Esta função é selecionada pelos fusíveis (**INTRC_IO**), o que permite, além de reduzir o custo do projeto por não necessitar de um cristal externo, utilizar os pinos referentes à localização do cristal (no 16F628A os pinos 15(A6) e 16(A7)), como Entrada/Saída.

Vale salientar que também é possível utilizar, em alguns modelos, o pino de Reset externo pino 4 (A5) como I/O, habilitando o Reset interno (**NOMCLR**). Dessa forma, o circuito auxiliar do PIC reduz-se somente à alimentação V_{DD} (+5V no pino 14) e V_{SS} (Gnd no pino 5), ou seja, este PIC de 18 pinos apresenta 16 pinos de I/O. Neste caso é importante desabilitar o *Watch Dog Timer* (**NOWDT**), porque a flag de estouro ficará "ressetando" o PIC caso não seja periodicamente limpa.

Alguns modelos como o PIC16F628A e o PIC16F877A apresentam em um pino de IO (chamado de PGM) com a função de programação em baixa tensão (LVP) habilitada. Dessa forma, é importante desabilitar essa função (**NOLVP**) para garantir a função do pino como entrada e saída.

Para desabilitar o reset para tensãoe abaixp de 4V, utiliza-se o comando (**NOBROWNOUT**).

É possível acionar o *Power up Timer* (PUT), que é um o retardo de 72 ms após o Reset para evitar ruídos na alimentação do microcontrolador quando a fonte é ligada.

Caso não seja necessário proteger o programa contra cópias (NOPROTECT).

#DEFINE: Nome utilizado para substituir um identificador, que pode ser um byte da memória de dados, um bit ou uma constante. #DEFINE NOME IDENTIFICADOR.

Exemplo:

```
#define LED PIN_B0
```

#USE DELAY(): Informa a velocidade de clock do sistema. Esta diretiva é utilzada para o cálculo das funções de atraso (**delay_us() e delay_ms()**) e também para gerar a taxa de transmissão e recepção na comunicação serial (***Baud Rate***).

Exemplo:

```
#use delay(clock=4000000)
```

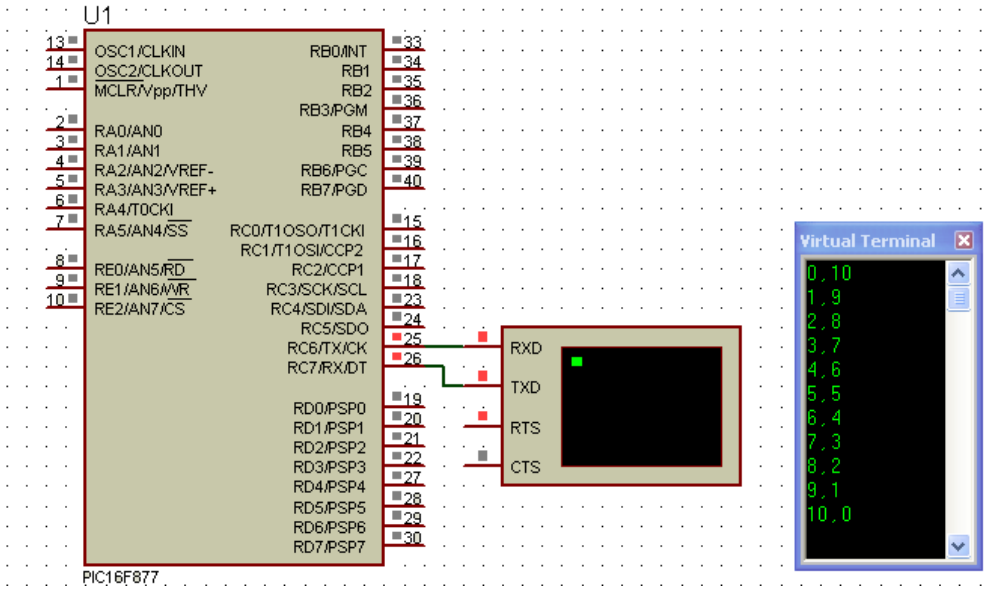
#USE RS232(): Utilizada para ordenar o compilador a gerar o código para comunicação serial assíncrona padrão rs232, como mostra a prática 2.

Exemplo:

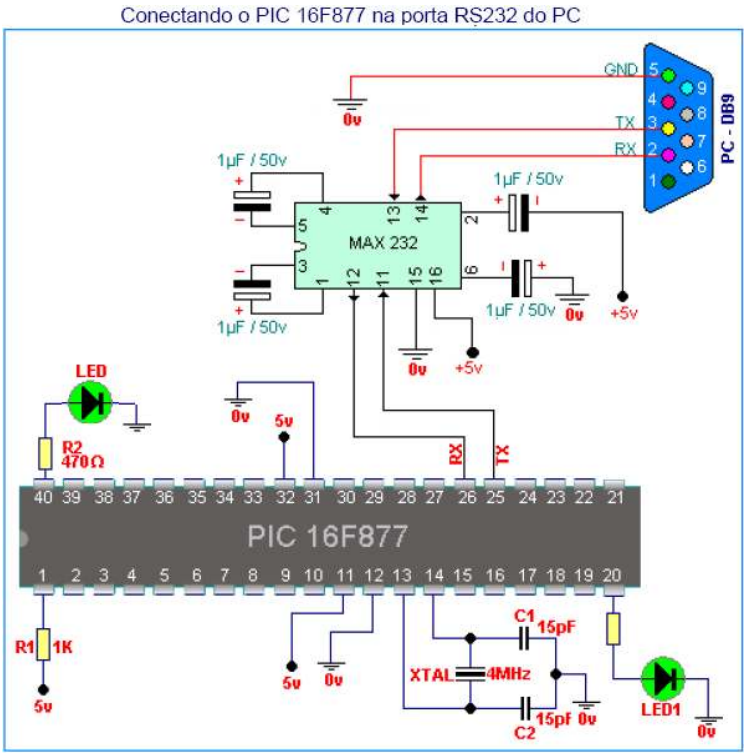
```
#use rs232(baud=2400,parity=N,xmit=PIN_C6,rcv=PIN_C7)
```

PROGRAMA PARA ENVIAR VARIAÇÕES DE X E Y PELA SERIAL:

```
#include <16f877.h>
#use delay(clock=4000000)
#fuses XT, NOWDT, NOLVP, NOBROWNOUT, PUT, NOPROTECT
#use rs232(baud=2400,parity=N,xmit=PIN_C6,rcv=PIN_C7)

main()
{ int x,y;
for (x=0,y=10;x<=10;x++,y--) printf("%u , %u\r\n",x,y);
  }
```



Neste programa deve-se verificar no simulador Proteus, com o botão direito sobre o dispositivo, se o clock do PIC está em 4MHz e se Taxa de transmissão do Virtual Terminal, que é o receptor, também está em 2400 bps como o programa.



O circuito acima é conectado à porta RS232 do computador por meio do driver Max232. O LED1 é ligado ou desligado através do computador.

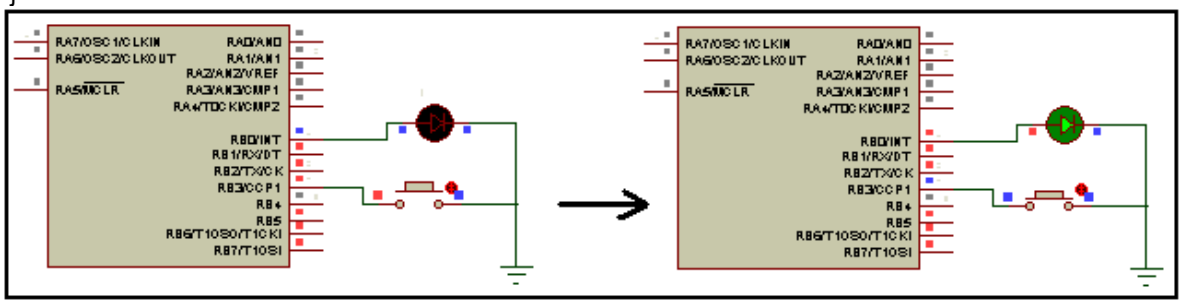
PROGRAMA PARA ASCENDER UM LED QUANDO CHAVE FOR ACIONADA:

```
#INCLUDE <16F628A.H>
#use delay(clock=4000000)
#fuses INTRC_IO, NOMCLR, NOWDT, NOLVP, NOBROWNOUT, PUT, NOPROTECT
#define led pin_b0  //o comando  output_high(led); aciona o pull-up interno do pino
#define chave pin_a3 // Necessita de um rsistor de pull-up externo

main()
{
  for( ; ; ) {        output_low(led);      // loop infinito
                                  while(!input(chave))  {  output_high(led);    }
                                   // enquanto a chave for zero ascende led
            }
 }
```

PORT_B_PULLUPS(): HABILITAÇÃO DOS PULL-UPS INTERNOS

A porta B pode assumir resistores de pull-up internos habilitados por software com o comando **port_b_pullups(true);** eliminando a necessidade de resistores externos ligados ao +Vcc. Veja próximo exemplo:

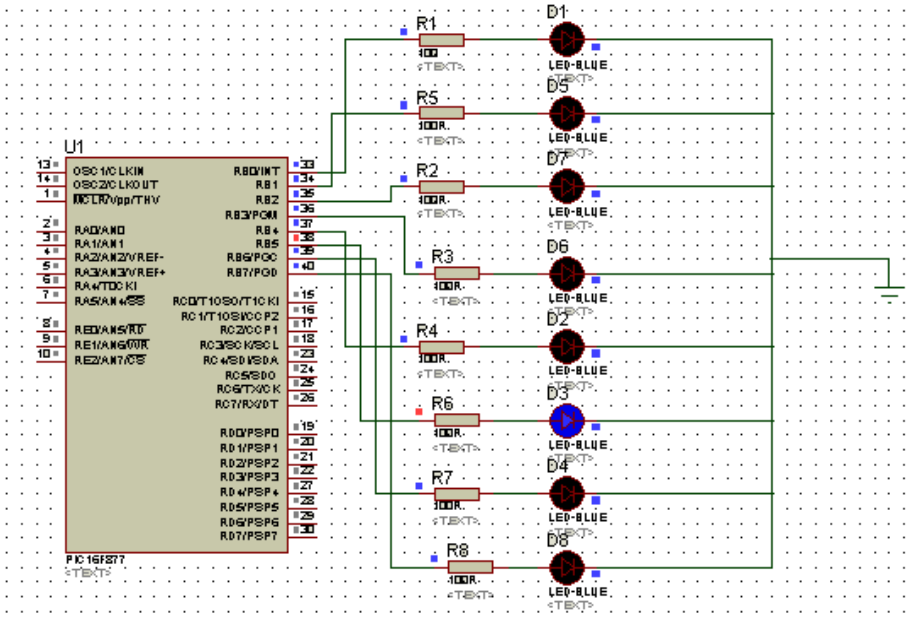
```
#INCLUDE <16F628A.H>
#use delay(clock=4000000)
#fuses INTRC_IO, NOMCLR, NOWDT, NOLVP, NOBROWNOUT, PUT, NOPROTECT
#define led pin_b0
#define chave pin_b3

main()
{   port_b_pullups(true);  //Habilita o pull-up interno de todos os pinos da porta B
   for( ; ; ) {        output_low(led);      // loop infinito
                                    while(!input(chave))  {  output_high(led);    }
                                     // enquanto a chave for zero ascende led
            }
}
```



PROGRAMA PARA ACENDER UMA SEQUÊNCIA DE LEDS NA PORTA B

```
#include <16F877.H>
#fuses XT, NOWDT, NOPROTECT
#use Delay(Clock=4000000)  // Seta pelo clock do Cristal utilizado
#use standard_IO(B) // Saída ou Entrada direta para Porta B - mais rápida
#byte porta_do_uc=0x06 //  Define porta_do_uc como a posição de memória de dados 6h(port_b)
 main()
```

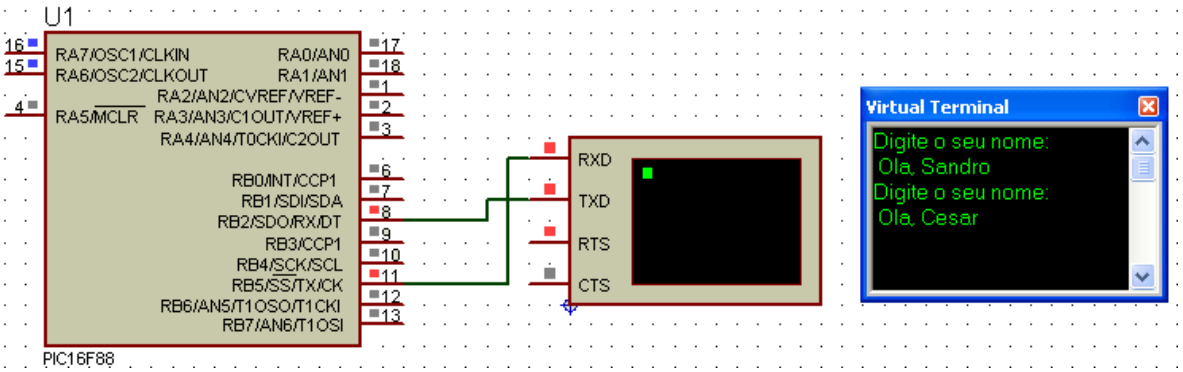
```
{
port_b_pullups(true);
set_tris_b(0x00); // Define Porta B como saída porque não se utiliza o output_high(PIN_B0);
porta_do_uc = 0x01;
      while(true)
   {
      rotate_left(&porta_do_uc, 1);//Rotacione o conteúdo de 1 byte na memória
                                   // apontada por porta_do_uc
      delay_ms(1000);
   }
}
```



MENSAGEM PELA RS-232 COM A FUNÇÃO GETS PARA LEITURA DE STRING

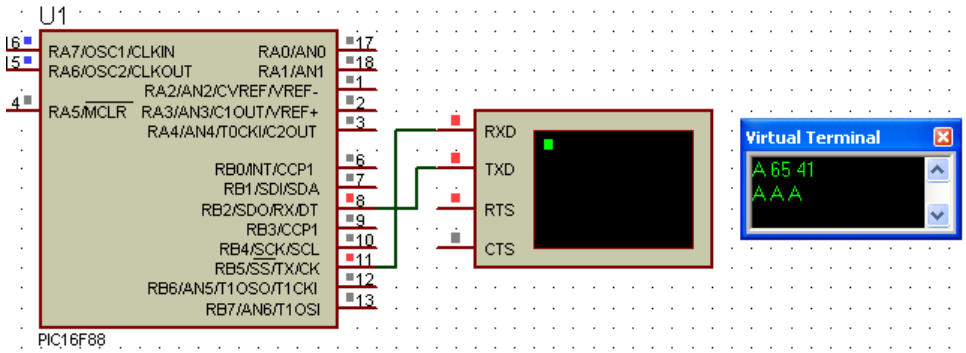
```
#include <16F88.h>
#fuses PUT,NOBROWNOUT,NOWDT,INTRC_IO,NOMCLR
#use delay(clock=8000000)
#use rs232(baud=9600,xmit=PIN_B5,rcv=PIN_B2) // no 16F688 xmit=PIN_C4,rcv=PIN_C5
#BYTE OSCCON=0X8F

main ()
{
while(1)
    {
      char nome[10];
      printf ("Digite o seu nome: ");
      gets (nome);
      printf ("\r\n Ola, %s\r\n",nome);
      getchar();
```

```
    }
}
```



PROGRAMA PARA VISUALIZAR A CONVERSÃO DE FORMATOS

```
#include <16F88.h>
#fuses PUT,NOBROWNOUT,NOWDT,INTRC_IO,NOMCLR
#use delay(clock=8000000)
#use rs232(baud=9600,xmit=PIN_B5,rcv=PIN_B2) // no 16F688 xmit=PIN_C4,rcv=PIN_C5
#BYTE OSCCON=0X8F

main()
{
printf("%c %d %x \r\n",'A','A','A');
printf("%c %c %c \r\n",'A',65,0x41);
getchar();
}
```

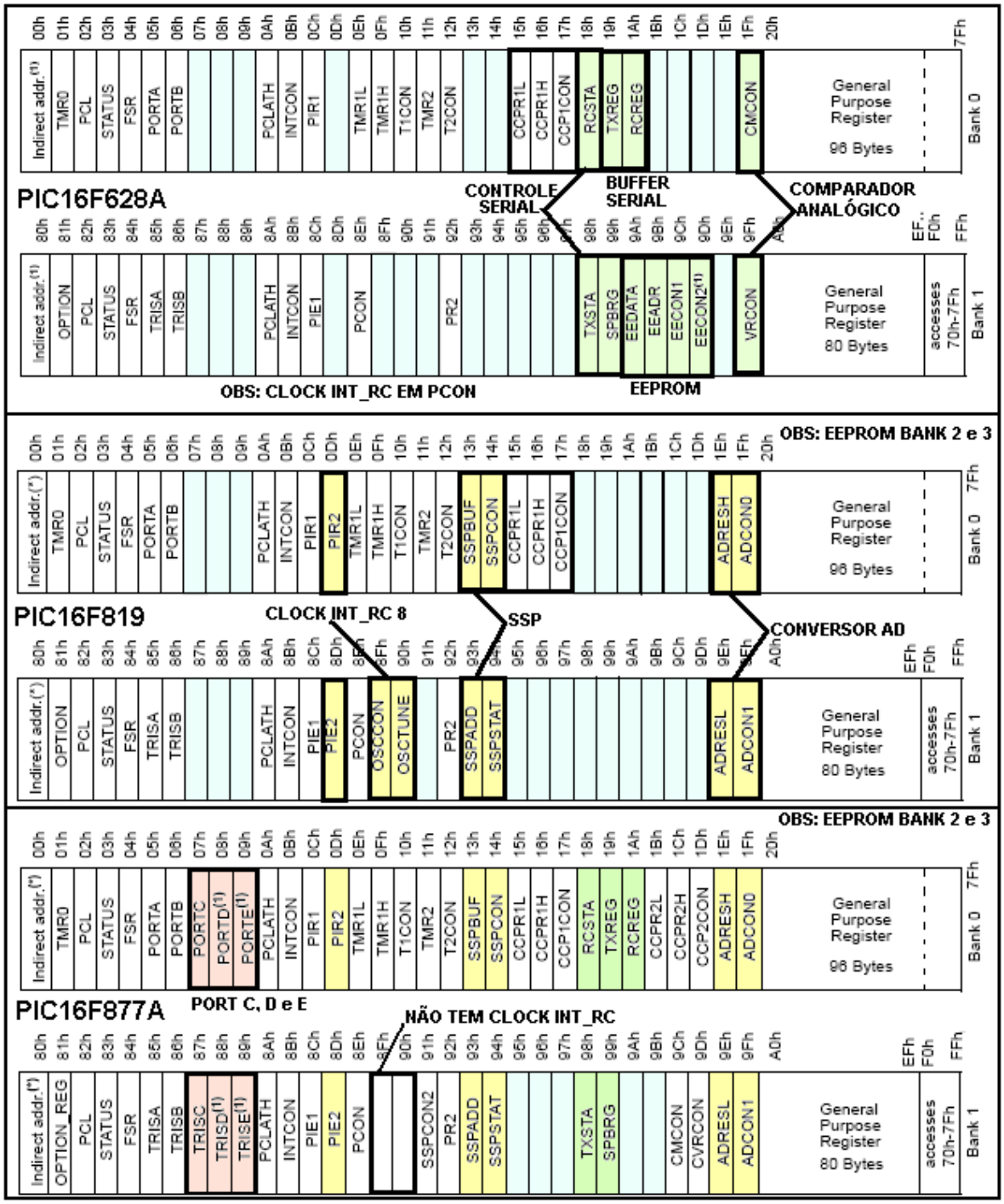


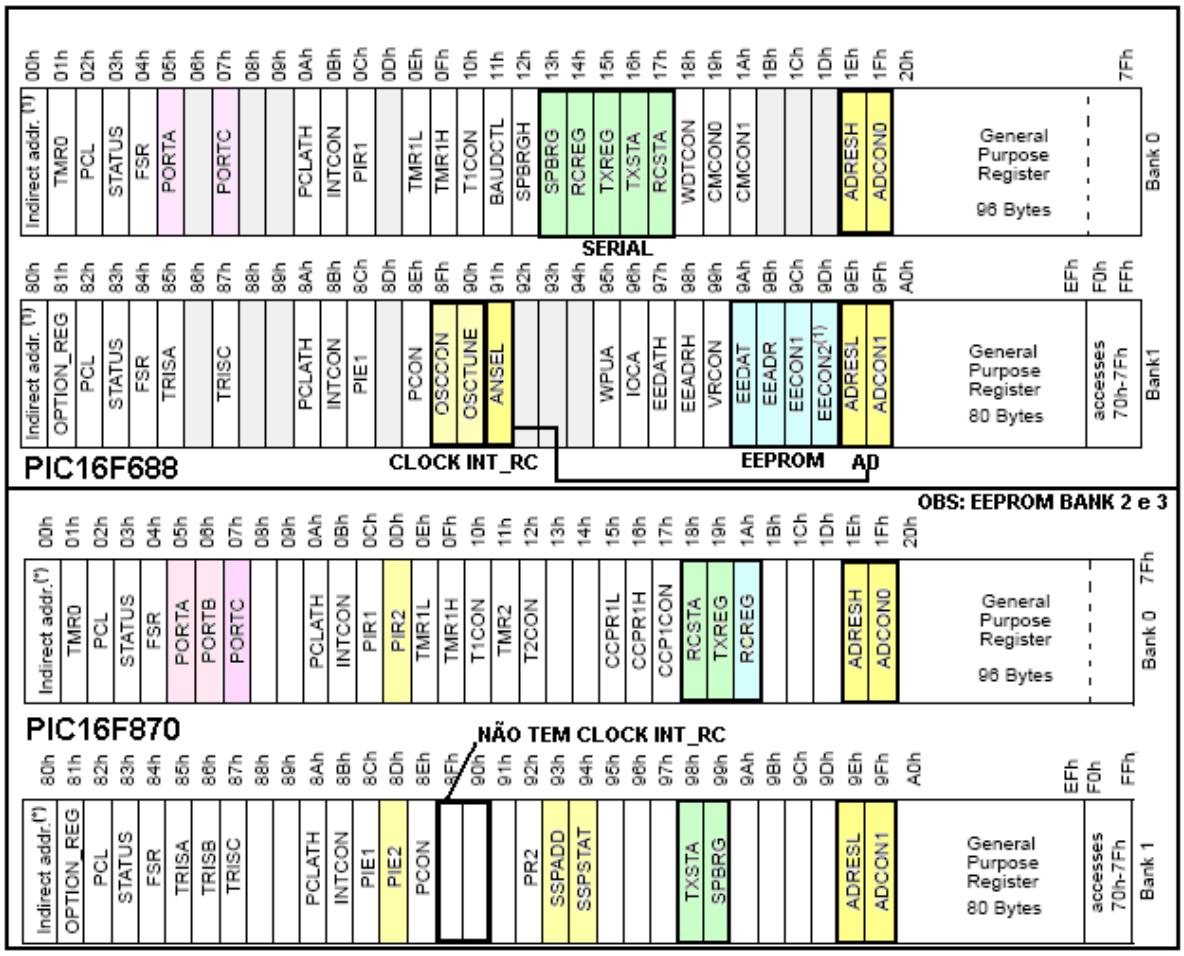
O MAPA DOS REGISTROS DE CONTROLE

Através do mapa dos registros dos bancos 0 e 1 da RAM é possível identificar os periféricos internos de cada microcontrolador. Note que os registros de controle em branco são comuns a todos os modelos aqui citados. Considerando o PIC16F628A como mais recente padrão de RAM da família PIC16F, nota-se que houve um acréscimo (em amarelo) de periféricos internos no modelo PIC16F819, que não apresenta interface de comunicação assíncrona (USART, em verde).

Conhecendo a posição dos registros de controle da RAM é possível configurar um registro de controle manualmente e, conseqüentemente, um periférico interno, bastando para isso, definir a posição e o nome do registro antes da função principal (exemplo: **#BYTE OSCCON=0X8F**) e configurá-lo dentro da função principal (**OSCCON=0B01100100;**).

O PIC16F877A, de 40 pinos, apresenta todas as características inerentes dos microcontroladores citados, inclusive USART, mais 3 portas C, D e E (lilás), porém não apresenta circuito de Clock RC interno como comprova a tabela a seguir:

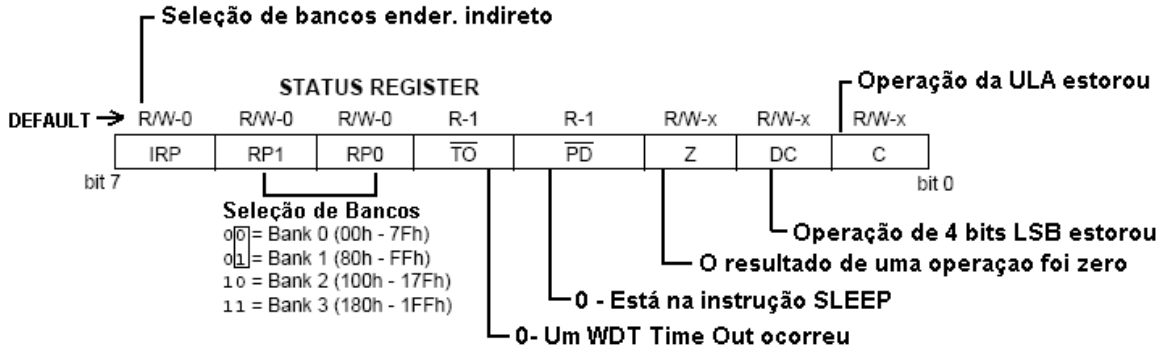

PIC16F628A
CONTROLE SERIAL
BUFFER SERIAL
COMPARADOR ANALÓGICO
OBS: CLOCK INT_RC EM PCON
EEPROM
General Purpose Register
96 Bytes
80 Bytes
Bank 0
Bank 1
PIC16F819
CLOCK INT_RC 8
SSP
CONVERSOR AD
OBS: EEPROM BANK 2 e 3
PIC16F877A
PORT C, D e E
NÃO TEM CLOCK INT_RC



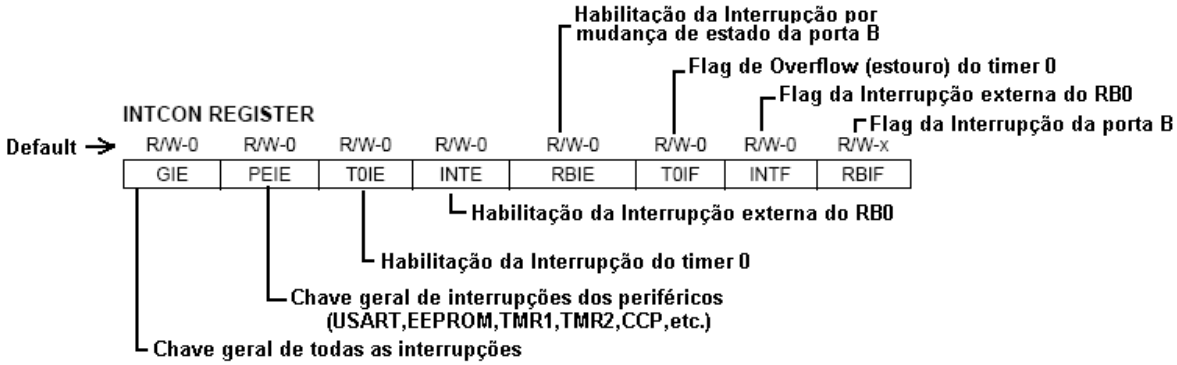
DESCRIÇÃO DOS PRINCIPAIS REGISTROS DE CONTROLE

Para programar os microcontroladores é importante conhecer os registros especiais para conhecer e saber configurar todas as funções dos periféricos intenos. Em relação ao banco de acesso aos registros, pode ser consultado o mapa de registros acima.

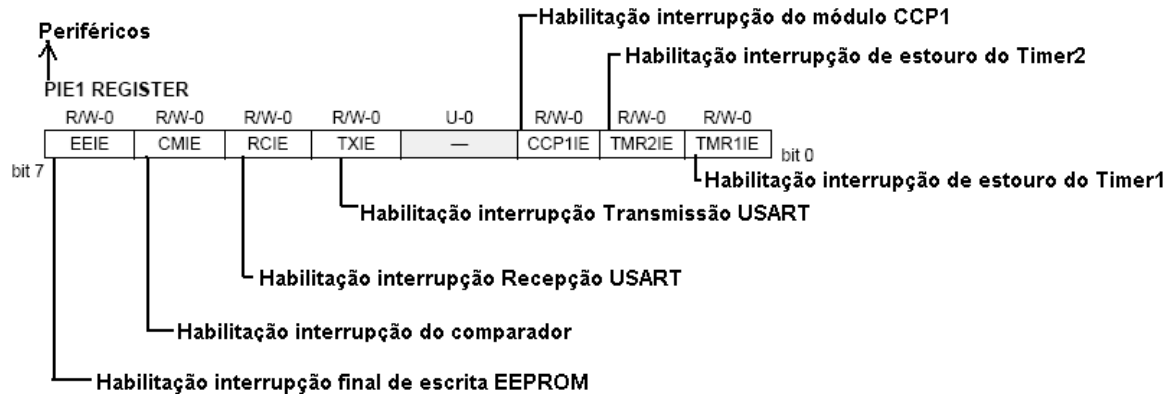
REGISTRO STATUS



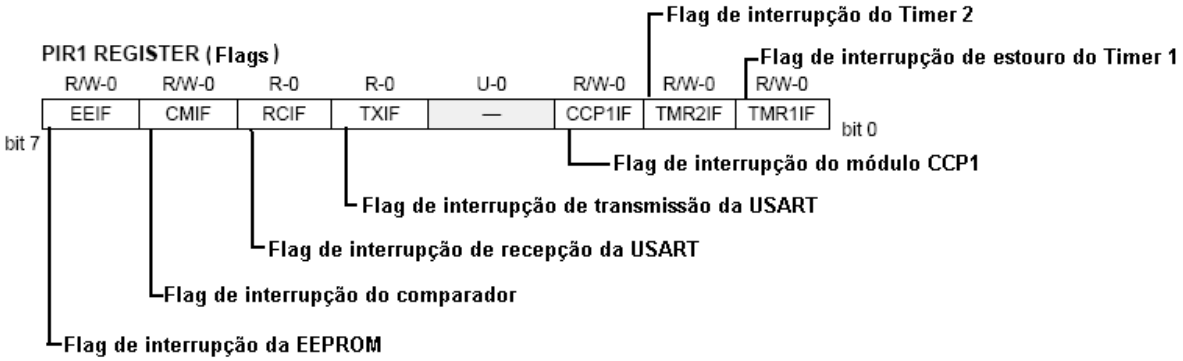
REGISTRO INTCON



REGISTRO DE HABILITAÇÃO DA INTERRUPÇÃO DOS PERIFÉRICOS

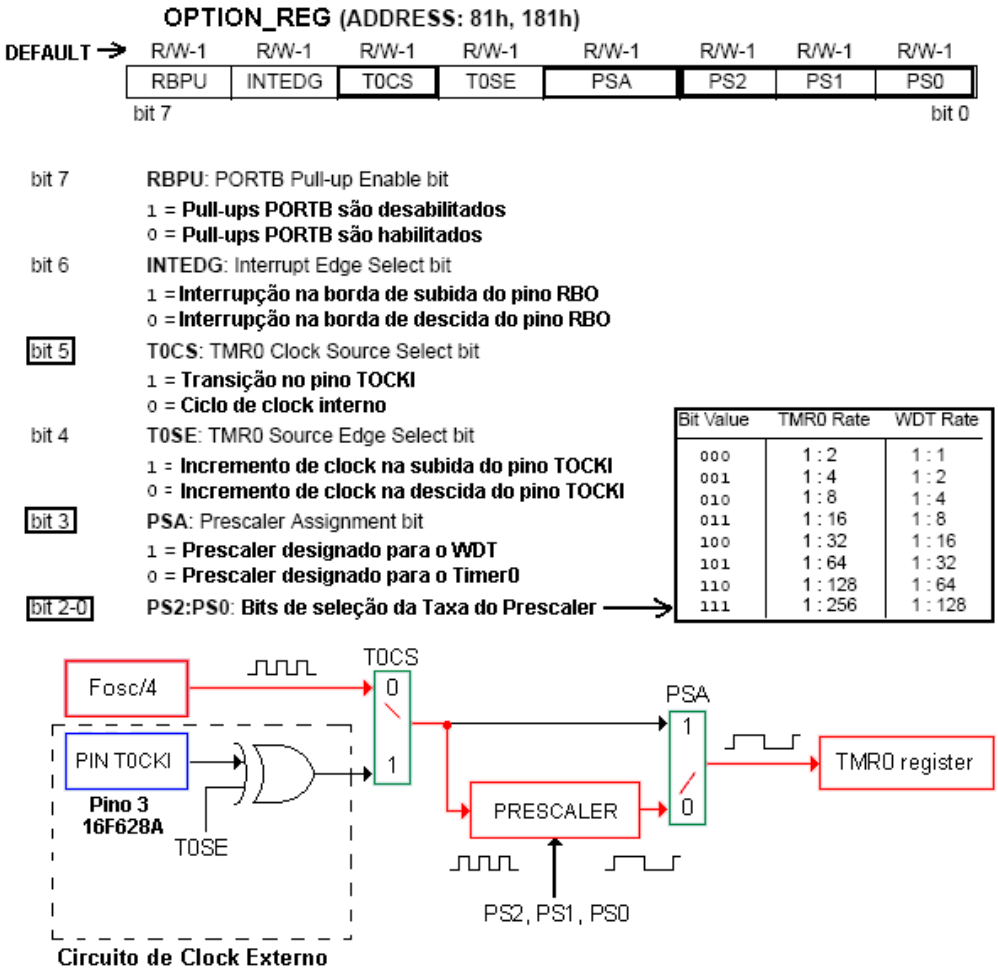


REGISTRO DE FLAGS DE INTERRUPÇÃO DOS PERIFÉRICOS



OPTION_REG E O US0 DO TMR0

A configuração do Timer TMR0 de 8 bits ocupa a maior parte do registro OPTION_REG, como mostra a figura abaixo:



Normalmente, para configurar o TMR0 como temporizador se configura o bit TOCS em "0" para gerar uma freqüência interna do Timer (F_T) igual a $F_{OSC}/4$, ou seja, para um oscilador de 4MHz, a freqüência interna do Timer é de 1MHz, e o bit PSA em "0" para a utilização do p*rescaler* (Divisor de freqüência).

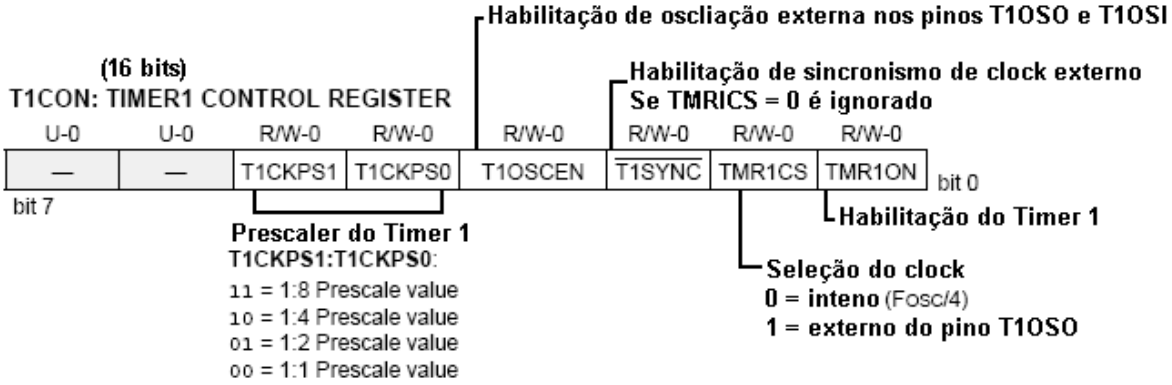
Essa configuração em C é automática mediante alguns comandos como mostra o programa para piscar um led com interrupção do Timer 0:

```
#include <16f628A.h>
#use delay(clock=4000000)
#fuses INTRC_IO, NOMCLR, NOWDT, NOLVP, NOBROWNOUT, PUT, NOPROTECT
// Habilitar 16f628A sempre PUT e BROWNOUT, Desabilitar Reset externo NOMCLR e os outros
#int_timer0
```

```
void trata_t0 ()
{
        boolean led;
        int multiplic;
        // reinicia o timer 0 em 6 mais a contagem que já passou
        set_timer0(6+get_timer0());          //6=256-250
        multiplic++;                         //Incrementa
        if (multiplic == 125)// 250 *125     // Se já ocorreram 125 interrupções:
        {
                multiplic=0;
                led = !led;  // inverte o led
                output_bit (pin_b0,led);
        }
}
main()
{
// configura o timer 0 para clock interno do oscilador e prescaler dividindo por 32
        setup_timer_0 ( RTCC_INTERNAL | RTCC_DIV_32 ); // 250 *125 * 32  = 1 seg
        set_timer0(6); // inicia o timer 0 em 6
        enable_interrupts (global | int_timer0);    // habilita interrupção do timer0

        while (true); // espera interrupção
}
```

Note que a configuração das interrupções em C é automática, ou seja, o compilador gera todo o código do tratamento da interrupção (flags, configuração de registradores, contexto, etc.) e a única tarefa do programador é habiltar a interrupção específica e a construção da função de interrupção após uma diretiva, no exemplo, **#int timer0**.

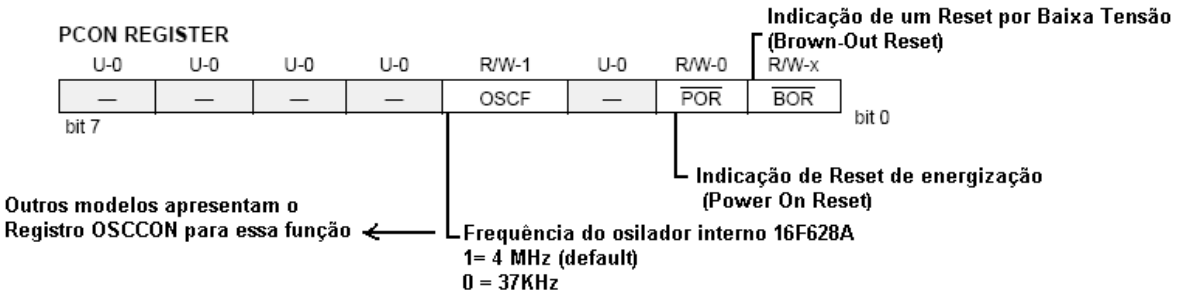
REGISTRO DE CONTROLE DO TIMER 1



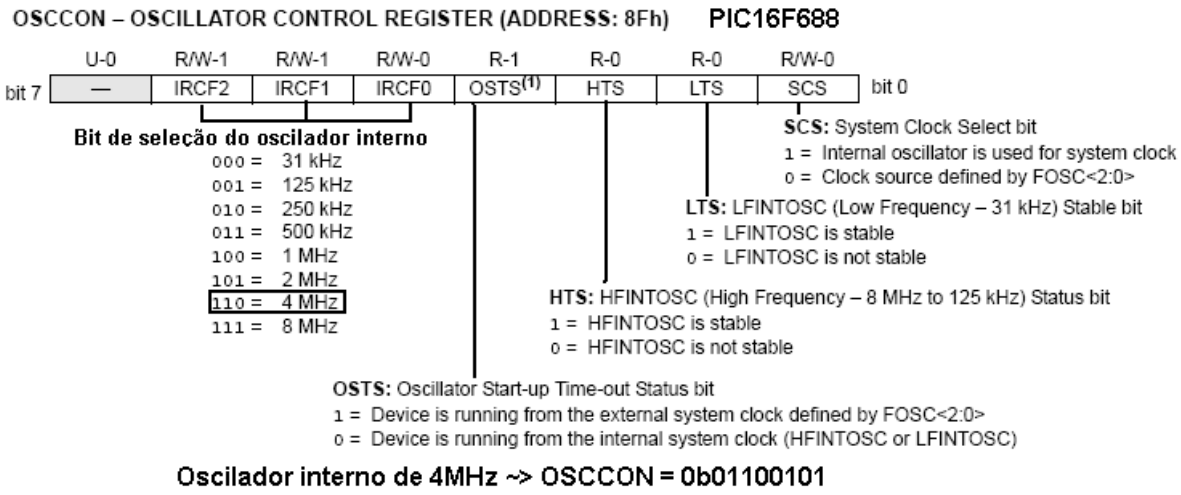
Em C, por interrupção do Timer 1 seria:

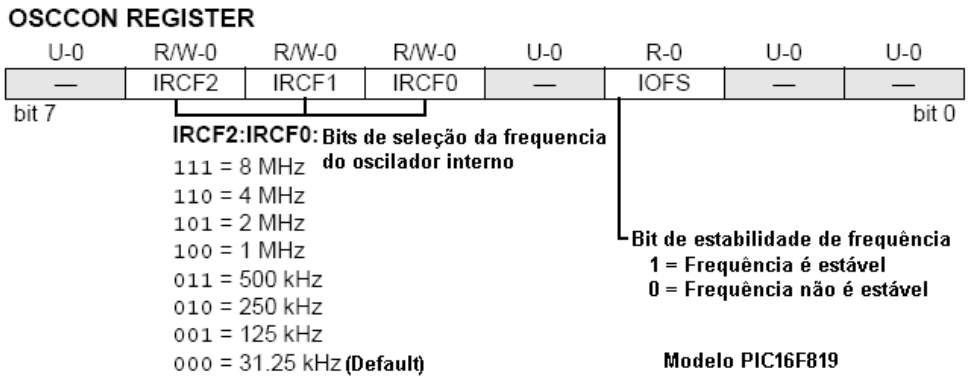
```
#include <16f628A.h>
#use delay(clock=4000000)
#fuses INTRC_IO,PUT,BROWNOUT,NOWDT,NOMCLR,NOPROTECT,NOLVP, // sem cristal
// Habilitar 16f628A sempre PUT e BROWNOUT, Desabilitar Reset externo NOMCLR e os outros

#int_timer1
void trata_t1 ()
{
        boolean led;
        int multiplic;
        // reinicia o timer 1 em 15536 mais a contagem que já passou
        set_timer1(15536 + get_timer1());     //vai contar 50000
        multiplic++;
        // se já ocorreram 2 interrupções
        if (multiplic == 10)                  //vai contar 10 * 50000
        {
                multiplic=0;
                led = !led;  // inverte o led
                output_bit (pin_b0,led);
        }
}
main()
{
        // configura o timer 1 para clock interno e prescaler dividindo por 8
        setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_2 );//vai contar 2 * 10 * 50000 = 1s
        // inicia o timer 1 em 15536
        set_timer1(15536);
        // habilita interrupções
        enable_interrupts (global );
        enable_interrupts (int_timer1);
        while (true); // espera interrupção
}
```

REGISTRO DE CONTROLE DE ENERGIZAÇÃO



REGISTRO CONTROLE DO OSCILADOR





Em C o fusível INTRC_IO seta o bit menos significativo do OSCCON (SCS), e as finições para 4MHz podem ser definidas como #define OSC_4MHZ 0x61, #define OSC_INTRC 1, #define OSC_STATE_STABLE 4.

Caso o oscilador deva ser externo XT, a Microship apresenta uma tabela com algumas faixas de capacitores de filtro usados para cada freqüência, embora o mais comum seja de 15pF.

FAIXAS DE CAPACITORES DE FILTRO
PARA CRISTAIS OSCILADORES
(Testado pela Microship)

Ranges Tested:			
Mode	**Freq**	**C1**	**C2**
XT	455 kHz 2.0 MHz 4.0 MHz	68 - 100 pF 15 - 68 pF 15 - 68 pF	68 - 100 pF 15 - 68 pF 15 - 68 pF
HS	8.0 MHz 16.0 MHz	10 - 68 pF 10 - 22 pF	10 - 68 pF 10 - 22 pF

REGISTROS PORT A, B, C, D e E

Esses registros indicam o valor das portas de entrada e saída do microcontrolador. Em relação ao *pull-up* interno (conexão ao V_{DD} para que o pino possa utilizar o nível lógico alto na saída), a configuração é variável para cada modelo:

- PIC16F628A (18 pinos): Port A (*Tristate*) e Port B (*pull-up* interno configurável por software);

- PIC16F688 (14 pinos): Port C (*Tristate*) e Port A (*pull-up* interno configurável por software exceto A3);

- PIC16F819 (18 pinos): Port A(com *pull-up* de 0 a 4, os demais pinos *Tristate*) e Port B (pull-up interno configurável por software);

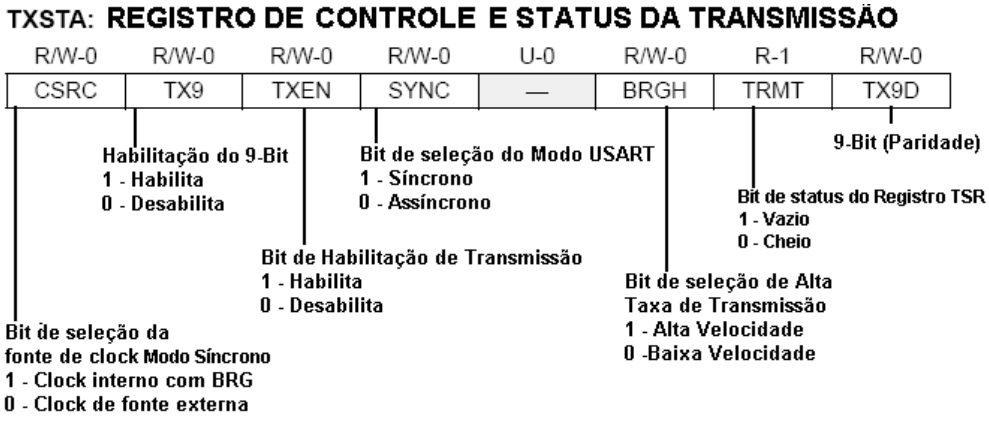
- PIC16F870 (28 pinos) a 877A (40 pinos): Port A (com *pull-up* de 0 a 3, os demais *Tristate*), Port B (*pull-up* interno configurável por software), Port C (com *pull-up*) e PORT D e E (*Tristate*).

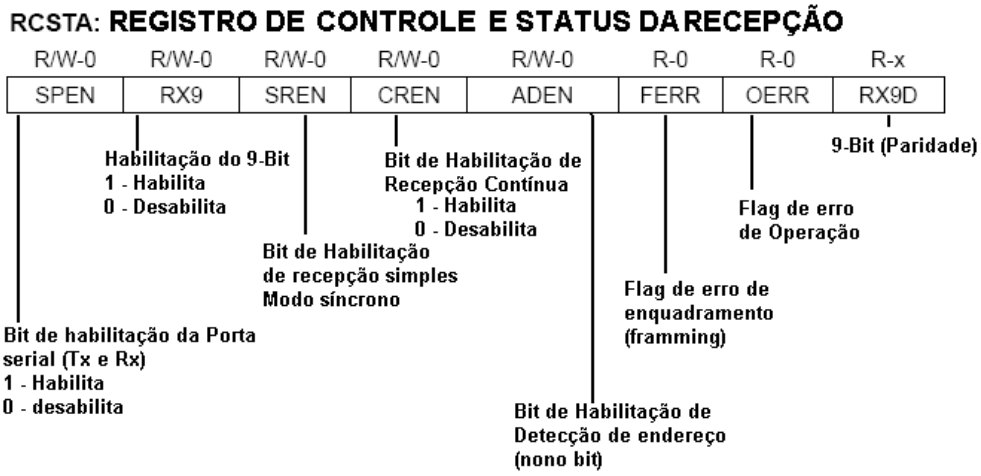
REGISTROS TRIS

Esses registradores servem para configurar os pinos das portas como entrada ou saída. Quando é colocado “1” em um bit do TRIS, o pino relacionado a ele é configurado como entrada. Para configurar o pino como saída, é necessário escrever “0” no bit relacionado. Uma maneira prática para memorizar essa regra é associar “1” ao “I” de Input (entrada) e o “0” ao “O” de Output (saída). Para configurar o PORTA, deve ser utilizado o registrador TRISA, para o PORTB é utilizado o TRISB, etc.

INTERFACE USART

Registros utilizados para controle da comunicação serial síncrona e ***assíncrona*** (mais utilizada).





Gerador de Baud Rate (SPBRG)

FOSC = 4 MHz
Baud Rate **Desejada** = 2400
BRGH = 0 **(Baixa Velocidade)**
SYNC = 0 **(Assíncrona)**

X = **Valor do SPBRG (0 a 255)**

$$\text{Baud Rate Desejada} = \frac{Fosc}{64(x+1)}$$

$$2400 = \frac{4000000}{64(x+1)}$$

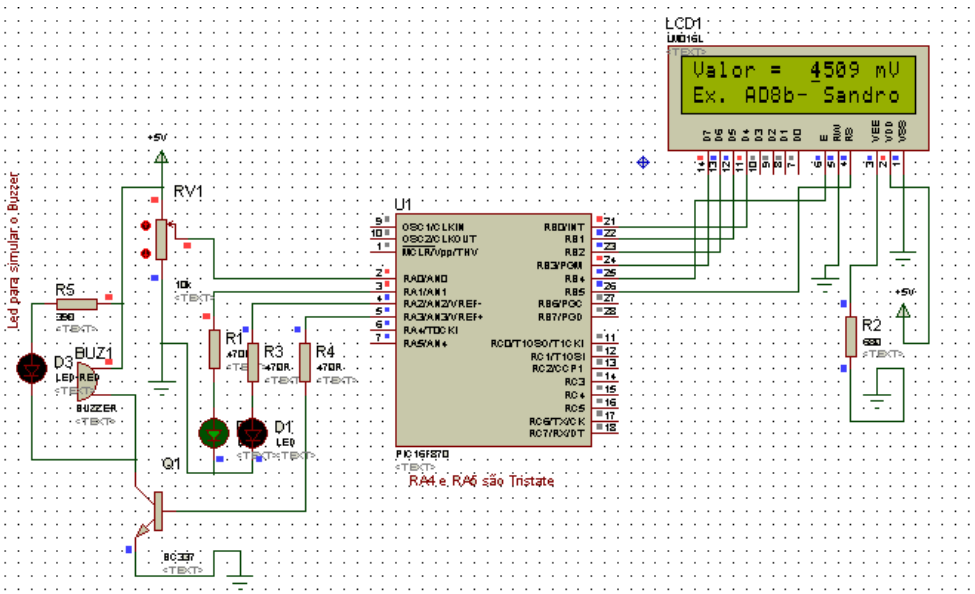
$$x = 25.042$$

$$\text{Baud Rate Calculada} = \frac{4000000}{64(25+1)} = 2403.8$$

$$\text{Erro} = \frac{\text{Baud Rate Calculada} - \text{Baud Rate Desejada}}{\text{Baud Rate Desejada}}$$

$$\text{Erro} = 0.16\%$$

MONITOR DE TENSÃO: Exemplo de aplicação de AD com 8 bits para verificação da tensão de um circuito com alimentação de +5V. Caso a tensão seja maior que 4V, um LED verde é aceso, caso a tensão seja menor que 4V, um LED vermelho é aceso, e se a tensão for menor que 3,5V, um sinalizador de aviso sonoro (buzzer) é ligado.



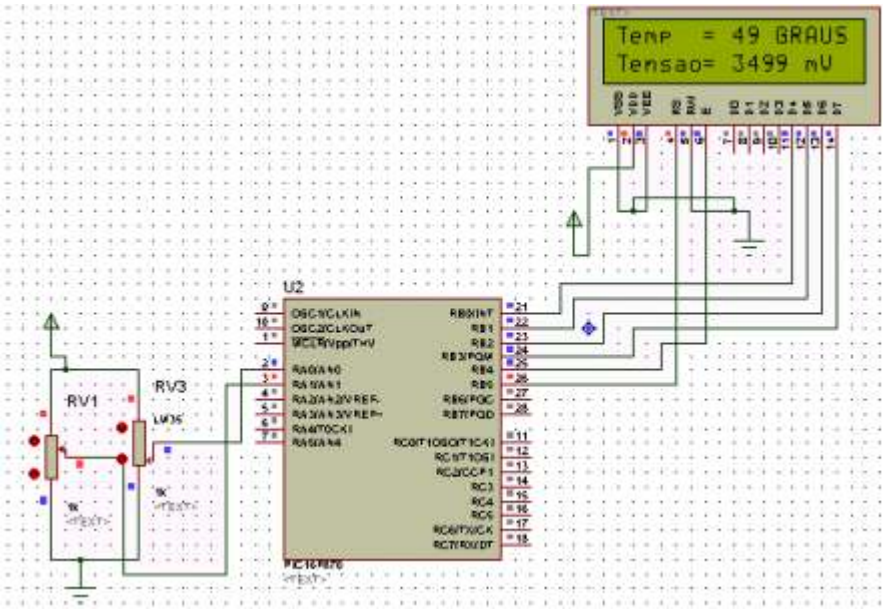

```
#include <16F870.h>
device adc=8   // Resultado do AD em ADRESH, resolução 4 x menor de  ~5mV para ~20mV
#use delay(clock=4000000)   //(Devido à exclusão dos 2 bits de sensibilidade menos significativos)
                            // 00b->0mV  (01b->5mV  10b->10mV   11b->15mV) não visto em 8 bits
#fuses XT,PUT,BROWNOUT,NOWDT,NOPROTECT,NOLVP
// Habilitar 16f870 sempre PUT e BROWNOUT, Desabilitar WDT e os outros, O reset é externo.
#include <MOD_LCD_PORTB.c> // RB0-D4, RB1-D5, RB2-D6, RB3-D7, RB4-E, RB5-RS
#byte port_a = 0x05  // Define a posição 5h da RAM como PORT_A, pois não está na biblioteca CCS
main()       // e nem no compilador. Se a port_a for utilizada o registro TRIS deve ser definido
{
  unsigned int valorlido; //8 bits – de 0 a 255
  int32 valtensao; // valtensao = (5000 * valorlido) / 255 -> ex.: 5000*200=800.000 > 65535 (long int)

          lcd_ini();   // Configuração inicial do LCD
          setup_ADC_ports (RA0_analog);     //(Selecao_dos_pinos_analogicos_da_porta)
          setup_adc(ADC_CLOCK_INTERNAL );   //(Modo_de_funcionamento)
          set_adc_channel(0);     //(Qual_canal_do_AD_interno_vai_converter)
                    lcd_escreve ('\f');  // apaga o display
          printf (lcd_escreve,"Valor = ");
    lcd_pos_xy(1,2);
      printf (lcd_escreve,"Ex. AD8b- Sandro");

  while (true)
          {
    valorlido = read_adc(); // efetua a conversão A/D
                    valtensao = (5000 * (int32)valorlido) / 255;  //ex.: 5000*200=800.000 > 65535 (long int)
    lcd_pos_xy(10,1);   // Coloca na posição de sobrescrever
                    printf (lcd_escreve,"%lu mV  ",valtensao);

 set_tris_A(0b00000001);   //pin_a0 como entrada e demais como saída. 3 formas de setar os pinos:
  if(valtensao>=4000) {          bit_set(port_a,1);                    // T >= 4V
                                      bit_clear(port_a,2);
```

```
                                        bit_clear(port_a,3);}

  if(valtensao<4000 && valtensao>=3500) { port_a=0b00000100;} // bit_set(port_a,2);  3V =< T < 4V

  if(valtensao<3500)    { output_LOW(PIN_A1);              // T < 3V
                          output_HIGH(PIN_A2);
                buzzer:
                output_HIGH(PIN_A3);              //não necessita de set_tris
                delay_ms(50);
                output_LOW(PIN_A3);//Buzzer
                                        }
        delay_ms (50); // aguarda 250 ms
        }
}
```

USO DE DOIS CANAIS SIMULTÂNEOS: Este exemplo utiliza o PIC16F819 com dois canais analógicos, um canal para a leitura da temperatura um sensor LM35 e o outro canal lê a tensão em mV de um potenciômetro. Neste caso, caso o oscilador interno seja utilizado, o registro OSCCON deve ser configurado. O PIC16F819 não existe no PROTEUS. Dessa forma, o conversor AD é simulado com o PIC16F870, considerando as mudanças de fusíveis.

```
#include <16f819.h>
#device ADC=10              // define que o AD utilizado será de 10 bits
#use delay (clock=4000000)    // Lê de 0 a 1023
#fuses PUT,NOWDT,BROWNOUT,NOLVP,NOMCLR,INTRC_IO // sem cristal
// No 16f870 habilitar sempre XT, MCLR, PUT, BROWNOUT, Desabilitar WDT.
#include <MOD_LCD_PORTB.c>
#BYTE OSCCON=0X8F  //Byte de controle do oscilador interno

long int AD(int CANAL)  // declara função (subrotina) usada para ler entrada analógica
  {
  int32 VALOR;                //Declara uma variável de 16 bits para canal de 10 bits
  set_adc_channel(CANAL);       //Congiguração do canal do conversor AD
  delay_us(100);              //Tempo para carregar capacitor canal escolhido
  VALOR = read_adc();         //Faz a leitura e armazena na variável AUXILIAR
  return(VALOR);            //Retorna valor analógico lido
  }

main()
  {
   unsigned int16 valorlido0, valorlido1;       //10 bits
   int32 valtensao0,valtensao1;
OSCCON=0b01100100;   //configuração do oscilador interno para 4 MHz no 16F819 e 16F688
   // valtensao0 = (5000 * valorlido) / 1023 -> ex.: 5000*200=800.000 > 65535 (long int)
  lcd_ini();
  setup_adc_ports(RA0_RA1_RA3_ANALOG);  //Habilita entradas analógicas
  setup_adc(ADC_CLOCK_INTERNAL);              //Configuração do clock do conversor AD

   lcd_escreve ('\f');  // apaga o display
```

```
          printf (lcd_escreve,"Temp  = ");
     lcd_pos_xy(1,2);
       printf (lcd_escreve,"Tensao= ");

 while(1) //Loop infinito
    {
    valorlido0 = AD(0);                  // Chama AD e lê o retorno do AD canal 0.
    valtensao0 = (500 * (int32)valorlido0) / 1023;
    lcd_pos_xy(9,1);// Coloca na posição de sobrescrever
             printf (lcd_escreve,"%lu GRAUS  ", valtensao0);
    delay_ms(500);                                        // para evitar que o LCD fique piscando

     valorlido1 = AD(1);                 // Chama AD e lê o retorno do AD canal 1 = VREF.
     valtensao1 = (5000 * (int32)valorlido1) / 1023;
    lcd_pos_xy(9,2);
    printf(lcd_escreve,"%lu mV   ",valtensao1);         // escreve valor lido no LCD
    delay_ms(500);                                    // para evitar que o LCD fique piscando
    }
 }
```

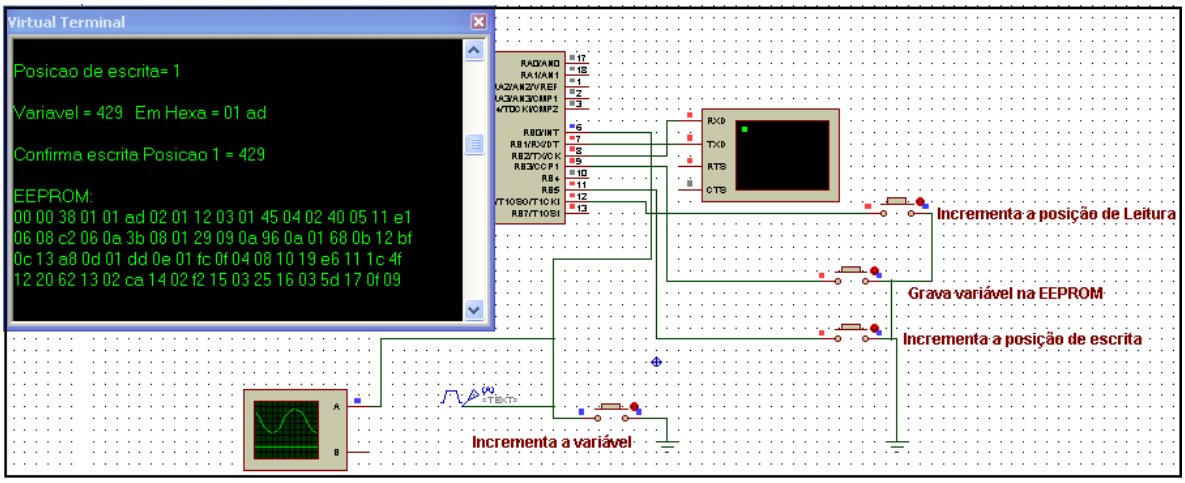


PROGRAMA PARA ESCREVER E LER NA EEPROM INTERNA:

```
#include <16F688.h>
#fuses INTRC_IO,NOWDT,NOPROTECT,NOMCLR
#use delay(clock=4000000)
#use rs232(baud=9600,xmit=PIN_C4,rcv=PIN_C5)
#include <input.c>  //Converte ascii em numero

#BYTE OSCCON=0X8F
```

```
BYTE i, j, endereco, valor;
main() {
OSCCON=0B01100100; // oscilador interno com 4MHz, o bit INTRC foi setado nos fusíveis
while (1) {
 printf("\r\n\nEEPROM:\r\n");   // Imprimi uma matriz de 4 linhas e 16 colunas - 64 bytes
                for(i=0; i<=3; ++i) {
for(j=0; j<=15; ++j) { printf( "%2x ", read_eeprom(i*16+j) );}//Imprime uma linha na EEPROM
                 printf("\n\r");       //posiciona próxima linha
                                }
            printf("\r\nLocal hex para mudar: ");
            endereco = gethex();//Retém um número (byte -dois digitos) em código hexadecimal

            printf("\r\nNovo Valor em hex: ");
            valor = gethex();

            write_eeprom( endereco, valor );
           }
}
```

PROGRAMA QUE GRAVA E LÊ 24 PALAVRAS DE 3 BYTES CADA (UM BYTE PARA ENDEREÇO E DOIS PARA UMA VARIÁVEL DE 16 BITS INCREMENTADA POR INTERRUPÇÃO).



```
#include <16F628a.h>
#fuses INTRC_IO, NOMCLR, NOWDT, NOLVP, NOBROWNOUT, NOPROTECT
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_B2, rcv=PIN_B1) // no 16F688 xmit=PIN_C4,rcv=PIN_C5
#include <input.c>  //Converte ascii em numero decimal
#BYTE OSCCON=0X8F

byte i, j, endereco, valor, valor1, posicao, posleitura;
int16 reg,resultado;

#INT_EXT  // INTERRUPÇÃO EXT DO PINO A2
VOID INT_EXTERNA()
{
delay_ms(30); //Debouncing
reg++;
 while (!input(pin_b0)) { }
 //output_high(pin_b0); Garante a estabilidade do pull_up interno
}

main() {
OSCCON=0B01100100; // oscilador interno com 4MHz, o bit INTRC foi setado nos fusíveis
      enable_interrupts(GLOBAL);     // Possibilita todas interrupcoes
      ENABLE_INTERRUPTS(INT_EXT);  //pin_a2
      EXT_INT_EDGE(H_TO_L);        //borda de descida: bit 6 baixo no option_reg (81h)
port_b_pullups(true);
while (1) {

 printf("\r\n\nEEPROM:\r\n");    // Imprimi uma matriz de 4 linhas e 16 colunas
for(i=0; i<=3; ++i) {
for(j=0; j<=17; ++j) { printf( "%2x ", read_eeprom(i*18+j) );}//Imprime uma linha na EEPROM
                  printf("\n\r");                      //posiciona próxima linha
                                    }
while (input(pin_b3)) //Quando  b3 for aterrado escreve na EEPROM, senão verifica butões
       {

       if (!input(pin_b5)) {++posicao; //Incrementa posicao para escrita
                        if (posicao>=24) {posicao=0;} //Limite do Buffer 72
                        printf ("\r\nPosicao de escrita= %u \r\n",posicao);
                        while (!input(pin_b5));
                         endereco=3*posicao;}

         if (!input(pin_b6)) {++posleitura; //Incrementa posicao para leitura
                        if (posleitura>=24) {posleitura=0;} //Limite do Buffer 72
                        printf ("\r\nPosicao de Leitura = %u \r\n",posleitura);
                        while (!input(pin_b6));
                        endereco=3*posleitura;
                        resultado=(read_eeprom(endereco+1)*256)+read_eeprom(endereco+2);
                        printf ("\r\nLeitura Posicao %u = %lu \r\n",posleitura,resultado);
                         }
       }
           valor1 = reg/256; valor = reg%256;// valor1 é a parte alta do int16 e valor a parte baixa
           printf ("\r\nVariavel = %lu   Em Hexa = %x %x\r\n",reg,valor1,valor);
           write_eeprom( endereco, posicao); //Escreve em hexa
           write_eeprom( endereco+1, valor1 );
```

```
        write_eeprom( endereco+2, valor );
 resultado=(read_eeprom(endereco+1)*256)+read_eeprom(endereco+2);
 printf ("\r\nConfirma escrita Posicao %u = %lu \r\n",posicao,resultado);

  }
}
```

INSTRUÇÕES PARA DIVIDIR UMA VARIÁVEL DE 16 BITS EM DOIS BYTES HEXADECIMAIS PARA SEREM ARMAZENADOS NA EEPROM

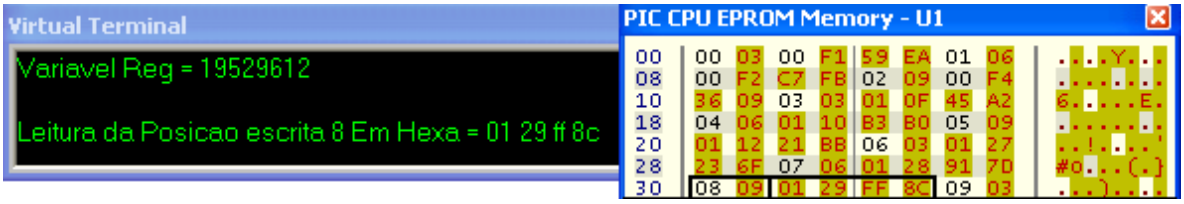
//Variáveis globais: int16 reg,valor1,valor2;

```
        valor1 = reg/256; valor = reg%256;// valor1 é a parte alta do int16 e valor a parte baixa
        printf ("\r\nVariavel = %lu   Em Hexa = %x %x\r\n",reg,valor1,valor);
        write_eeprom( endereco, valor1 );
        write_eeprom( endereco+1, valor );
```

INSTRUÇÕES PARA DIVIDIR UMA VARIÁVEL DE 32 BITS EM QUATRO BYTES HEXADECIMAIS PARA SEREM ARMAZENADOS NA EEPROM

//Variáveis globais: int32 reg,res1,res2,resp1,resp2,resp3;
//BYTE valorb1,valorb2,valorb3,valorb4;

```
res1 = reg/256; valorb1 = reg%256;
res2 = res1/256; valorb2= res1%256;
valorb4 = res2/256; valorb3= res2%256;
printf ("\r\nVariavel Reg = %lu \r\n",reg);
write_eeprom( endprom, posicao); write_eeprom( endprom+1, segundo );
write_eeprom( endprom+2, valorb4); write_eeprom( endprom+3, valorb3 );
write_eeprom( endprom+4, valorb2); write_eeprom( endprom+5, valorb1 );
printf ("\r\nLeitura da Posicao escrita %u Em Hexa = %x %x %x %x\r\n",posicao,valorb4,valorb3,valorb2,valorb1);

 resp1=(256*read_eeprom(endprom+4))+read_eeprom(endprom+5);
 resp2=(256*256*read_eeprom(endprom+3))+resp1;
 resp3=(256*256*256*read_eeprom(endprom+2))+resp2;
 printf ("\r\Calculo dos 4 Bytes em decimal = %lu \r\n",resp3);  //resp3 será igual a reg
```



REFERÊNCIAS BIBLIOGRÁFICAS:

[1] Data Sheet do microcontrolador PIC18FXX50 disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/39632b.pdf. Acessado em 08/06/2009

[2] Microchip Technology Inc, disponível em http://www.microchip.com. Acessado em 08/06/2009.

[3] STALLINGS, William. Arquitetura e Organização de Computadores 5a Edição. São Paulo: Prentice Hall, 2002.

[4] USB.Org, disponível em http://www.usb.org/. Acessado em 08/06/2009.

[5] Ibrahim, Dagan. Advanced PIC microcontroller Projects in C: from USB to RTOS with the PIC18F series, Newnes, 2008.

[6] CCS, Inc. – Home, disponível em http://www.ccsinfo.com/. Acessado em 08/06/2009.

[7] Serial Port Terminal – Test and debug drial port devices with advanced serial port terminal, disponível em http://www.eltima.com/products/serial-port-terminal. Acessado em 08/06/2009.

[8] PEREIRA, Fábio. Microcontroladores PIC: programação em C. 1. Ed. São Paulo: Érica, 2003. 358p.

APÊNDICE I: AMBIENTES DE INTERFACE DELPHI E C++BUILDER

O Delphi e o BCB (C++Builder), desenvolvidos pela *Borland,* são ambientes de desenvolvimento de aplicações orientados a objeto, ou seja, são ambientes gráficos programados de forma dedicada e que podem ser utilizados em aplicações de comunicação entre microcontroladores e computadores pessoais como sistemas supervisórios. O Delphi e o BCB (C++Builder) são praticamente idênticos apresentando a mesma interface de tela, os mesmos objetos, as mesmas funções e, até mesmo, os mesmos exemplos de projetos. A única diferença é que a linguagem de origem do Delphi é o Object Pascal (código fonte em Pascal) e a linguagem de origem do BCB é o C++ (código fonte em C). O Delphi surgiu primeiro, por isso é mais

comercial, mas como os programadores de C++ mostraram a necessidade de um ambiente similar orientado a objeto, surgiu o BCB.

Os ambientes foram desenvolvidos para que os programadores desenvolvam os projetos sem digitar, em alguns casos, ou digitando o mínimo possível linhas de comando do código fonte, pois os objetos selecionados pelo mouse geram as linhas de comando no editor de código.

A similaridade entre os ambientes Delphi e BCB permite que o projetista de um ambiente possa projetar também no outro respeitando as principais diferenças das linguagens de código fonte Pascal e C, estre elas:

- Blocos de comando: em Pascal é delimitado pelas palavras ***begin*** e ***end***, em C pelos caracteres { };

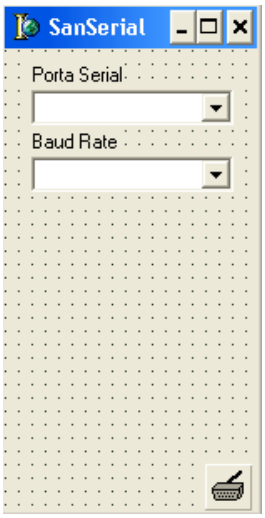
- O operador de atribuição numérica: em Pascal é:= e em C apenas = ;
- O operador de relação de igualdade: em Pascal é = e em C = = ;
- Em Pascal existem as funções (*funtion*) que retornam valores e os procedimentos (*procedures*) que são funções que não retornam valores, enquanto em C, as funções não recebem essas identificações e podem ou não retornar valores, dependendo do comando return( );
- A divisão de classes e subclasses de um objeto: em Pascal é somente um ponto (.) em C é um traço mais a indicação de maior (->).

COMO CRIAR UMA COMUNICAÇÃO BIDIRECIONAL ENTRE O AMBIENTE DELPHI E O PIC SANUSB USANDO O CPORT

Para estabelecer a comunicação, basta gravar qualquer programa da pasta exemplos no SanUSB que emule a comunicação serial virtual. Após instalar o driver CDC no Windows, siga os passos abaixo.

1 – Vá na paleta de componentes no CportLib e insira na área de trabalho, o Comport e 2 ComCombobox, um para a porta serial e o outro para o Baud Rate;

2- Nomeie, inserindo dois *Label* do componente Standard, o ComCombobox superior como Porta serial e o inferior como *Baud Rate*;



2- Clique no ComCombobox superior, vá em properties e altere AutoApply para true, comport: comport1 e ComProperty: CpPort; clique no ComCombobox inferior, vá em properties e altere AutoApply para true, comport: comport1 e ComProperty: CpBaudRate;

3- Execute (F9) o projeto para ver se o executável é gerado; Para voltar ao projeto (Alt+F4);

4- Insira um botão do componente Standard para conectar a porta serial, Vá em caption e o chame de Conecta&r, onde o '&' permite habilitar o botão com Alt+r, vá em name e o chame de Bconectar; Clique duas vezes no botão Bconectar e escreva as condições abaixo para abrir e fechar a porta serial, mostrando Conectar e Desconectar. Lembre que após digitar o ponto, o Delphi mostra automaticamente as propriedades de cada objeto (como Caption).

```
if Bconectar.Caption='Conecta&r'
then  begin
ComPort1.Connected:=true;
Bconectar.Caption:='Desconecta&r'
end
else begin
ComPort1.Connected:=false;
Bconectar.Caption:='Conecta&r'
end
```

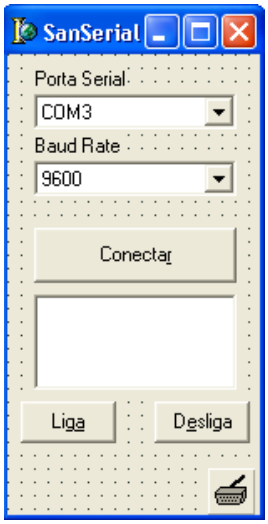
4- Insira dois botões do componente Standard para ligar e desligar um Led, Vá em caption e chame-os de Liga e Desliga; Clique duas vezes no botão Liga e escreva a String para ligar o LED:

```
if Comport1.Connected = true then
  begin
    Comport1.WriteStr('L1');
  end
       else
       begin
             ShowMessage('Conecte a porta serial!!');
       End
```

Clique duas vezes no botão Desliga e escreva a String para desligar o LED:

```
if Comport1.Connected = true then
  begin
    Comport1.WriteStr('D1');
  end
       else
       begin
             ShowMessage('Conecte a porta serial!!');
       end
```

5- Para mostrar os bytes recebidos insira um Memo do componente Standard e configure name para MemoRx, vá na propriedade Line e apague qualquer texto Default dentro do MemoRx.



A recepção é um evento de interrupção (Events no Object Inspector) do componente Cport, clique no Cport vá nas propriedades, selecione a COM disponível em Port, vá em Events e clique duas vezes em OnRxChar, selecionando ComPort1RxChar. Irá aperecer no script:

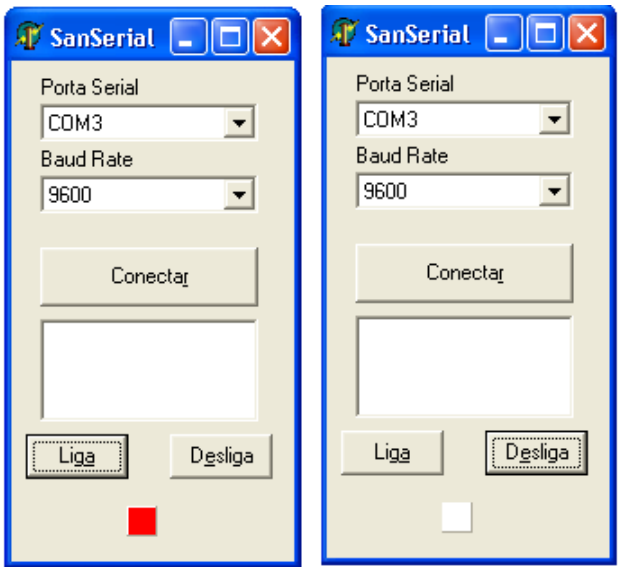
```
procedure TForm1.ComPort1RxChar(Sender: TObject; Count: Integer);
```

Então escreva:

```
var recebe:string;
begin
//Escreva a expressão de recepção serial abaixo e declare a variável  recebe como string, veja acima de begin:
Comport1.ReadStr(recebe,count);

  if recebe = 'L1' then
  begin
  MemoRx.Lines.Add('Chegou ' + recebe); //Escreve dentro do Memo
  PLedOn.Visible:=true;
  PLedOff.Visible:=false;
  end;
    if recebe = 'D1' then
```

```
begin
  MemoRx.Lines.Add('Chegou ' + recebe);
  PLedOn.Visible:=false;
  PLedOff.Visible:=true;
end;
```

6- Insira dois panel para representar um Led, o primeiro apague caption, cor branca e nome PledOff e o outro por trás apague caption, cor vermelha e nome PledOn e Visible:=false. One o Led irá ascender (vermelho) quando o botão Liga for pressionando e apagar (branco) quando o botão Desliga for pressionando.



É importante salientar que pela USB as strings (L1, D1, etc.) emuladas são enviadas do PIC para o supervisório através bytes individuais, então existem duas saídas para compreensão do supervisório:

- comandos enviados pelo microcontrolador com apenas um byte e não um conjunto de bytes, por exemplo, L ao invés de L1 e M ao invés de L2.

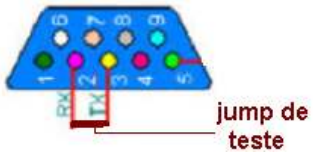
Exemplo:

```
 if recebe='L' then begin  // Correto!
  PLedOn.Visible:=true;
  PLedOff.Visible:=false;
  end;
```

- inserir na recepção serial uma variával global *recebe* (deve ser declarada antes de implementation) do tipo string para receber os caracteres individuais emulados pela USB e construir uma string dentro do evento de recepção serial do ComPort. (recebe:=recebe+receb1;)

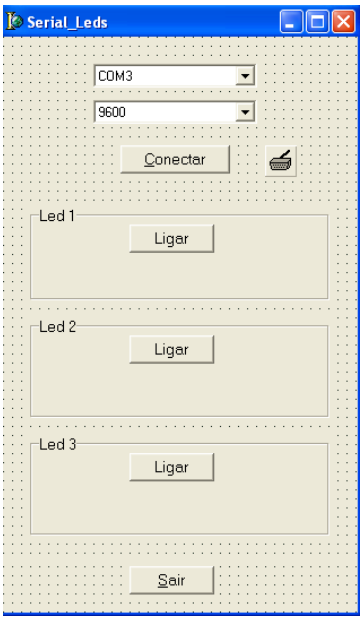
```
var
  Serial_Leds: TSerial_Leds;
  recebe:string;        //Variável Global
implementation
.....
......
procedure TSerial_Leds.ComPort1RxChar(Sender: TObject; Count: Integer);
var  receb1:string;

begin
ComPort1.ReadStr(receb1,count);
recebe:=recebe+receb1;              //Artifício usado para que a variável global
                                    //recebe os caracteres individuais emulados pela USB
  if recebe = 'L1' then
  begin
    Label1.Caption:= 'Led ligado';
    Button2.Caption:= 'Desligar';
    recebe:='';                     //Limpa a variável global recebe
  end;
```

É importante salientar que para testar a comunicação bidirecional do executável sem o SanUSB, basta colocar um jump unindo os pinos 2 (Rx) e 3 (Tx) de uma porta serial RS-232 real.



SUPERVISÓRIO EM DELPHI PARA CONTROLAR TRÊS LEDS

Neste exemplo, o supervisório envia L1 depois que o botão *ligar Led 1* é pressionado, o *label* do botão muda para Desliga. Quando o botão Des*ligar Led 1* é pressionado, o supervisório envia D1. O microcontrolador atua o led e depois confirma para o supervisório enviando L1 ou D1 que atualiza o label indicando Led 1 ligado ou desligado.



Esta interface pode ser construída de forma similar ao exemplo anterior, inserindo na recepção serial uma variával global *recebe* do tipo string para receber os caracteres individuais emulados pela USB, no evento de recepção serial do ComPort. Ela deve ser declarada antes de implementation:

```
var
  Serial_Leds: TSerial_Leds;
  recebe:string;        //Variável Global
implementation
```

```
.....
......
procedure TSerial_Leds.ComPort1RxChar(Sender: TObject; Count: Integer);
var  receb1:string;

begin
ComPort1.ReadStr(receb1,count);
recebe:=recebe+receb1;              //Artifício usado para que a variável global
                                    //recebe os caracteres individuais emulados pela USB
  if recebe = 'L1' then
  begin
    Label1.Caption:= 'Led ligado';
    Button2.Caption:= 'Desligar';
    recebe:='';                     //Limpa a variável global recebe
  end;
```

PROGRAMA DO MICROCONTROLADOR

```
#include <SanUSB.h>
#include <usb_san_cdc.h>

char dado1,dado2;

main() {

usb_cdc_init(); // Inicializa o protocolo CDC
usb_init(); // Inicializa o protocolo USB
while(!usb_cdc_connected()) {} // espere enquanto o protocolo CDC não se conecta com o driver CDC
usb_task(); // Une o periférico com a usb do PC
usb_wait_for_enumeration(); //espera até que a USB do Pic seja reconhecida pelo PC

port_b_pullups(true);
output_low(pin_b0);
output_low(pin_b6);
output_low(pin_b7);


while(1)
{
  if (usb_cdc_kbhit(1)) //se o endpoint de contem dados do PC
  {

    dado1=usb_cdc_getc();
    if (dado1=='L')
    {
      dado2=usb_cdc_getc();
      if (dado2=='1')
      {
        output_high(pin_b0);
        printf(usb_cdc_putc,"L1");
      }
       if (dado2=='2')
      {
        output_high(pin_b6);
        printf(usb_cdc_putc,"L2");
```

```
      }
       if (dado2=='3')
       {
         output_high(pin_b7);
         printf(usb_cdc_putc,"L3");
       }
    }
    if (dado1=='D')
    {
      dado2=usb_cdc_getc();
      if (dado2=='1')
      {
        output_low(pin_b0);
        printf(usb_cdc_putc,"D1");
      }
       if (dado2=='2')
      {
        output_low(pin_b6);
        printf(usb_cdc_putc,"D2");
      }
       if (dado2=='3')
      {
        output_low(pin_b7);
        printf(usb_cdc_putc,"D3");
      }
    }
  }
}}
```

APÊNDICE II:

UTILIZANDO O SanUSB COM O C18 E O IDE MPLAB (Ambiente de Desenvolvimento Integrado)

1) Instale o MPLAB e o compilador C18. É possível baixar gratuitamente o MPLAB e a versão estudante do compilador C18 no site da Microchip (www.microchip.com).

2) Para compilar os programas com o C18 e o SanUSB, crie uma pasta, como por exemplo, C:\InicioSanC18 e insira nessa pasta programas, como o exemplo pisca.c, e os arquivos c018i.o, clib.lib e p18f2550.lib e san18f2550.lkr e a biblioteca cabeçalho SanUSB.h e SanUSBint.h (com interrupção). Todos esses programas estão na pasta do arquivo InicioSanC18 do Grupo SanUSB (http://br.groups.yahoo.com/group/GrupoSanUSB/).

3) Para possibilitar a compilação, é necessário apontar para as bibliotecas contidas na pasta InicioSanC18 e em mcc18\h. Por isso, abra o MPLAB, vá em em Project -> Build Options -> **Include Search Path** -> e insira InicioSanC18 (a pasta do projeto) e também C:\mcc18\h (a pasta de bibliotecas padrão)

4) No MPLAB, vá em Project -> Project Wizard -> Avançar. No *step one* insira o Device PIC18F2550.

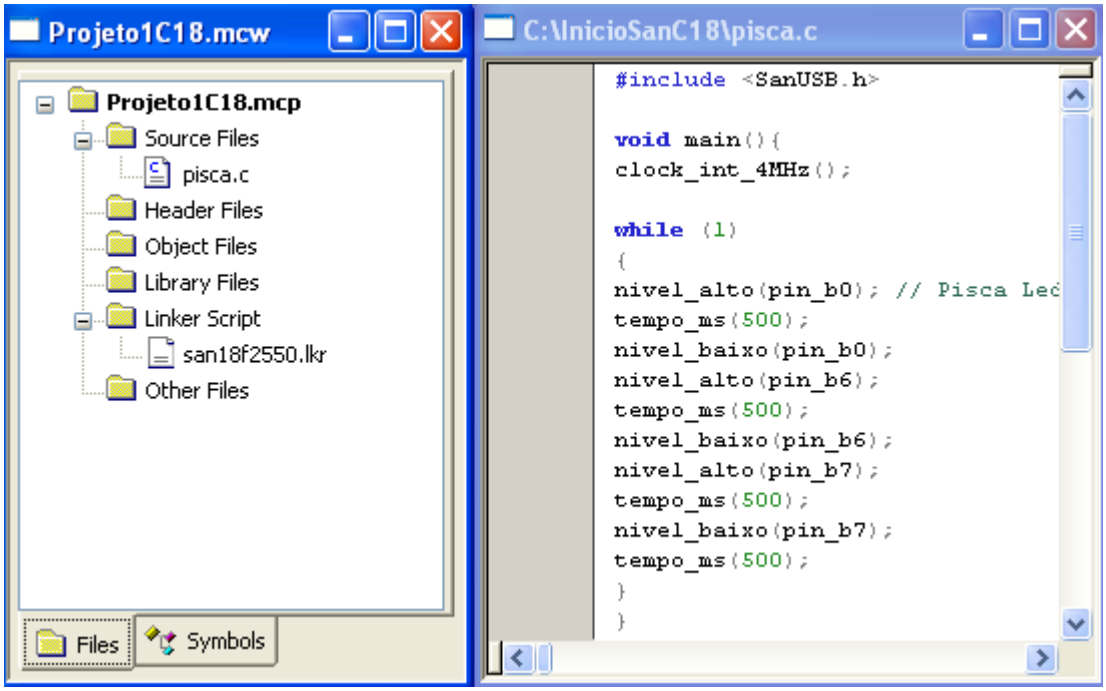
5) No *step two*, selecione a ferramenta *Active Toolsuite* como ***Microchip C18 Tollsuite*** e aponte o endereço de cada um de seus executáveis (*contents*) do C18 e avance:

Toolsuite contents:	**Location:**
MPASMWIN.exe	C:\mcc18\mpasm\MPASMWIN.exe
mcc18.exe	C:\mcc18\bin\mcc18.exe
MPLink.exe	C:\mcc18\bin\MPLink.exe
MPLib.exe	C:\mcc18\bin\MPLib.exe

6) No *step three* indique o nome do projeto e a pasta (Browse..) onde ele será criado, por exemplo, C:\InicioSanC18\Projeto1C18.

7) Após avançar, adicione no *step four* o arquivo pisca.c e o *linkador* san18f2550.lkr e clique em avançar e concluir. Esses arquivos irão aparecer no *Workspace* (espaço de trabalho) do IDE MPLAB. Se não aparecer basta clicar em *View -> Project*. É possível também adicionar esses arquivos clicando com o botão direito sobre a pasta *Source Files* no *Workspace* e adicionar o programa pisca.c e adicionar também na pasta *Linker* do *Workspace* o arquivo san18f2550.lkr que estão dentro da pasta do projeto.

É possível visualizar o programa com um duplo clique sobre pisca.c. Para compilar pressione F10. Para compilar outros programas basta criá-los com a extensão .c dentro da mesma pasta do projeto e inserir no *Souce Files* do *Wokspace*, removendo o anterior. Informações detalhadas sobre esse compilador podem ser encontradas na pasta instalada C:\MCC18\doc.



Funções SanUSB

Este capítulo descreve todas as funções em português da biblioteca SanUSB no C18. É importante salientar que além dessas funções, são válidas as funções padrões ANSI C e também as funções do compilador C18 detalhadas na pasta instalada C:\MCC18\doc. A fim de facilitar o entendimento, as funções SanUSB foram divididas em oito grupos, definidos por sua utilização e os periféricos do hardware que estão relacionadas.

Funções básicas da aplicação do usuário

Este grupo de funções define a estrutura do programa uma vez que o usuário deve escrever o programa.c de sua aplicação inserindo o arquivo #include<SanUSB.h>.

O microcontrolador possui um recurso chamado *watchdog timer* (wdt) que nada mais é do que um temporizador cão-de-guarda contra travamento do programa. Caso seja habilitado habilita_wdt() na função principal *main(),* este temporizador está configurado para contar aproximadamente um intervalo de tempo de 16 segundos. Ao final deste intervalo, se a *flag* limpa_wdt() não for zerada, ele provoca um reset do microcontrolador e conseqüentemente a reinicialização do programa. A aplicação deve permanentemente zerar a *flag* limpa_wdt() dentro do laço infinito (while(1)) na função principal main() em intervalos de no máximo 16 segundos. Este recurso é uma segurança contra qualquer possível falha que venha travar o programa e paralisar a aplicação. Para zerar o wdt, o usuário pode também utilizar a função ClrWdt() do compilador C18.

A seguir estão as características detalhadas destas funções.

clock_int_4MHz()

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

```
#include<SanUSB.h>

        void main (void) {
        clock_int_4MHz();
```

nivel_alto()

Função: Força nível lógico alto (+5V) em uma saída digital.
Argumentos de entrada: Nome da saída digital que irá para nível lógico alto. Este nome é construído pelo inicio pin_ seguido da letra da porta e do número do pino. Pode ser colocado

também o nome de toda a porta, como por exemplo, portb.
Argumentos de saída: Não há.
Observações: Não há.
Exemplo:

```
nivel_alto(pin_b7); //Força nível lógico 1 na saída do pino B7
nivel_alto(portb); //Força nível lógico 1 em toda porta b
```

nivel_baixo()

Função: Força nível lógico baixo (0V) em uma saída digital.
Argumentos de entrada: Nome da saída digital que irá para nível lógico baixo. Este nome é construído pelo inicio pin_ seguido da letra da porta e do número do pino. Pode ser colocado também o nome de toda a porta, como por exemplo, portc.

Argumentos de saída: Não há.
Observações: Não há.
Exemplo:

```
nivel_baixo(pin_b7); //Força nível lógico 0 na saída do pino B7
nivel_baixo(portc); //Força nível lógico 0 em toda porta c
```

saída_pino(pino,booleano)

Função: Acende um dos leds da placa McData.
Argumentos de entrada: Pino que irá receber na saída o valor booleano, valor booleano 0 ou 1.
Argumentos de saída: Não há.
Observações: Não há.
Exemplo:

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

tempo_us()

Função: Tempo em múltiplos de 1us.
Argumentos de entrada: Tempo de delay 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);        //Delay de 200 us
```

tempo_ms()

Função: Tempo em múltiplos de 1 ms.
Argumentos de entrada: Tempo de delay 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);        //Delay de 500 ms
```

entrada_pin_xx

Função: Lê nível lógico de entrada digital de um pino.
Argumentos de entrada: Não há..
Observações: Este nome é construído pelo 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:

```
#pragma interrupt interrupcao
void interrupcao()
{
        if (ext1_interrompeu)     {       //espera a interrupção externa 1 (em B1)
        ext1_interrompeu = 0;             //limpa a flag de interrupção
        PORTBbits.RB0 =! PORTBbits.RB0;} //inverte o LED em B0

        if (timer0_interrompeu)       {       //espera o estouro do timer0
        timer0_interrompeu = 0;             //limpa a flag de interrupção
        PORTBbits.RB0 =! PORTBbits.RB0;    //inverte o LED em B7
```

```
tempo_timer16bits(0,62500); } }
```

liga_timer16bits(timer,multiplicador)

Função: Liga os timers e ajusta o multiplicador de tempo na função *main()*. Argumentos de entrada: Timer de 16 bits (0,1ou 3) e multiplica que é o valor do prescaler para multiplicar o tempo. Argumentos de saída: Não há. Observações: O timer 0 pode ser multiplicado por 2, 4, 6, 8, 16, 32, 64, 128 ou 256. O Timer 1 e o Timer 3 podem ser multiplicados por 1, 2, 4 ou 8. Exemplo:

```
liga_timer16bits(0,16); //Liga timer 0 e multiplicador de tempo igual a 16
liga_timer16bits(3,8);  //Liga timer 0 e multiplicador de tempo igual a 8
```

tempo_timer16bits(timer,conta_us)

Função: Define o timer e o tempo que será contado em us até estourar. Argumentos de entrada: Timer de 16 bits (0,1ou 3) e tempo que será contado em us (valor máximo 65536). Argumentos de saída: Não há. Observações: O Não há. Exemplo:

```
habilita_interrupcao(timer0);
liga_timer16bits(0,16);         //liga timer0 - 16 bits com multiplicador (prescaler) 16
tempo_timer16bits(0,62500);     //Timer 0 estoura a cada 16 x 62500us = 1 seg.
```

habilita_wdt()

Função: Habilita o temporizador cão-de-guarda contra travamento do programa. Argumentos de entrada: Não há. Argumentos de saída: Não há. Observações: O *wdt* inicia como padrão sempre desabilitado. Caso seja habilitado na função principal *main(),* este temporizador está configurado para contar aproximadamente um intervalo de tempo de 16 segundos. Ao final deste intervalo, se a *flag* limpa_wdt() não for zerada, ele provoca um reset do microcontrolador e conseqüentemente a reinicialização do programa. Exemplo:

```
#include<SanUSB.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<SanUSB.h>

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

escreve_eeprom(posição,valor)

Função: Escrita de um byte da memória EEPROM interna de 256 bytes do microcontrolador.
Argumentos de entrada: Endereço da memória entre 0 a 255 e o valor entra 0 a 255.
Argumentos de saída: Não há.
Observações: O resultado da leitura é armazenado no byte EEDATA.
Exemplo:

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

le_eeprom()

Função: Leitura de um byte da memória EEPROM interna de 256 bytes do microcontrolador.
Argumentos de entrada: Endereço da memória entre 0 a 255.
Argumentos de saída: Não há.
Observações: O resultado da leitura é armazenado no byte EEDATA.
Exemplo:

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

Funções do Conversor Analógico Digital (A/D)

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

habilita_canal_AD()

Função: Habilita entradas analógicas para conversão AD.
Argumentos de entrada: Número do canal analógico que irá ser lido. Este dado habilita um ou vários canais AD e pode ser AN0, AN0_a_AN1 , AN0_a_AN2 , AN0_a_AN3, AN0_a_AN4,

AN0_a_AN8, AN0_a_AN9, AN0_a_AN10, AN0_a_AN11, ou AN0_a_AN12.
Argumentos de saída: Não há.
Observações: Não há.
Exemplo:

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

le_AD8bits()

Função: Leitura de uma entrada analógica com 8 bits de resolução.
Prototipagem: unsigned char analog_in_8bits(unsigned char).
Argumentos de entrada: Número do canal analógico que irá ser lido. Este número pode ser 0, 1 , 2 , 3, 4, 8, 9, 10, 11 ou 12.
Argumentos de saída: Retorna o valor da conversão A/D da entrada analógica com resolução de 8 bits.
Observações: Não há.
Exemplo:

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

le_AD10bits()

Função: Leitura de uma entrada analógica com 8 bits de resolução.
Prototipagem: unsigned char analog_in_8bits(unsigned char).
Argumentos de entrada: Número do canal analógico que irá ser lido. Este número pode ser 0, 1 , 2 , 3, 4, 8, 9, 10, 11 ou 12.
Argumentos de saída: Retorna o valor da conversão A/D da entrada analógica com resolução de 10 bits.
Observações: Não há.
Exemplo:

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

Funções da comunicação serial RS-232

As funções a seguir são utilizadas na comunicação serial padrão RS-232 para enviar e receber dados, definir a velocidade da comunicação com o oscilador interno 4MHz. As configurações da comunicação são: sem paridade, 8 bits de dados e 1 stop bit. Esta configuração é denominada 8N1 e não pode ser alterada pelo usuário.

taxa_rs232();

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_rs232 e escreve_rs232. As taxas programáveis são 1200 bps, 2400 bps, 9600 bps, 19200 bps. Exemplo:

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

le_rs232();

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_rs232();
        if (c >= '0' && c <= '9')      { c -= '0';    PORTB = c;}  }}
```

escreve_rs232();

Função: Transmite um byte pela RS-232. Argumentos de entrada: O dado a ser transmitido deve ser de 8 bits do tipo char. Argumentos de saída: Não há. Observações: A função escreve_rs232 não aguarda o fim da transmissão do byte. Como não existe um buffer de transmissão o usuário deve garantir a transmissão com a função envia_byte( ) para enviar o próximo byte. Exemplo:

```
escreve_rs232(PORTC); //escreve o valor da porta C
while (envia_byte());
escreve_rs232('A'); // escreve A
while (envia_byte());
```

Programas:

//1- Programa para piscar 3 leds na porta B
#include <SanUSB.h>

```
void main(){
clock_int_4MHz();

while (1)
{
nivel_alto(pin_b0); // Pisca Led na função principal
tempo_ms(500);
nivel_baixo(pin_b0);
nivel_alto(pin_b6);
tempo_ms(500);
nivel_baixo(pin_b6);
nivel_alto(pin_b7);
tempo_ms(500);
nivel_baixo(pin_b7);
tempo_ms(500);
}
}
```

//2- Programa de aplicação da função OU Exclusivo e NOT

```
#include <SanUSB.h>

short int ledXOR, ledpisca;

void main(void){
clock_int_4MHz();

while(1)
{
ledXOR = entrada_pinb1^entrada_pinb2;  //OU exclusivo entre os pinos b1 e b2 ->input

saida_pino(pin_b7,ledXOR);          //O pino B7 recebe o valor de LedXOR

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

//3- Utiliza a interrupção externa 1, do timer 0 e do timer3

```
#include <SanUSB.h>
// inserir o desvio _asm goto interrupcao _endasm  na função void _high_ISR (void){  } em SanUSB.h

#pragma interrupt interrupcao
void interrupcao()
{

 if (timer0_interrompeu)   {       //espera o estouro do timer0
timer0_interrompeu = 0;              //limpa a flag de interrupção
PORTBbits.RB0 =! PORTBbits.RB0;   //Pisca o LED em B7
tempo_timer16bits(0,62500); }

 if (timer3_interrompeu)   {       //espera o estouro do timer0
timer3_interrompeu=0;                //limpa a flag de interrupção
PORTBbits.RB7 =! PORTBbits.RB7;   //Pisca o LED em B7
```

```
tempo_timer16bits(3,50000); }

  if (ext1_interrompeu)      {        //espera a interrupção externa 1 (em B1)
ext1_interrompeu = 0;                 //limpa a flag de interrupção
PORTBbits.RB0 =! PORTBbits.RB0;  //altera o LED em B0
tempo_ms(100);               }       //Delay para mascarar o ruido do botão(Bouncing)
}

void main()       {
clock_int_4MHz();
TRISB = 0b01111110;          //B0 e B7 como Saída

habilita_interrupcao(timer0);
liga_timer16bits(0,16);           //liga timer0 - 16 bits com multiplicador (prescaler) 8
tempo_timer16bits(0,62500);       //Conta 16 x 62500us = 1 seg.

habilita_interrupcao(timer3);
liga_timer16bits(3,8);           //liga timer1 - 16 bits com multiplicador (prescaler) 8
tempo_timer16bits(3,50000);                      //Conta 8 x 50000us = 0,4 seg.

habilita_interrupcao(ext1);     // habilita a interrupção externa 1 com borda de descida

while(1);
}
```

//4- Programa que utiliza o wdt, reseta em 16 seg. se a limpaflag_wdt() não for limpa

```
#include <SanUSB.h>
void main(){
clock_int_4MHz();
habilita_wdt(); // Se a flag wdt não foor limpa (limpaflag_wdt();) reseta em 16 segundos.

nivel_alto(pin_b0);
nivel_alto(pin_b7);
tempo_ms(3000);

          while (1)
          {
          //limpaflag_wdt();
          nivel_alto(pin_b0); // Pisca Led na função principal
          nivel_baixo(pin_b7);
          tempo_ms(500);
          nivel_baixo(pin_b0);
          nivel_alto(pin_b7);
          tempo_ms(500);
   }
          }
```

//5- Lê o canal AD em 8 e 10 bits e grava o resultado na EEPROM interna

```
#include <SanUSB.h>
unsigned int resultado; //16 bits
int b1=0,b2=0,endprom,endereco=0;

void main(){
clock_int_4MHz();
TRISB=0x00; //Porta B saída
habilita_canal_AD(AN0);
```

```
while(1){

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

resultado = le_AD10bits(0);//Lê canal  0 da entrada analógica com  resolução de 10 bits (ADRES)
b1=resultado/256; b2=resultado%256; //Quebra o resultado em 2 bytes
escreve_eeprom( endprom+1, b1); escreve_eeprom( endprom+2, b2 );

++endereco; //Incrementa endereco
if(endereco>=75){endereco=0;}
endprom=3*endereco;
tempo_ms(1000);             }
}
```

//6 - Utiliza interrupção serial por recepção de caractere e interrupção do timer0

```
#include <SanUSB.h>
// inserir o desvio _asm goto interrupcao _endasm  na função void _high_ISR (void){   } em SanUSB.h

#pragma interrupt interrupcao
void interrupcao()
{
  unsigned char c;

 if (serial_interrompeu)  {
serial_interrompeu=0;
c = le_rs232();
    if (c >= '0' && c <= '9')      { c -= '0';    PORTA = c;}
    PORTBbits.RB0 =! PORTBbits.RB0;     //Inverte  o LED em B0
    tempo_ms(500);

 if (timer0_interrompeu)     {         //espera o estouro do timer0
timer0_interrompeu = 0;                  //limpa a flag de interrupção
PORTBbits.RB7 =! PORTBbits.RB7;    //Inverte o LED em B7
tempo_timer16bits(0,62500);  }
                                                    }
}

void main()       {
clock_int_4MHz();
TRISB = 0b01111110;         //B0 e B7 como Saída

habilita_interrupcao(recep_serial);
taxa_rs232(9600);

habilita_interrupcao(timer0);
liga_timer16bits(0,16);          //liga timer0 - 16 bits com multiplicador (prescaler) 16
tempo_timer16bits(0,62500);       //Conta 16 x 62500us = 1 seg.

putrsUSART ( (const far rom char *) "\n\rDigite um numero de 0 a 9!\n\r");

escreve_rs232(PORTC); //escreve valor da PORTC
while (envia_byte());
escreve_rs232('A'); // escreve A
```

```
while (envia_byte());

while(1);           }
```

7 - Abaixo é mostrado um programa com a configuração para a interrupção do *timer* 0, em que pisca alera o estado de um *LED* a cada 0,5 segundo no pino B7 e também a interrupção externa 1 (com *pull-up* externo no pino B1) em que muda e estado de outro *LED* no pino B0. Em seguida são mostrados os registros de configuração do *timer* 0 e das interrupções.

```
#include <SanUSB.h>

#pragma interrupt interrupcao
void interrupcao()
{
  if (INTCONbits.TMR0IF)        {        //espera o estouro do timer0
INTCONbits.TMR0IF = 0;                   //limpa a flag de interrupção
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 (INTCON3bits.INT1IF)             {       //espera a interrupção externa 1 (em B1)
INTCON3bits.INT1IF = 0;                    //limpa a flag de interrupção
PORTBbits.RB0 =! PORTBbits.RB0; }       //altera o LED em B0
delay_ms(100);                             //Delay para mascarar o ruido do botão(Bouncing)
}

void main()       {
clock_int_4MHz();
TRISB = 0b01111110;                 //B0 e B7 como Saída

RCONbits.IPEN = 1;                  //apenas interrpções de alta prioridade (desvio 0x808)
INTCONbits.GIEH = 1;                //Habilita interrupções de alta prioridade (0x808)
INTCONbits.TMR0IE = 1;              // habilita a interrupção do TMR0
T0CON = 0b10000010;                 //setup_timer0 - 16 bits com prescaler 1:8 e oscilador interno

INTCON3bits.INT1IE = 1;          // habilita a interrupção externa 1 (pino B1)
INTCON2bits.INTEDG1 = 0;     // habilita que a interrupção externa 1 ocorra somente na borda de descida

while (1);
                    }
```

RCON: RESET CONTROL REGISTER

R/W-0	R/W-1(1)	U-0	R/W-1	R-1	R-1	R/W-0(2)	R/W-0
IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$

IPEN: Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts

$\overline{TO}$: Watchdog Time-out Flag bit

```
RCONbits.IPEN = 1;
```

INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)

GIE/GIEH: Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all high priority interrupts

PEIE/GIEL: Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

TMR0IE: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

INT0IE: INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

```
INTCONbits.GIEH = 1;        //Habilita interrupções de alta prioridade (0x808)
INTCONbits.TMR0IE = 1;      // habilita a interrupção do TMR0
```

INTCON2: INTERRUPT CONTROL REGISTER 2

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP

$\overline{\text{RBPU}}$: PORTB Pull-up Enable bit
1 = All PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

INTEDG0: External Interrupt 0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

```
INTCON2bits.INTEDG1 = 0;    // habilita que a interrupção externa 1 ocorra na borda de descida
```

INTCON3: INTERRUPT CONTROL REGISTER 3

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF

INT2IP: INT2 External Interrupt Priority bit
1 = High priority
0 = Low priority

INT1IP: INT1 External Interrupt Priority bit
1 = High priority
0 = Low priority

INT2IE: INT2 External Interrupt Enable bit
1 = Enables the INT2 external interrupt
0 = Disables the INT2 external interrupt

INT2IF: INT2 External Interrupt Flag bit
1 = The INT2 external interrupt occurred (must be cleared in software)
0 = The INT2 external interrupt did not occur

INT1IE: INT1 External Interrupt Enable bit
1 = Enables the INT1 external interrupt
0 = Disables the INT1 external interrupt

INT1IF: INT1 External Interrupt Flag bit
1 = The INT1 external interrupt occurred (must be cleared in software)
0 = The INT1 external interrupt did not occur

```
INTCON3bits.INT1IE = 1;     // habilita a interrupção externa 1

 if (INTCON3bits.INT1IF)        {        //espera o estouro do timer0
INTCON3bits. INT1IF = 0;                    //limpa a flag de interrupção
```

T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

TMR0ON: Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

T08BIT: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

T0CS: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

T0PS2:T0PS0: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

PSA: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

```
T0CON = 0b10000010;          //setup_timer0 - 16 bits com prescaler 1:8 e oscilador interno
```

APÊNDICE III: O AMPLIFICADOR OPERACIONAL

Um amplificador operacional (abreviadamente AmpOp) é basicamente um dispositivo amplificador de tensão, caracterizado por um elevado ganho em tensão, impedância de entrada elevada (não "puxam" corrente), impedância de saída baixa e elevada largura de banda. O termo "operacional" surgiu porque foi projetado inicialmente para realizar operações matemáticas em computadores analógicos.

Estes dispositivos são normalmente dotados de uma malha de realimentação com funções que transcendem a simples amplificação.

O ampop é um componente que possui dois terminais de entrada e um terminal de saída que é referenciado à massa. O seu símbolo elétrico, que se apresenta na Figura 1, é um triângulo que aponta no sentido do sinal. Das duas entradas, uma, assinalada com o sinal (-) é chamada de entrada inversora e a outra, a que corresponde o sinal (+) é chamada entrada não-inversora. A saída faz-se no terminal de saída que se encontra referenciado à massa. O amplificador é normalmente alimentado com tensões simétricas, tipicamente +12 V e –12 V ou +15 V e –15 V, que são aplicadas aos respectivos terminais de alimentação V+ e V-. Note-se que nos esquemas elétricos freqüentemente estes terminais são omitidos, representando-se apenas as entradas e a saída.

Em alguns casos podem estar disponíveis terminais adicionais que permitem compensar deficiências internas do amplificador, como a tensão de desvio (ou offset).

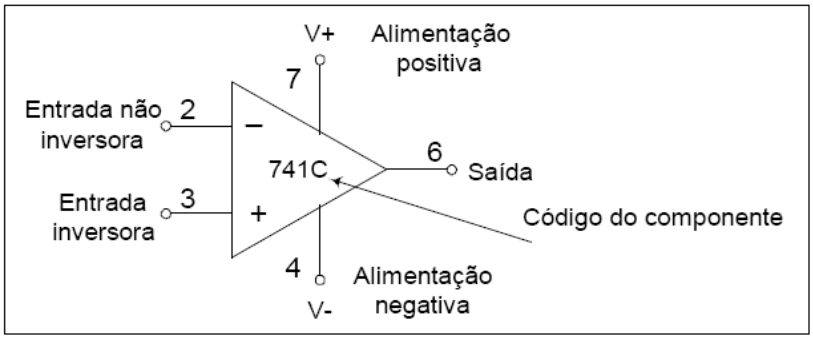


Figura 1: Simbologia de um amplificador operacional

Ganho de tensão - Normalmente chamado de ganho de malha aberta (sem realimentação), medido em C.C.(ou em freqüências muito baixas), é definido como a relação da variação da tensão de saída para uma dada variação da tensão de entrada. Este parâmetro, notado como A, tem seus valores reais que vão desde alguns poucos milhares até cerca de cem milhões em amplificadores operacionais sofisticados. Normalmente, A é o ganho de tensão diferencial em C.C.. O ganho de modo comum é, em condições normais, extremamente pequeno.

O amplificador diferencial (AmpD) é o primeiro estágio de um AmpOp estabelecendo algumas de suas principais características. Por definição um AmpD é um circuito que tem duas entradas nas quais são aplicadas duas tensões Vi_1 e Vi_2 e uma saída (a) ou duas saídas (b). No caso ideal, $Vo = A.(Vi_1 - Vi_2)$ onde A é o Ganho de tensão diferencial. Se considerarmos a condição ideal, se $Vi_1 = Vi_2$, a saída será nula, isto é, um AmpD é um circuito que amplifica só a diferença entre duas tensões rejeitando os sinais de entrada quando estes forem iguais.

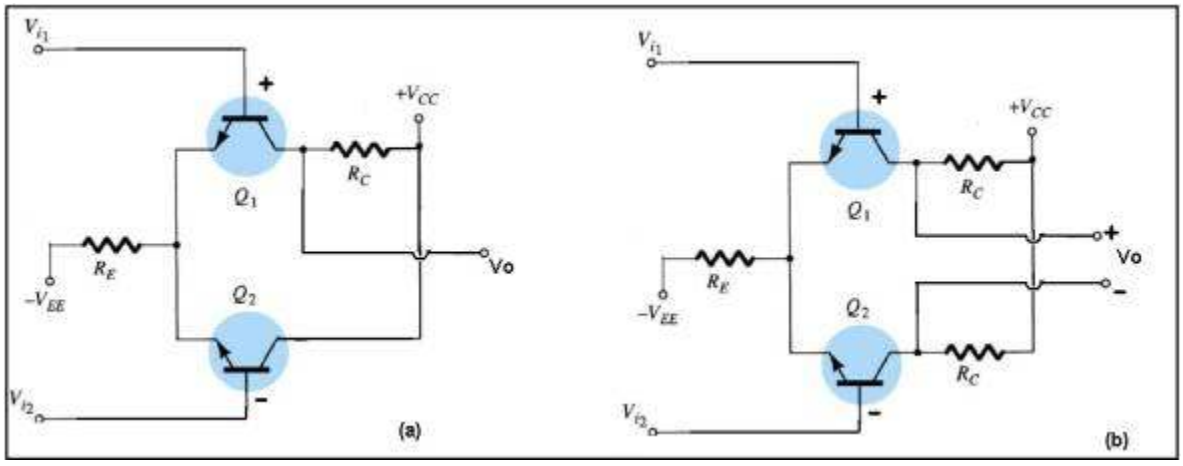


Figura 2: Circuito simplificado de um amplificador diferencial

Geralmente, o amplificador diferencial apresenta apenas um terminal de saída (a), pois na maioria dos circuitos um lado da carga é conectado ao terra.

Os amplificadores operacionais possuem elevada impedância de entrada e baixa impedância na saída. Para amplificadores operacionais como o 741, a resistência de entrada é de 1 MΩ, a resistência de saída é da ordem de 75Ω e o ganho pode chegar a 100.000.

Note em (a), que o lado da entrada positiva é o mesmo lado da alimentação +Vcc e que, quando a entrada não-inversora ou o transistor Q1 é saturado, parte da corrente de +Vcc tende a ir, passando por Rc, no sentido de Vo, gerando uma tensão na saída Vo positiva. Quando a entrada é inversora ou o transistor Q2 é saturado, parte da corrente tende a ir em sentido contrário, gerando uma tensão na saída Vo negativa.

Na verdade, a estrutura interna de um amplificador operacional é muito complexa, sendo constituído por dezenas de transistores e resistências contidos numa muito pequena pastilha de Silício (chip). A figura 3 mostra o diagrama de componentes internos do LM741.

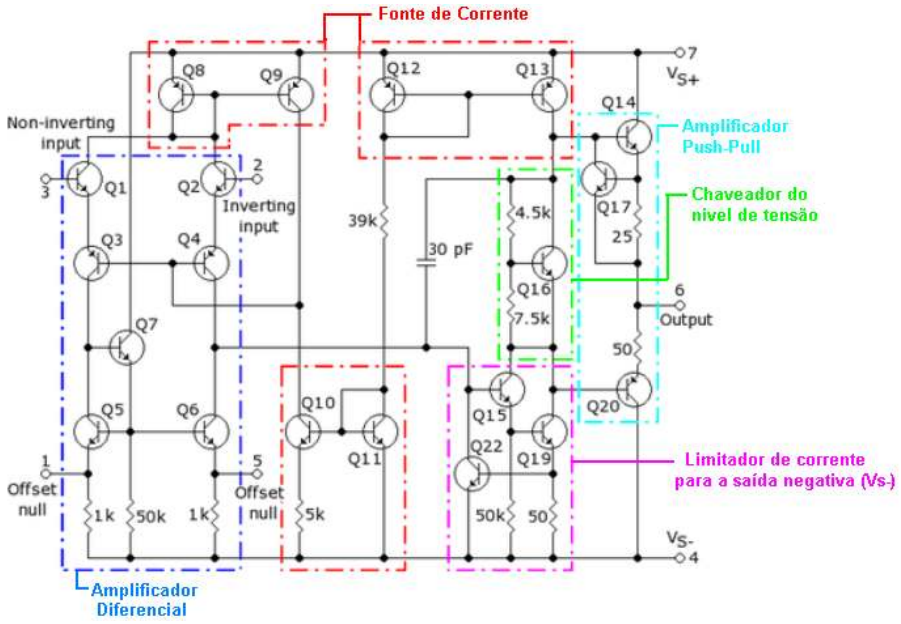


Figura 3: Diagrama de componentes internos do LM741

Analogia de um Amplificador Operacional

Um amplificador operacional é alimentado pelo desequilibro das duas entradas. Quando há uma tensão diferencial, ele satura rapidamente.

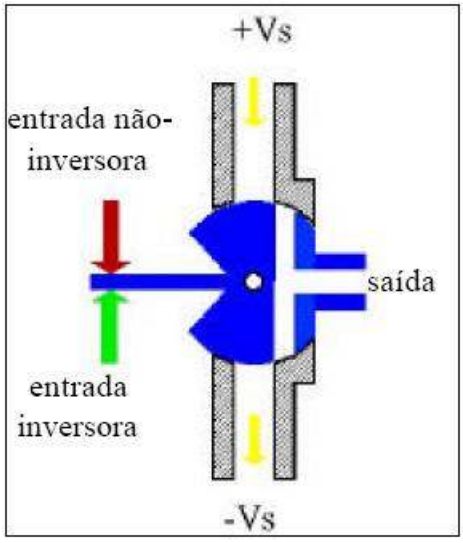


Figura 4: Analogia de um Amplificador operacional

Essa simples analogia de um AO e o fluxo de água está próxima da dinâmica real. À medida que a diferença de força nas duas entradas se torna finita, a peça azul gira, e a saída é conectada a umas das duas tensões de alimentação. Os canais são de tal forma que a saída é rapidamente enviada ao fornecimento +Vs ou -Vs. Quando o equilíbrio entre as entradas é restaurado, então a saída é mais uma vez configurada em zero.

Tensão de "offset" - A saída de um amplificador operacional ideal é nula quando suas entradas estão em mesmo nível de tensão. Nos amplificadores reais, devido principalmente a um casamento imperfeito dos dispositivos de entrada, normalmente diferencial, a saída do amplificador operacional pode ser diferente de zero quando ambas entradas estão no potencial zero. Significa dizer que há uma tensão CC equivalente, na entrada, chamada de tensão de "offset". O valor da tensão de "offset" nos amplificadores comerciais estão situado na faixa de 1 a 100 mV. Os componentes comerciais são normalmente dotados de entradas para **ajuste da tensão de "offset".**

Amplificador operacional real

Na prática os Amp-Op's são circuitos integrados que, como qualquer sistema físico tem suas limitações. Um dos Amp-Op's mais difundidos até hoje é o 741, que recebe inúmeras codificações de acordo com seu fabricante, como por exemplo: uA741, LM741 entre outras. **Os pinos 1 e 5 são destinados ao ajuste da tensão de off-set**. O Amp-Op 741 é mostrado na figura a seguir:

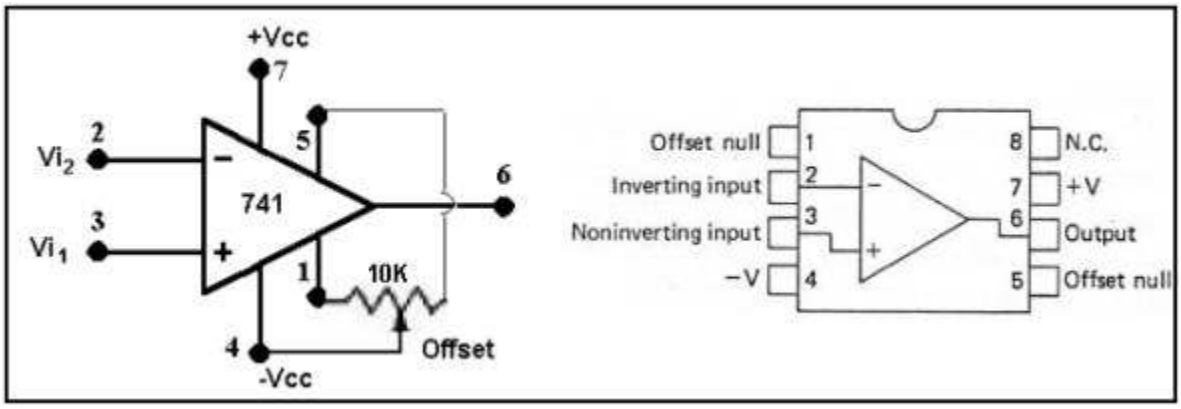


Figura 5: Representação de um amplificador operacional 741

A descrição dos pinos é a seguinte:

1 e 5 - São destinados ao ajuste da tensão de off-set

2- Entrada inversora

3- Entrada não-inversora

4- Alimentação negativa (-3V a -18V)

7- Alimentação positiva (+3V a +18V)

6- Saída

8- Não possui nenhuma conexão

Modos de Operação do Amplificador Operacional

O amplificador operacional pode ser utilizado basicamente de três modos distintos.

1) Sem Realimentação (Malha aberta)

Como foi visto, O amplificador operacional é um amplificador diferencial, que amplifica a diferença entre as tensões presentes as suas entradas. Se V_1 e V_2 forem as tensões aplicadas às entradas não inversora e inversora respectivamente e Vo for a tensão de saída, então:

$$Vo=A\ (V_1-V_2) \qquad (1)$$

em que A é o ganho do amplificador, dito em malha aberta (sem realimentação). Este ganho é normalmente muito elevado, sendo da ordem de 10^5 ou superior. A tensão máxima de saída é igual à tensão de alimentação, por exemplo, ±15 V, o que significa que em malha aberta, uma diferença de tensão da ordem de 100mV entre as duas entradas é suficiente para elevar a saída a este valor, saturando o amplificador. Na Figura 2 representa-se esta "característica de transferência" de um amplificador operacional, isto é, o traçado da tensão de saída em função da tensão de entrada.

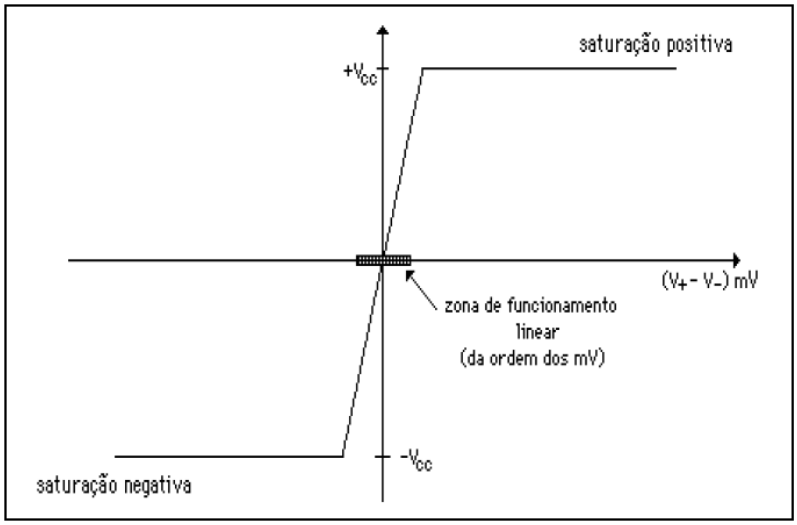


Figura 4: Função de transferência de um amplificador operacional em malha aberta

O amplificador operacional como um amplificador diferencial de ganho bastante alto deixa claro que a tensão da saída é levada muito rapidamente para as tensões de alimentação. Com um ganho de cerca de 1 milhão, é necessária somente uma diferença de alguns micro-volts entre as duas entradas para levar o amplificador até a saturação.

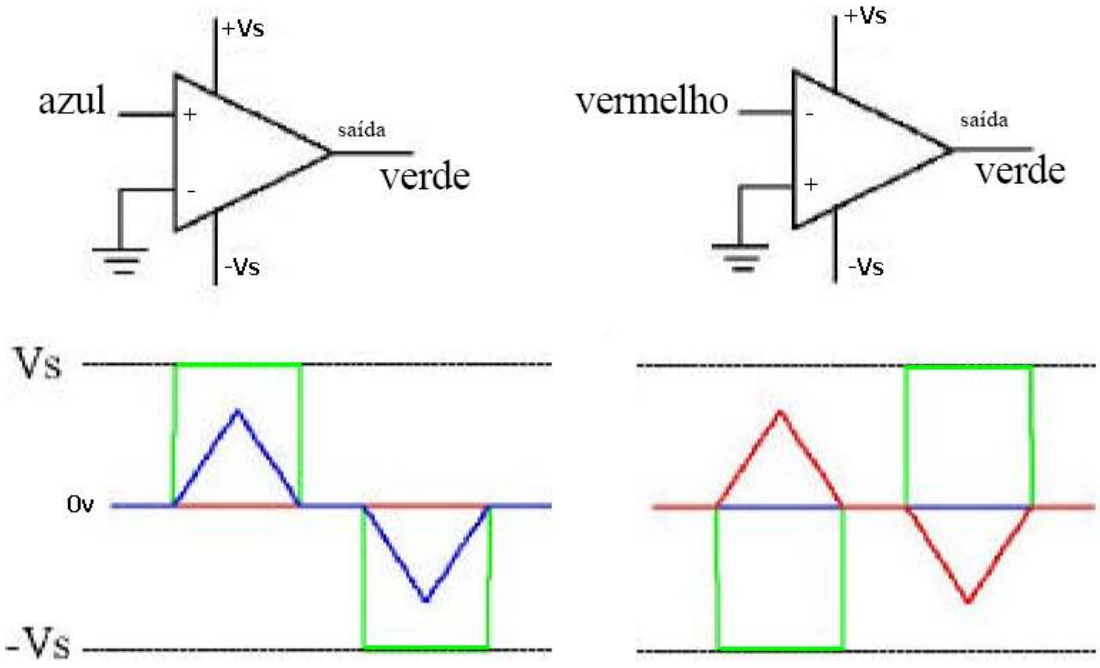
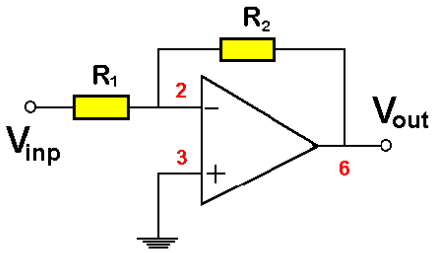


Figura 5: Tensão de saída de um amplificador operacional em malha aberta

Esse sistema em malha aberta também é conhecido como comparador de tensão entre Vi (tensão de entrada) e Vref (tensão de referência), que nesse tem a Vref igual ao Gnd.

2) Amplificador Inversor com Realimentação Negativa

Esta é a montagem básica mais utilizada com amplificadores operacionais no cotidiano de laboratórios, no interior de equipamentos que amplificam sinais, etc..



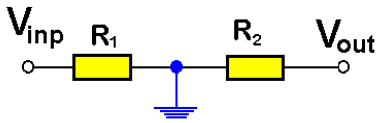
Características:

- A tensão na saída (Vo) será nula ou a desejada quando as entradas inversora (-) e não inversora (+) apresentem o mesmo potencial.

- **Como a entrada não inversora (+) está aterrada**, **a entrada inversora (-) será um terra virtual.**

- Nenhuma das entradas (em teoria) permite a passagem de corrente elétrica do exterior para o amplificador operacional (impedância de entrada infinita).

- Se a entrada inversora é um terra virtual, temos que, simplesmente, resolver o circuito abaixo, onde **o terra virtual** é representado:

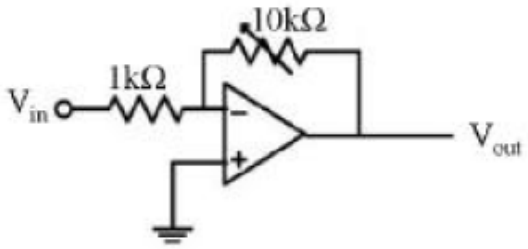


Para que haja o terra virtual é necessário que Iin = - Iout, então:

$$\frac{V_{in} - V_{-}}{R_1} = -\frac{V_{out} - V_{-}}{R_2} \Rightarrow \frac{V_{out}}{V_{in}} = -\frac{R_2}{R_1}$$

Quase todas as aplicações de AmpOps envolvem realimentação negativa. Nesse caso, quando a tensão de saída aumenta, uma parte da tensão de saída é realimentada para a entrada inversora, i reduzindo a saída. Muito rapidamente, o AmpOp encontra seu ponto operacional. Note que o ganho do AmpOp depende da relação entre R_2 e R_1.

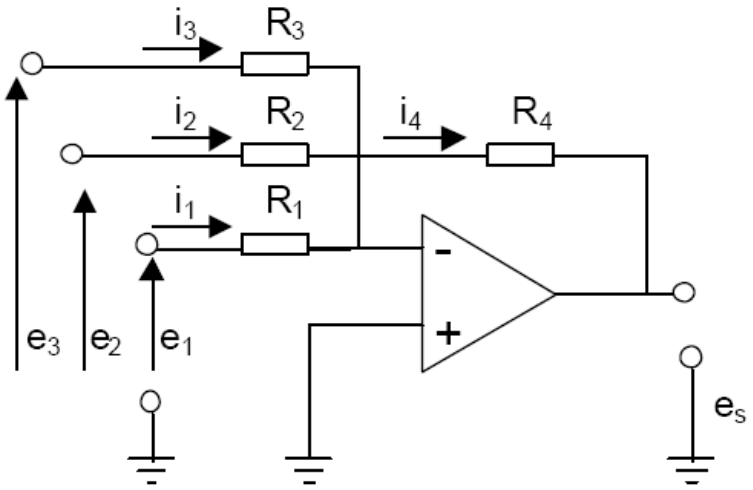
Exemplo de um amplificador inversor:



Nesse exemplo, o ganho de tensão com o resistor de realimentação variável em 10KΩ é 10. Diminuindo-se o valor desse resistor, o ganho pode ficar bastante pequeno, e o dispositivo se torna essencialmente um buffer inversor.

2.1) Amplificador Somador Inversor

Nesse circuito, a corrente i_4 é igual a soma de i_1 , i_2 e i_3. Na figura abaixo, observa-se que o circuito é um amplificador somador, em que cada entrada pode ser operada com fatores de escala diferentes. $i_4 = - e_s / R_4$



$$e_S = -R_4\left(\frac{e_1}{R_1}+\frac{e_2}{R_2}+\frac{e_3}{R_3}\right)$$

Uma das aplicações mais utiizadas do somador inversor é a realização de um conversor digital-analógico (DA). Com efeito, considerando, por exemplo, que as fontes de sinal digital de entrada valem 1 V ou 0 V, e as resistências Ri se encontram organizadas binariamente em função da ordem de grandeza do bit, por exemplo, R1=R, R2=R/2, R3=R/4... Rk=R / 2^{k-1}.

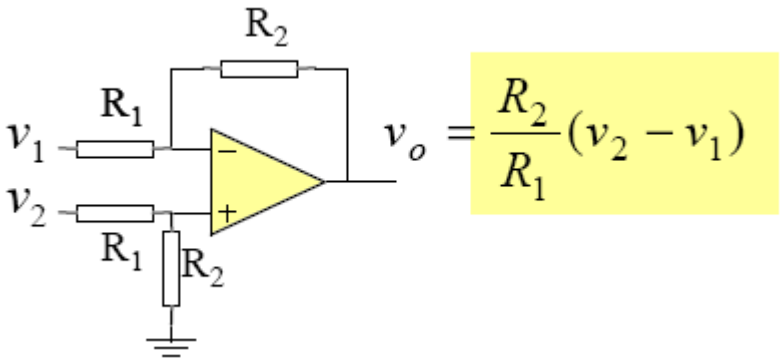
Dessa forma, considerando R4 igual a R e as palavras digitais 10011 e 00001 (em decimal 19 e 1, respectivamente), os valores de tensão na saída serão:

Vo = -(16 + 0 + 0 + 2 + 1) = -19V

Vo = -(0 + 0 + 0 + 0 + 1) = -1V

Na prática pode se considerar o valor de R4 muito maior que R para limitar o valor da tensão máxima de saída em 5V. Uma prática interessante é construir um conversor DA a partir de sinais digitais de uma porta do microcontrolador e conferindo o valor convertido com um multímetro.

2.2) Amplificador Subtrador



2.3) Amplificador Integrador

O integrador e o diferenciador são circuitos que simulam os operadores matemáticos integral e derivada respectivamente. Além disso, são usados para modificar formas de onda, gerando pulsos, ondas quadradas, ondas triangulares etc.

A Fig2.30 mostra o circuito básico de um integrador

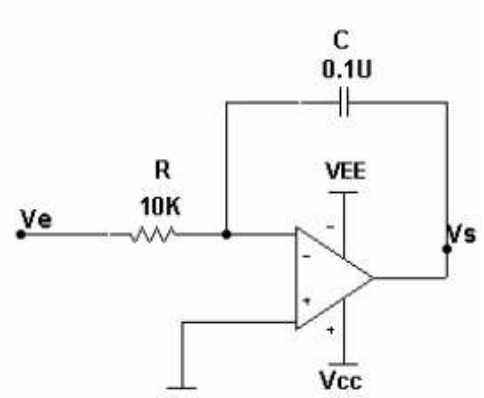


Fig2.30: Integrador

A expressão da tensão de saída em função da entrada é dada por:

$$Vs = -\frac{1}{R.C}.\int Vedt$$

Isto é , a tensão de saída é proporcional à integral da tensão de entrada. O sinal de menos se deve à configuração inversora do AmpOp.

Por exemplo, se a entrada for uma tensão constante, a saída será uma rampa. Se for uma tensão positiva a rampa será descendente(inclinação negativa), se for uma tensão negativa a rampa será ascendente (inclinação positiva).

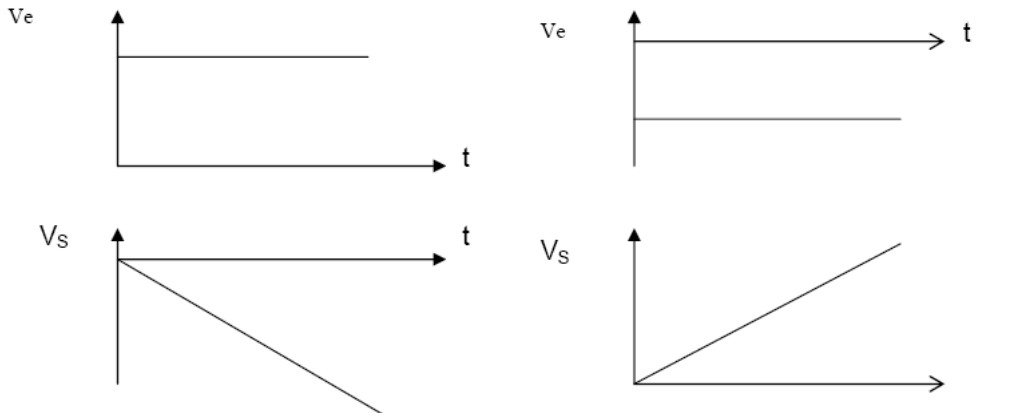


Fig2.31: Resposta de um integrador a um degrau de tensão (a) positiva e (b) Negativo.

Na pratica o circuito da Fig2.30 apresenta um problema, como o **circuito não tem realimentação em CC (o capacitor é circuito aberto quando carregado em CC)**, desta forma o ganho é muito alto (comportamento igual a malha aberta), fazendo o AmpOp saturar mesmo com tensões da ordem de mV como a tensão de offset de entrada.

A solução é diminuir o ganho em CC colocando em paralelo com o capacitor C um resistor de realimentação, R_P, como na Fig2.32. O circuito, porém, só se comportará como integrador para freqüências muito acima da frequência de corte f_C (saturação do capacitor), pois quando o capacitor satura ele se torna aberto e o circuito se comporta como amplificador inversor de ganho igual a:

$$Av = -\frac{Rp}{R}$$

Na frequência de corte a impedância de C fica igual a R_P , isto é, $X_C = R_P$ ou

$$\frac{1}{2.\pi.f_C.C} = R_P \quad \text{daí obtemos}$$

$$f_C = \frac{1}{2\pi R_p C}$$

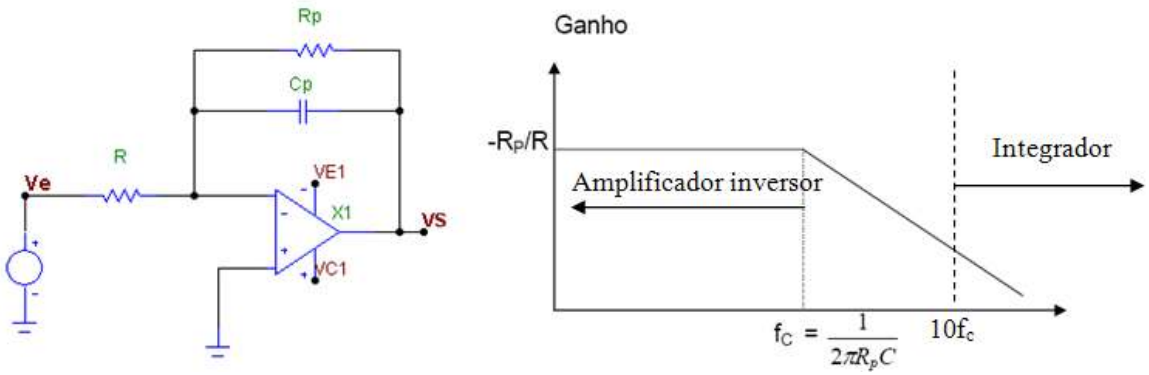


Fig2.31: Integrador com resistor de realimentação limitador de ganho

Exercício Resolvido

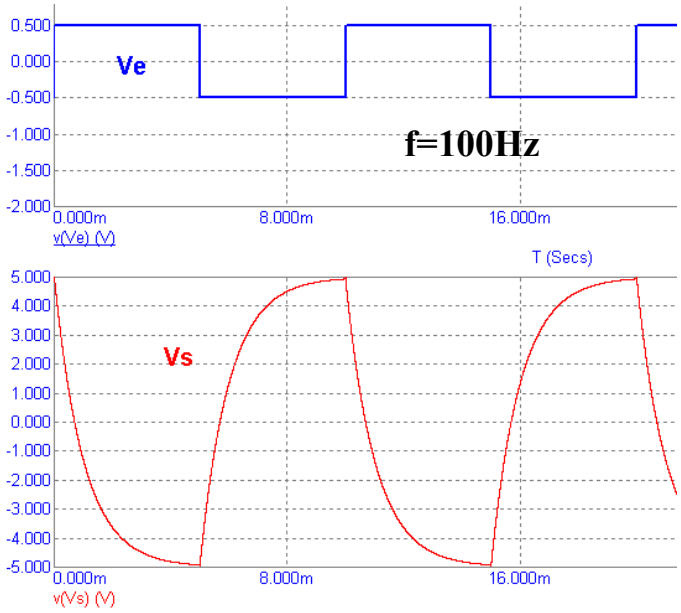
2.21 Se na Fig2.31 R_P = 10K , R = 1K e C = 0,1uF, para que freqüências obteremos na saída uma onda triangular se a entrada for uma onda quadrada ?

Solução: A freqüência de corte do circuito é:

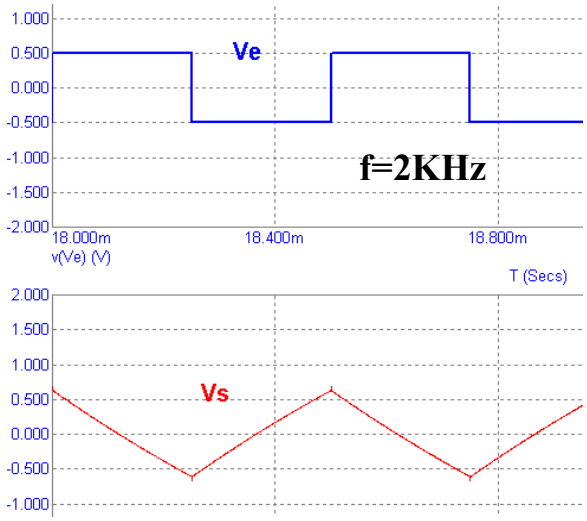
$$f_C = \frac{1}{2.\pi.10.10^3.0{,}1.10^{-6}} = 160Hz$$

Portanto, para freqüências muito acima de 160Hz teremos uma boa integração, isto é, obteremos na saída uma onda triangular com grande linearidade.

Quanto maior for a frequência do sinal em relação à frequência de corte, melhor será a integração do sinal. Na Fig2.33a a freqüência da onda quadrada de entrada é menor do que fc e na Fig2.33b a freqüência da onda quadrada é muito maior do que fc, resultando uma saída com menor amplitude mas perfeitamente triangular.



(a)



(b)

Fig2.33: Resposta de um integrador a uma entrada quadrada a diferentes freqüências

Diferenciador

O diferenciador é um circuito que dá uma saída proporcional à derivada do sinal de entrada é. A derivada é um operador dual da integral, e no circuito os componentes trocam de posição, Fig2.34.

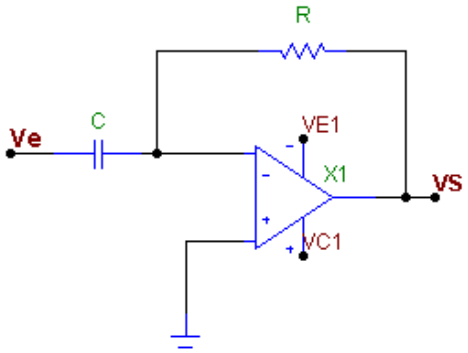


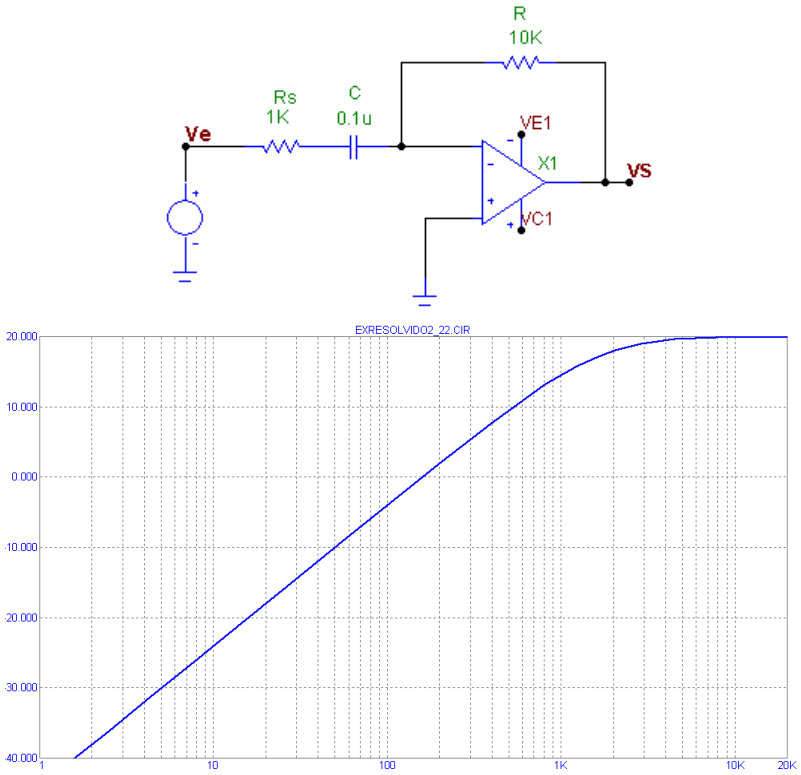
Fig2.34: Diferenciador

A expressão da saída em função da entrada é dada por:

$$V_S = -R.C.\frac{dV_e}{dt}$$

Isto é, a tensão de saída é proporcional à derivada da tensão de entrada. Por exemplo se a entrada for uma tensão constante a saída será nula pois a derivada de uma constante é zero, se a entrada for uma rampa, a saída será constante. O sinal negativo se deve à configuração inversora.

Na prática o circuito da Fig2.34 é sensível a ruído, tendendo a saturar. A solução é limitar o ganho em altas freqüências colocando em série com C uma resistência Rs como na Fig2.35a. A Fig2.35b é a curva de resposta em frequência do circuito.



(a) (b)

Fig2.35: (a) Diferenciador prático e (b) curva de resposta em freqüência

O circuito da figura 2.35a somente funcionará como diferenciador para freqüências muito abaixo da freqüência de corte, acima o circuito se comportará como amplificador inversor de ganho igual a R/R_S.

O circuito só se comportará como diferenciador se $f << f_C$, pois nessas condições a reatância de C será muito maior do que R_S e na prática é como se não existisse R_S e portanto o circuito terá comportamento semelhante ao da Fig2.34.

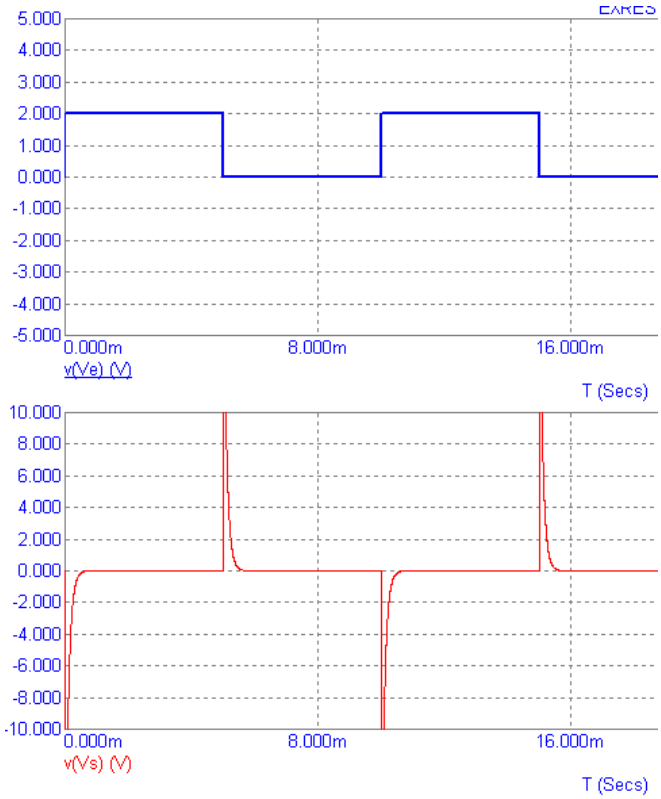
Exercício Resolvido

2.22. Para o circuito da Fig2.35a qual a forma de onda de saída se a entrada é quadrada, para as freqüências de 100Hz e 2KHz.?

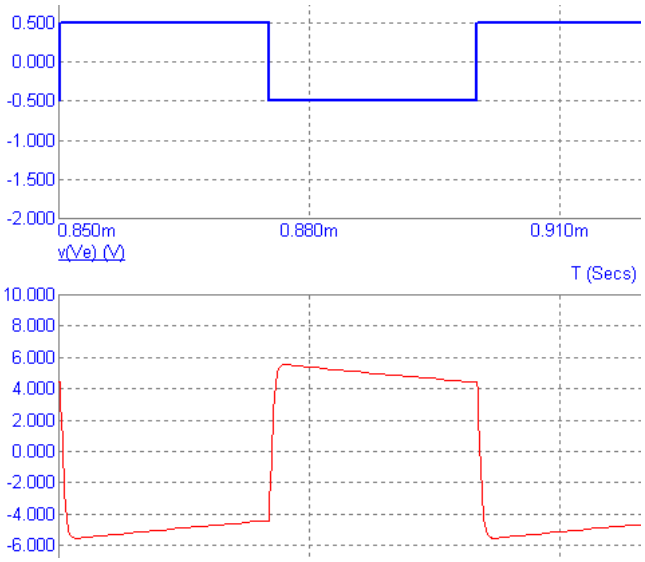
Solução: A freqüência de corte é

$$f_C = \frac{1}{2.\pi.10^3.0{,}1.10^{-6}} = 1600Hz$$

Para freqüências muito abaixo de 1600Hz a saída serão pulsos muito estreitos, negativos na borda de subida e positivos na borda de descida.



(a)



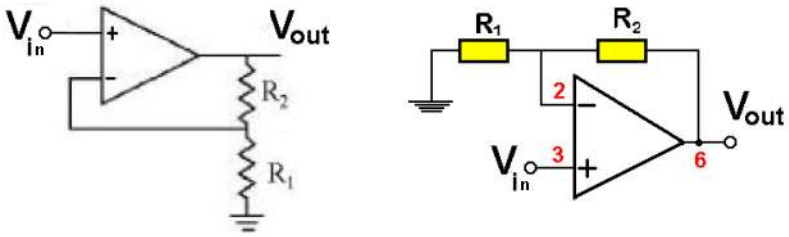
(b)

Fig2.36: Resposta de um diferenciador a uma onda quadrada de (a) f<<fc (b) f>fc

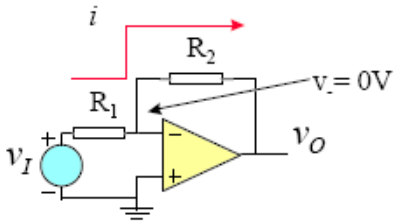
3) Amplificador Não-Inversor com Realimentação Negativa

Continuando com o conceito que a **tensão na saída (Vo) será nula ou a desejada quando as entradas inversora (-) e não inversora (+) apresentem o mesmo potencial** (V+ = V-). Então:

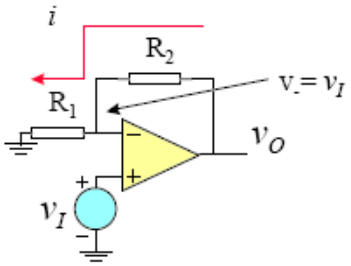
V+ = Vin e V- = Vout *R1 /(R1+R2) -> Vout/Vin = (R1 +R2)/R1 = 1 + R2/R1



O amplificador não-inversor envia a entrada para o terminal não-inversor. **A realimentação deve, obviamente, ir para a entrada inversora**. O ganho é ajustado por amostragem da tensão de saída através de um divisor de tensão. Ajustando-se a razão de R_2 para R_1, pode-se variar a intensidade da realimentação e, dessa forma, mudar o ganho.



Amplificador inversor



Amplificador **não**-inversor

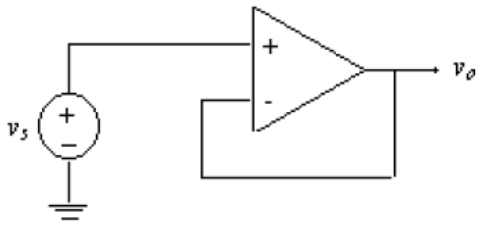
$$\frac{v_I - 0}{R_1} = \frac{0 - v_O}{R_2} \Rightarrow v_O = -v_I \frac{R_2}{R_1}$$

$$\frac{v_I - 0}{R_1} = \frac{v_O - 0}{R_2 + R_1} \Rightarrow v_O = v_I \frac{R_2 + R_1}{R_1} \Rightarrow v_O = v_I \left(\frac{R_2}{R_1} + 1 \right)$$

Note que o desenho de representação do crcuito para os amplificadores invesor e não inversor pode ser o mesmo, pois a realimentação sempre é negatva, modificando somente o local de entrada da tensão V_{in}, que pode ser na entrada inversora (-) ou não inversora (+).

3.1) Seguidor de Tensão

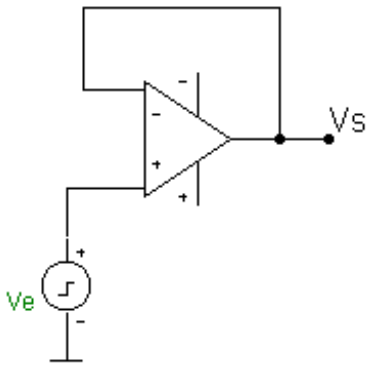
O circuito seguidor de tensão constitui uma das aplicações mais comuns do amplificador operacional. Esse circuito com ganho unitário é designado como isolador ou buffer (amortecedor).



Vo/Vs = 1 -> Vo = Vs

3.2) Slew Rate (Taxa de Inclinação)

Para compreendermos o significado de *Slew Rate* (SR), consideremos o buffer da Figura abaixo alimentado pelos pulsos da Figura a. A tensão de saída teórica e a que realmente se obtém estão indicadas nas Figura b e Figura c respectivamente.



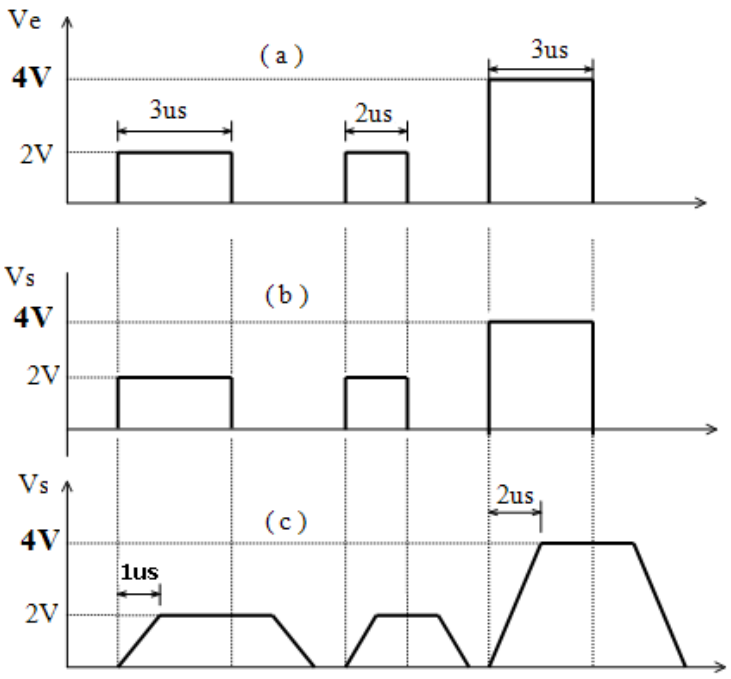


Fig2.9: Buffer – Resposta a um pulso de entrada

O **Slew Rate** (SR) significa taxa de inclinação ou de resposta que é a máxima taxa de variação da tensão na saída com o tempo, isto é:

$$\mathbf{SR = \Delta V_S / \Delta t.}$$

Ela surge devido ao tempo necessário para saturação dos transistores internos do AmpOp.

Na Fig2.9 o AmpOp do exemplo tem um SR = 2V/1μs = 2v/μs, isto significa que se o sinal de entrada for mais rápido do que isso, o AmpOP não responderá na mesma velocidade distorcendo o sinal na saída.

No caso de uma entrada senoidal, $\mathbf{V_S = V_M.senwt}$, a taxa de inclinação na saída de cada ponto é variável sendo dada por sua **derivada**: $\mathbf{SR=dV_S/dt = w.V_M.coswt}$ com valor máximo na origem (wt = 0), pois cos0 = 1, e valendo portanto :

$$\mathbf{SR = w.V_M}$$

$$\mathbf{SR = 2\pi f .V_M}$$

A Fig2.10 mostra o comportamento da derivada, inclinação ou slew rate, de uma senóide, sendo máxima na origem e zero para wt = 90º.

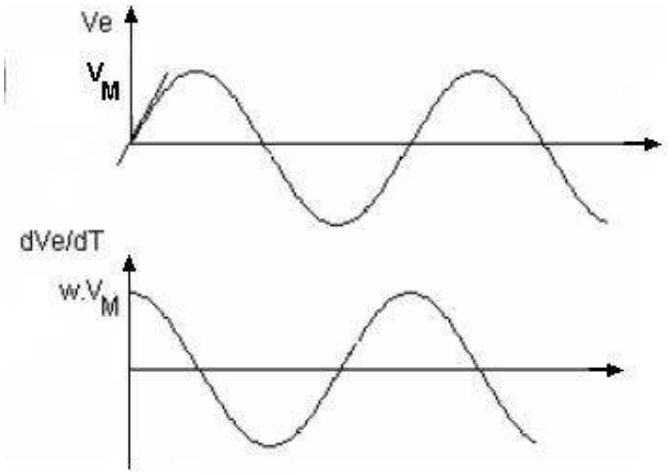


Fig2.10: Comportamento da derivada da senóide

A conclusão: enquanto a capacidade de SR do AmpOp for maior do que $2\pi f.V_M$ ($SR = 2\pi f.V_M$) não haverá distorção, caso contrário a senoide começa a ficar achatada. O AmpOp 741 possui o SR = 0,5V/µs, o LF351 possui SR = 13V/µs e o LM318 possui SR=70V/µs.

Exercício Resolvido

2.9. Um AmpOp tem SR = 2V/μs, qual a máxima freqüência (w=2.π.f) que pode ter um sinal senoidal de 10V de amplitude na entrada do AmpOp para que não haja distorção por slew rate na saída?

Solução:

Para que não haja distorção $\mathbf{SR = 2\pi f_{máx} .V_M}$

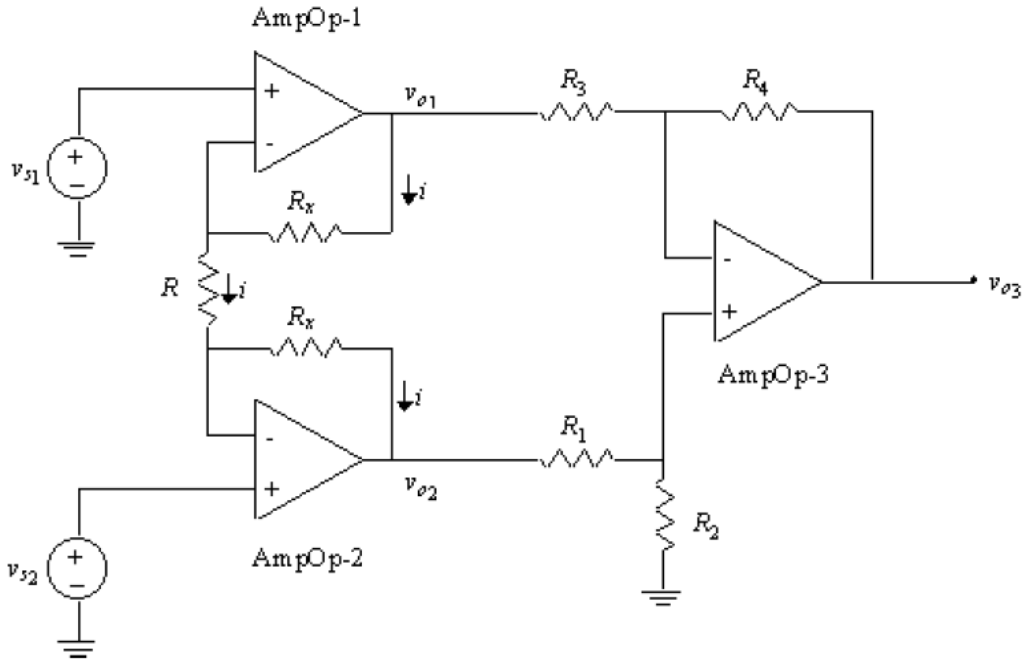
$2.10^6 V/s = 2.\pi.f_{máx}.10V$

$f_{máx} = 31847Hz$

Amplificador de Instrumentação

Muito utilizados por sensores com sinais de tensão diferenciais como termopares e sensores de corrente. O amplificador de instrumentação representado na figura abaixo, adota dois amplificadores não inversores (AmpOps 1 e 2) na entrada e um amplificador diferencial (AmpOp 3) na saída. Neste caso, a resistência de entrada vista por cada uma das duas fontes é infinita (com a mesma resistência de entrada dos terminais positivos dos AmpOps 1 e 2), e o ganho de tensão é dado pelo produto de dois cocientes entre as resistências no amplificador diferencial.

Nesse caso a tensão de referência na entrada v_{s2} pode ser flutuante, ou seja, é possível amplificar faixas de tensões entre as entradas vs1 e vs2 não simétricas.



A análise deste circuito pode ser efetuada em três passos:

(i) determinação das tensões V_{o1} e V_{o2} nas saídas não inversoras dos AmpOps 1 e 2;

Para vo1 = vo2 = 0, então:

$v1+ = v1- = v_{s1}$ -> $v_{o1} = v1- + Rx \,.\, i$ e $i = (v_{s1}-v_{s2}) / R$ -> $v_{o1} = v_{s1} + Rx \,.\, (v_{s1}-v_{s2}) / R$

$v2+ = v2- = v_{s2}$ -> $v_{o2} = v2- - Rx \,.\, i$ e $i = (v_{s1}-v_{s2}) / R$ -> $v_{o2} = v_{s2} - Rx \,.\, (v_{s1}-v_{s2}) / R$

(ii) obtenção das expressões das tensões nos respectivos nós de saída;

$v_{o1} - v_{o2} = (v_{s1}- v_{s2}) + 2.\, Rx \,.\, (v_{s1}-v_{s2}) / R$ -> $vo1 - vo2 = (v_{s1}-v_{s2}) (1 + 2.\, Rx / R)$

(iii) aplicação da expressão do amplificador diferencial não inversor para determinar a tensão na saída do circuito.

Assim, verifica-se que:

$V_{o3} = R4/R3\ (v_{o1} - v_{o2})$ -> $V_{o3} = R4/R3\ (v_{s1}-v_{s2})\ (1 + 2.\ Rx / R)$

$$\mathbf{V_{o3} / (v_{s1}-v_{s2}) = R4/R3\ (1 + 2.\ Rx / R)}$$

APÊNDICE IV: BIBLIOTECAS UTILIZADAS

BIBLIOTECA i2c_sanusb.c:

```
/**********http://br.groups.yahoo.com/group/GrupoSanUSB/************************/
// Definições dos pinos de comunicação I2C SanUSB

  #ifndef scl
            #define scl  pin_c1                    // pino de clock
            #define sda  pin_c2                    // pino de dados
            //#define scl  pin_d5                  // pino de clock
            //#define sda  pin_d6                  // pino de dados

            #define EEPROM_SIZE 32768       // tamanho em bytes da memória EEPROM
  #endif

  #define seta_scl  output_float(scl)                  // seta o pino scl
  #define apaga_scl  output_low(scl)                   // apaga o pino scl
  #define seta_sda  output_float(sda)                  // seta o pino sda
  #define apaga_sda  output_low(sda)                   // apaga o pino sda

/*****************************************************************************/
//SUB-ROTINAS DA COMUNICAÇÃO I2C

  void I2C_start(void)
  // coloca o barramento na condição de start
  {
    apaga_scl;          // coloca a linha de clock em nível 0
    delay_us(5);
            seta_sda;   // coloca a linha de dados em alta impedância (1)
    delay_us(5);
            seta_scl;   // coloca a linha de clock em alta impedância (1)
    delay_us(5);
            apaga_sda;              // coloca a linha de dados em nível 0
    delay_us(5);
            apaga_scl; // coloca a linha de clock em nível 0
    delay_us(5);
  }

  void I2C_stop(void)
  // coloca o barramento na condição de stop
```

```
{
        apaga_scl; // coloca a linha de clock em nível 0
  delay_us(5);
        apaga_sda;          // coloca a linha de dados em nível 0
  delay_us(5);
        seta_scl;   // coloca a linha de clock em alta impedância (1)
  delay_us(5);
        seta_sda;   // coloca a linha de dados em alta impedância (1)
  delay_us(5);
}

void i2c_ack()
// coloca sinal de reconhecimento (ack) no barramento
{
        apaga_sda;          // coloca a linha de dados em nível 0
  delay_us(5);
        seta_scl;   // coloca a linha de clock em alta impedância (1)
  delay_us(5);
        apaga_scl; // coloca a linha de clock em nível 0
  delay_us(5);
        seta_sda;   // coloca a linha de dados em alta impedância (1)
  delay_us(5);
}

void i2c_nack()
// coloca sinal de não reconhecimento (nack) no barramento
{
        seta_sda;   // coloca a linha de dados em alta impedância (1)
  delay_us(5);
        seta_scl;   // coloca a linha de clock em alta impedância (1)
  delay_us(5);
        apaga_scl; // coloca a linha de clock em nível 0
  delay_us(5);
}

boolean i2c_le_ack()
// efetua a leitura do sinal de ack/nack
{
        boolean estado;
        seta_sda;   // coloca a linha de dados em alta impedância (1)
  delay_us(5);
        seta_scl;   // coloca a linha de clock em alta impedância (1)
  delay_us(5);
        estado = input(sda);    // lê o bit (ack/nack)
        apaga_scl; // coloca a linha de clock em nível 0
  delay_us(5);
        return estado;
}

void I2C_escreve_byte(unsigned char dado)
{
// envia um byte pelo barramento I2C
        int conta=8;
        apaga_scl;          // coloca SCL em 0
        while (conta)
        {
                // envia primeiro o MSB
                if (shift_left(&dado,1,0)) seta_sda; else apaga_sda;
                // dá um pulso em scl
                seta_scl;
    delay_us(5);
                conta--;
                apaga_scl;
    delay_us(5);
        }
        // ativa sda
        seta_sda;
}

unsigned char I2C_le_byte()
```

```
// recebe um byte pelo barramento I2C
{
        unsigned char bytelido, conta = 8;
        bytelido = 0;
        apaga_scl;
  delay_us(5);
        seta_sda;
  delay_us(5);
        while (conta)
        {
                // ativa scl
                seta_scl;
                // lê o bit em sda, deslocando em bytelido
                shift_left(&bytelido,1,input(sda));
                conta--;
                // desativa scl
                apaga_scl;
        }
        return bytelido;
}

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
  i2c_le_ack();
  i2c_escreve_byte(endereco >> 8);        // parte alta do endereço
  i2c_le_ack();
  i2c_escreve_byte(endereco); // parte baixa do endereço
  i2c_le_ack();
  i2c_escreve_byte(dado);                                 // dado a ser escrito
  i2c_le_ack();
  i2c_stop();
  while(read_eeprom(0xfd));
  delay_ms(10); // aguarda a programação da memória
}

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
  i2c_le_ack();
  i2c_escreve_byte(endereco); // envia a parte baixa do endereço
  i2c_le_ack();
  i2c_start();
        // envia comando de leitura
  i2c_escreve_byte(0xa1 | (dispositivo << 1));
  i2c_le_ack();
  dado = i2c_le_byte();          // lê o dado
  i2c_nack();
  i2c_stop();
  return dado;
}
/*****************************************************************************/
//*****************************************************************************/
void escreve_rtc(byte enderecoram, byte dado)
// Escreve um dado em um endereço do dispositivo
// endereco - é o endereço da memória a ser escrito
```

```
// dado - é a informação a ser armazenada
{
  i2c_start();
  i2c_escreve_byte(0xd0);
  i2c_le_ack();
  i2c_escreve_byte(enderecoram);              // parte baixa do endereço de 16 bits
  i2c_le_ack();
  i2c_escreve_byte(dado);                                  // dado a ser escrito
  i2c_le_ack();
  i2c_stop();
  while(read_eeprom(0xfd));
  delay_ms(10); // aguarda a programação da memória
}
byte le_rtc(int enderecoram)
// 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;
            i2c_start();
  i2c_escreve_byte(0xd0); // endereça o dispositivo e colocando leitura 0xd1
  i2c_le_ack();
  i2c_escreve_byte(enderecoram);              // envia a parte baixa do endereço
  i2c_le_ack();
  i2c_start();
            // envia comando de leitura
  i2c_escreve_byte(0xd1); // Coloca o LSB (R\W) em estado de leitura
  i2c_le_ack();
  dado = i2c_le_byte();              // lê o dado
  i2c_nack();
  i2c_stop();
  return dado;
}
```

MOD_LCD_SANUSB.C:

```
/************************************************************************/
/*  MOD_LCD_SANUSB.C - Biblioteca de manipulação dO módulo LCD         */
/*                                                                      */
/*  http://br.groups.yahoo.com/group/GrupoSanUSB/                   */
/*                                                                      */
/************************************************************************/
//// lcd_ini();   //Deve ser escrita no inicio da função main()
////
//// lcd_pos_xy(x,y)  Seta a posição em que se deseja escrever
////
//// Ex.: lcd_pos_xy(1,2);  //  x (coluna) y(linha)
////      printf(lcd_escreve,"BIBLIOTECA SANUSB"); //Escreve na linha 2

#ifndef lcd_enable
            #define lcd_enable     pin_b1         // pino Enable do LCD
            #define lcd_rs                                  pin_b0                  // pino rs do LCD

            #define lcd_d4                                  pin_b2                  // pino de dados d4 do LCD
            #define lcd_d5                                  pin_b3                  // pino de dados d5 do LCD
            #define lcd_d6                                  pin_b4                  // pino de dados d6 do LCD
            #define lcd_d7                                  pin_b5                  // pino de dados d7 do LCD
#endif

#define lcd_type 2           // 0=5x7, 1=5x10, 2=2 linhas
#define lcd_seg_lin 0x40    // Endereço da segunda linha na RAM do LCD

// a constante abaixo define a seqüência de inicialização do módulo LCD
byte CONST INI_LCD[4] = {0x20 | (lcd_type << 2), 0xf, 1, 6};

byte lcd_le_byte()
// lê um byte do LCD (somente com pino RW)
{
```

```
        byte dado;
        // configura os pinos de dados como entradas
        input(lcd_d4);
        input(lcd_d5);
        input(lcd_d6);
        input(lcd_d7);
        // se o pino rw for utilizado, coloca em 1
        #ifdef lcd_rw
                output_high(lcd_rw);
        #endif
  output_high(lcd_enable); // habilita display
        dado = 0;  // zera a variável de leitura
        // lê os quatro bits mais significativos
        if (input(lcd_d7)) bit_set(dado,7);
        if (input(lcd_d6)) bit_set(dado,6);
        if (input(lcd_d5)) bit_set(dado,5);
        if (input(lcd_d4)) bit_set(dado,4);
        // dá um pulso na linha enable
        output_low(lcd_enable);
        output_high(lcd_enable);
        // lê os quatro bits menos significativos
        if (input(lcd_d7)) bit_set(dado,3);
        if (input(lcd_d6)) bit_set(dado,2);
        if (input(lcd_d5)) bit_set(dado,1);
        if (input(lcd_d4)) bit_set(dado,0);
        output_low(lcd_enable);          // desabilita o display
        return dado;             // retorna o byte lido
}

void lcd_envia_nibble( byte dado )
// envia um dado de quatro bits para o display
{
        // coloca os quatro bits nas saidas
        output_bit(lcd_d4,bit_test(dado,0));
        output_bit(lcd_d5,bit_test(dado,1));
        output_bit(lcd_d6,bit_test(dado,2));
        output_bit(lcd_d7,bit_test(dado,3));
        // dá um pulso na linha enable
        output_high(lcd_enable);
        output_low(lcd_enable);
}


void lcd_envia_byte( boolean endereco, byte dado )
{
        // coloca a linha rs em 0
        output_low(lcd_rs);
        // aguarda o display ficar desocupado
        //while ( bit_test(lcd_le_byte(),7) ) ;
        // configura a linha rs dependendo do modo selecionado
        output_bit(lcd_rs,endereco);
        delay_us(100);          // aguarda 100 us
        // caso a linha rw esteja definida, coloca em 0
        #ifdef lcd_rw
                output_low(lcd_rw);
        #endif
        // desativa linha enable
        output_low(lcd_enable);
        // envia a primeira parte do byte
        lcd_envia_nibble(dado >> 4);
        // envia a segunda parte do byte
        lcd_envia_nibble(dado & 0x0f);
}


void lcd_ini()
// rotina de inicialização do display
{
        byte conta;
        output_low(lcd_d4);
```

```
        output_low(lcd_d5);
        output_low(lcd_d6);
        output_low(lcd_d7);
        output_low(lcd_rs);
        #ifdef lcd_rw
                output_high(lcd_rw);
        #endif
        output_low(lcd_enable);
  while(read_eeprom(0xfd));
        delay_ms(15);
        // envia uma seqüência de 3 vezes 0x03
        // e depois 0x02 para configurar o módulo
        // para modo de 4 bits
        for(conta=1;conta<=3;++conta)
        {
                lcd_envia_nibble(3);
                delay_ms(5);
        }
        lcd_envia_nibble(2);
        // envia string de inicialização do display
        for(conta=0;conta<=3;++conta) lcd_envia_byte(0,INI_LCD[conta]);
}

void lcd_pos_xy( byte x, byte y)
{
  byte endereco;
  if(y!=1)
        endereco = lcd_seg_lin;
  else
        endereco = 0;
  endereco += x-1;
  lcd_envia_byte(0,0x80|endereco);
}

void lcd_escreve( char c)
// envia caractere para o display
{
  switch (c)
        {
    case '\f'       :       lcd_envia_byte(0,1);
                            delay_ms(2);
                            break;
    case '\n':
          case '\r'   :     lcd_pos_xy(1,2);
                            break;
    case '\b'       :       lcd_envia_byte(0,0x10);
                            break;
    default :       lcd_envia_byte(1,c);
                            break;
  }
}

char lcd_le( byte x, byte y)
// le caractere do display
{
        char valor;
        // seleciona a posição do caractere
        lcd_pos_xy(x,y);
        // ativa rs
        output_high(lcd_rs);
        // lê o caractere
        valor = lcd_le_byte();
        // desativa rs
        output_low(lcd_rs);
        // retorna o valor do caractere
        return valor;
}
```

SanUSB.h:

```
//////////////////////http://br.groups.yahoo.com/group/GrupoSanUSB/////////////
#include <18F2550.h>
#device ADC=10
#fuses HSPLL,PLL5, USBDIV,CPUDIV1,VREGEN,NOWDT,NOPROTECT,NOLVP,NODEBUG
//#use delay(clock=48000000)//Frequência padrão da USB
#byte OSCCON=0XFD3
#use delay(clock=4000000) // Clock do oscilador interno do processador de 4MHz
#USE RS232 (BAUD=9600,XMIT=PIN_C6,RCV=PIN_C7)

// Reserva de área da memória flash
#build(reset=0x800)
#build(interrupt=0x808)
#org 0x0000,0x07ff
void SanUsBootloader() {
                        #asm
                        nop
                       #endasm
                       }

void clock_int_4MHz(void)
{
 OSCCON=0B01100110;
 while(read_eeprom(0xfd));
                  }
```

BIBLIOTECA usb_san_cdc.h:

```
//////////////////////////////////////////////////////////////////////////
////                        usb_san_cdc.h
////http://br.groups.yahoo.com/group/GrupoSanUSB


//api for the user:
#define usb_cdc_kbhit() (usb_cdc_get_buffer_status.got)
#define usb_cdc_putready() (usb_cdc_put_buffer_nextin<USB_CDC_DATA_IN_SIZE)
#define usb_cdc_connected() (usb_cdc_got_set_line_coding)
void usb_cdc_putc_fast(char c);
char usb_cdc_getc(void);
void usb_cdc_putc(char c);

//input.c ported to use CDC:
float get_float_usb();
signed long get_long_usb();
signed int get_int_usb();
void get_string_usb(char* s, int max);
BYTE gethex_usb();
BYTE gethex1_usb();

//functions automatically called by USB handler code
void usb_isr_tkn_cdc(void);
void usb_cdc_init(void);
void usb_isr_tok_out_cdc_control_dne(void);
void usb_isr_tok_in_cdc_data_dne(void);
void usb_isr_tok_out_cdc_data_dne(void);

void usb_cdc_flush_out_buffer(void);

//Tells the CCS PIC USB firmware to include HID handling code.
#DEFINE USB_HID_DEVICE  FALSE
#DEFINE USB_CDC_DEVICE  TRUE
```

```
#define USB_CDC_COMM_IN_ENDPOINT       1
#define USB_CDC_COMM_IN_SIZE          8
#define USB_EP1_TX_ENABLE  USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE  USB_CDC_COMM_IN_SIZE

//pic to pc endpoint config
#define USB_CDC_DATA_IN_ENDPOINT       2
#define USB_CDC_DATA_IN_SIZE          64
#define USB_EP2_TX_ENABLE  USB_ENABLE_BULK
#define USB_EP2_TX_SIZE  USB_CDC_DATA_IN_SIZE

//pc to pic endpoint config
#define USB_CDC_DATA_OUT_ENDPOINT       2
#define USB_CDC_DATA_OUT_SIZE          64
#define USB_EP2_RX_ENABLE  USB_ENABLE_BULK
#define USB_EP2_RX_SIZE  USB_CDC_DATA_OUT_SIZE

////////////////////////////////////////////////////////////////////////////
//
// Include the CCS USB Libraries.  See the comments at the top of these
// files for more information
//
////////////////////////////////////////////////////////////////////////////
#ifndef __USB_PIC_PERIF__
#define __USB_PIC_PERIF__            1
#endif

#if __USB_PIC_PERIF__
 #if defined(__PCM__)
  #error CDC requires bulk mode!  PIC16C7x5 does not have bulk mode
 #else
  #include <pic18_usb.h>   //Microchip 18Fxx5x hardware layer for usb.c
 #endif
#else
 #include <usbn960x.c>   //National 960x hardware layer for usb.c
#endif
#include <usb_san_desc.h>   //USB Configuration and Device descriptors for this UBS device
#include <usb.c>        //handles usb setup tokens and get descriptor reports

struct {
       int32   dwDTERrate;   //data terminal rate, in bits per second
       int8    bCharFormat;  //num of stop bits (0=1, 1=1.5, 2=2)
       int8    bParityType;  //parity (0=none, 1=odd, 2=even, 3=mark, 4=space)
       int8    bDataBits;    //data bits (5,6,7,8 or 16)
} usb_cdc_line_coding;

//length of time, in ms, of break signal as we received in a SendBreak message.
//if ==0xFFFF, send break signal until we receive a 0x0000.
int16 usb_cdc_break;

int8 usb_cdc_encapsulated_cmd[8];

int8 usb_cdc_put_buffer[USB_CDC_DATA_IN_SIZE];
int1 usb_cdc_put_buffer_free;
#if USB_CDC_DATA_IN_SIZE>=0x100
 int16 usb_cdc_put_buffer_nextin=0;
// int16 usb_cdc_last_data_packet_size;
#else
 int8 usb_cdc_put_buffer_nextin=0;
// int8 usb_cdc_last_data_packet_size;
#endif
```

```
struct {
  int1 got;
 #if USB_CDC_DATA_OUT_SIZE>=0x100
  int16 len;
  int16 index;
 #else
  int8 len;
  int8 index;
 #endif
} usb_cdc_get_buffer_status;

int8 usb_cdc_get_buffer_status_buffer[USB_CDC_DATA_OUT_SIZE];
#if (defined(__PIC__))
 #if __PIC__
  //#locate usb_cdc_get_buffer_status_buffer=0x500+(2*USB_MAX_EP0_PACKET_LENGTH)+USB_CDC_COMM_IN_SIZE
  #if USB_MAX_EP0_PACKET_LENGTH==8
    #locate usb_cdc_get_buffer_status_buffer=0x500+24
  #elif USB_MAX_EP0_PACKET_LENGTH==64
    #locate usb_cdc_get_buffer_status_buffer=0x500+136
  #else
   #error CCS BUG WONT LET ME USE MATH IN LOCATE
  #endif
 #endif
#endif

int1 usb_cdc_got_set_line_coding;

struct  {
  int1 dte_present; //1=DTE present, 0=DTE not present
  int1 active;     //1=activate carrier, 0=deactivate carrier
  int reserved:6;
} usb_cdc_carrier;

enum       {USB_CDC_OUT_NOTHING=0,       USB_CDC_OUT_COMMAND=1,       USB_CDC_OUT_LINECODING=2,
USB_CDC_WAIT_0LEN=3} __usb_cdc_state=0;

#byte INTCON=0xFF2
#bit INT_GIE=INTCON.7

//handle OUT token done interrupt on endpoint 0 [read encapsulated cmd and line coding data]
void usb_isr_tok_out_cdc_control_dne(void) {
  debug_usb(debug_putc,"CDC %X ",__usb_cdc_state);

  switch (__usb_cdc_state) {
    //printf(putc_tbe,"@%X@\r\n", __usb_cdc_state);
    case USB_CDC_OUT_COMMAND:
      //usb_get_packet(0, usb_cdc_encapsulated_cmd, 8);
      memcpy(usb_cdc_encapsulated_cmd, usb_ep0_rx_buffer,8);
     #if USB_MAX_EP0_PACKET_LENGTH==8
      __usb_cdc_state=USB_CDC_WAIT_0LEN;
      usb_request_get_data();
     #else
      usb_put_0len_0();
      __usb_cdc_state=0;
     #endif
      break;

   #if USB_MAX_EP0_PACKET_LENGTH==8
    case USB_CDC_WAIT_0LEN:
      usb_put_0len_0();
      __usb_cdc_state=0;
      break;
```

```
   #endif

     case USB_CDC_OUT_LINECODING:
       //usb_get_packet(0, &usb_cdc_line_coding, 7);
       //printf(putc_tbe,"\r\n!GSLC FIN!\r\n");
       memcpy(&usb_cdc_line_coding, usb_ep0_rx_buffer,7);
       __usb_cdc_state=0;
       usb_put_0len_0();
       break;

     default:
       __usb_cdc_state=0;
       usb_init_ep0_setup();
       break;
   }
}

//handle IN token on 0 (setup packet)
void usb_isr_tkn_cdc(void) {
  //make sure the request goes to a CDC interface
  if ((usb_ep0_rx_buffer[4] == 1) || (usb_ep0_rx_buffer[4] == 0)) {
    //printf(putc_tbe,"!%X!\r\n", usb_ep0_rx_buffer[1]);
    switch(usb_ep0_rx_buffer[1]) {
      case 0x00:  //send_encapsulated_command
        __usb_cdc_state=USB_CDC_OUT_COMMAND;
        usb_request_get_data();
        break;

      case 0x01:  //get_encapsulated_command
        memcpy(usb_ep0_tx_buffer, usb_cdc_encapsulated_cmd, 8);
        usb_request_send_response(usb_ep0_rx_buffer[6]);  //send wLength bytes
        break;

      case 0x20:  //set_line_coding
        debug_usb(debug_putc,"!GSLC!");
        __usb_cdc_state=USB_CDC_OUT_LINECODING;
        usb_cdc_got_set_line_coding=TRUE;
        usb_request_get_data();
        break;

      case 0x21:  //get_line_coding
        memcpy(usb_ep0_tx_buffer, &usb_cdc_line_coding, sizeof(usb_cdc_line_coding));
        usb_request_send_response(sizeof(usb_cdc_line_coding)); //send wLength bytes
        break;

      case 0x22:  //set_control_line_state
        usb_cdc_carrier=usb_ep0_rx_buffer[2];
        usb_put_0len_0();
        break;

      case 0x23:  //send_break
        usb_cdc_break=make16(usb_ep0_rx_buffer[2],usb_ep0_rx_buffer[3]);
        usb_put_0len_0();
        break;

      default:
        usb_request_stall();
        break;
    }
  }
}
```

```
//handle OUT token done interrupt on endpoint 3 [buffer incoming received chars]
void usb_isr_tok_out_cdc_data_dne(void) {
  usb_cdc_get_buffer_status.got=TRUE;
  usb_cdc_get_buffer_status.index=0;
#if (defined(__PIC__))
  #if __PIC__
   usb_cdc_get_buffer_status.len=usb_rx_packet_size(USB_CDC_DATA_OUT_ENDPOINT);
  #else
   usb_cdc_get_buffer_status.len=usb_get_packet_buffer(
     USB_CDC_DATA_OUT_ENDPOINT,&usb_cdc_get_buffer_status_buffer[0],USB_CDC_DATA_OUT_SIZE);
  #endif
#else
  usb_cdc_get_buffer_status.len=usb_get_packet_buffer(
    USB_CDC_DATA_OUT_ENDPOINT,&usb_cdc_get_buffer_status_buffer[0],USB_CDC_DATA_OUT_SIZE);
#endif
}

//handle IN token done interrupt on endpoint 2 [transmit buffered characters]
void usb_isr_tok_in_cdc_data_dne(void) {
  if (usb_cdc_put_buffer_nextin) {
    usb_cdc_flush_out_buffer();
  }
  //send a 0len packet if needed
//   else if (usb_cdc_last_data_packet_size==USB_CDC_DATA_IN_SIZE) {
//     usb_cdc_last_data_packet_size=0;
//     printf(putc_tbe, "FL 0\r\n");
//     usb_put_packet(USB_CDC_DATA_IN_ENDPOINT,0,0,USB_DTS_TOGGLE);
//   }
  else {
    usb_cdc_put_buffer_free=TRUE;
    //printf(putc_tbe, "FL DONE\r\n");
  }
}

void usb_cdc_flush_out_buffer(void) {
  if (usb_cdc_put_buffer_nextin) {
    usb_cdc_put_buffer_free=FALSE;
    //usb_cdc_last_data_packet_size=usb_cdc_put_buffer_nextin;
    //printf(putc_tbe, "FL %U\r\n", usb_cdc_put_buffer_nextin);
    usb_put_packet(USB_CDC_DATA_IN_ENDPOINT,usb_cdc_put_buffer,usb_cdc_put_buffer_nextin,USB_DTS_TOGGLE);
    usb_cdc_put_buffer_nextin=0;
  }
}

void usb_cdc_init(void) {
  usb_cdc_line_coding.dwDTERrate=9600;
  usb_cdc_line_coding.bCharFormat=0;
  usb_cdc_line_coding.bParityType=0;
  usb_cdc_line_coding.bDataBits=8;
  (int8)usb_cdc_carrier=0;
  usb_cdc_got_set_line_coding=FALSE;
  usb_cdc_break=0;
  usb_cdc_put_buffer_nextin=0;
  usb_cdc_get_buffer_status.got=0;
  usb_cdc_put_buffer_free=TRUE;
  while(read_eeprom(0xfd));
}

////////////////// END USB CONTROL HANDLING ////////////////////////////////

////////////////// BEGIN USB<->RS232 CDC LIBRARY ///////////////////////////
```

```
char usb_cdc_getc(void) {
  char c;

  while (!usb_cdc_kbhit()) {}

  c=usb_cdc_get_buffer_status_buffer[usb_cdc_get_buffer_status.index++];
  if (usb_cdc_get_buffer_status.index >= usb_cdc_get_buffer_status.len) {
    usb_cdc_get_buffer_status.got=FALSE;
    usb_flush_out(USB_CDC_DATA_OUT_ENDPOINT, USB_DTS_TOGGLE);
  }

  return(c);
}

void usb_cdc_putc_fast(char c) {
  int1 old_gie;

  //disable global interrupts
  old_gie=INT_GIE;
  INT_GIE=0;

  if (usb_cdc_put_buffer_nextin >= USB_CDC_DATA_IN_SIZE) {
    usb_cdc_put_buffer_nextin=USB_CDC_DATA_IN_SIZE-1;  //we just overflowed the buffer!
  }
  usb_cdc_put_buffer[usb_cdc_put_buffer_nextin++]=c;

  //renable global interrupts
  INT_GIE=old_gie;

  /*
  if (usb_tbe(USB_CDC_DATA_IN_ENDPOINT)) {
    if (usb_cdc_put_buffer_nextin)
      usb_cdc_flush_out_buffer();
  }
  */
  if (usb_cdc_put_buffer_free) {
    usb_cdc_flush_out_buffer();
  }
}

void usb_cdc_putc(char c) {
  while (!usb_cdc_putready()) {
    if (usb_cdc_put_buffer_free) {
      usb_cdc_flush_out_buffer();
    }
    //delay_ms(500);
    //printf(putc_tbe,"TBE=%U                CNT=%U                LST=%U\r\n",usb_tbe(USB_CDC_DATA_IN_ENDPOINT),
usb_cdc_put_buffer_nextin, usb_cdc_last_data_packet_size);
  }
  usb_cdc_putc_fast(c);
}

#include <ctype.h>

BYTE gethex1_usb() {
  char digit;

  digit = usb_cdc_getc();

  usb_cdc_putc(digit);

  if(digit<='9')
```

```
    return(digit-'0');
  else
    return((toupper(digit)-'A')+10);
}

BYTE gethex_usb() {
  int lo,hi;

  hi = gethex1_usb();
  lo = gethex1_usb();
  if(lo==0xdd)
    return(hi);
  else
    return( hi*16+lo );
}

void get_string_usb(char* s, int max) {
  int len;
  char c;

  --max;
  len=0;
  do {
    c=usb_cdc_getc();
    if(c==8) {  // Backspace
      if(len>0) {
        len--;
        usb_cdc_putc(c);
        usb_cdc_putc(' ');
        usb_cdc_putc(c);
      }
    } else if ((c>=' ')&&(c<='~'))
      if(len<max) {
        s[len++]=c;
        usb_cdc_putc(c);
      }
  } while(c!=13);
  s[len]=0;
}


// stdlib.h is required for the ato_ conversions
// in the following functions
#ifdef _STDLIB

signed int get_int_usb() {
  char s[5];
  signed int i;

  get_string_usb(s, 5);

  i=atoi(s);
  return(i);
}

signed long get_long_usb() {
  char s[7];
  signed long l;

  get_string_usb(s, 7);
  l=atol(s);
  return(l);
```

```
}

float get_float_usb() {
  char s[20];
  float f;

  get_string_usb(s, 20);
  f = atof(s);
  return(f);
}

#endif
```

BIBLIOTECA usb_san_desc.h:

```
//////////////////////////////////////////////////////////////////////
////                     usb_desc_cdc.h                           ////
//////////////////////////////////////////////////////////////////////

#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>


/////////////////////////////////////////////////////////////////
///
///  start config descriptor
///  right now we only support one configuration descriptor.
///  the config, interface, class, and endpoint goes into this array.
///
/////////////////////////////////////////////////////////////////

  #DEFINE USB_TOTAL_CONFIG_LEN      67  //config+interface+class+endpoint+endpoint (2 endpoints)

  const char USB_CONFIG_DESC[] = {
  //IN ORDER TO COMPLY WITH WINDOWS HOSTS, THE ORDER OF THIS ARRAY MUST BE:
     //    config(s)
     //    interface(s)
     //    class(es)
     //    endpoint(s)

  //config_descriptor for config index 1
       USB_DESC_CONFIG_LEN, //length of descriptor size          ==0
       USB_DESC_CONFIG_TYPE, //constant CONFIGURATION (CONFIGURATION 0x02)     ==1
       USB_TOTAL_CONFIG_LEN,0, //size of all data returned for this config      ==2,3
       2, //number of interfaces this device supports       ==4
       0x01, //identifier for this configuration.  (IF we had more than one configurations)      ==5
       0x00, //index of string descriptor for this configuration      ==6
       0xC0, //bit 6=1 if self powered, bit 5=1 if supports remote wakeup (we don't), bits 0-4 unused and bit7=1         ==7
       0x32, //maximum bus power required (maximum milliamperes/2)  (0x32 = 100mA)  ==8

  //interface descriptor 0 (comm class interface)
       USB_DESC_INTERFACE_LEN, //length of descriptor      =9
       USB_DESC_INTERFACE_TYPE, //constant INTERFACE (INTERFACE 0x04)       =10
       0x00, //number defining this interface (IF we had more than one interface)    ==11
       0x00, //alternate setting     ==12
       1, //number of endpoints   ==13
       0x02, //class code, 02 = Comm Interface Class     ==14
       0x02, //subclass code, 2 = Abstract     ==15
       0x01, //protocol code, 1 = v.25ter      ==16
       0x00, //index of string descriptor for interface      ==17

  //class descriptor [functional header]
       5, //length of descriptor    ==18
       0x24, //dscriptor type (0x24 == )      ==19
       0, //sub type (0=functional header) ==20
       0x10,0x01, //      ==21,22 //cdc version
```

```
  //class descriptor [acm header]
      4, //length of descriptor   ==23
      0x24, //dscriptor type (0x24 == )     ==24
      2, //sub type (2=ACM)  ==25
      2, //capabilities   ==26  //we support Set_Line_Coding, Set_Control_Line_State, Get_Line_Coding, and the notification Serial_State.

  //class descriptor [union header]
      5, //length of descriptor   ==27
      0x24, //dscriptor type (0x24 == )     ==28
      6, //sub type (6=union)   ==29
      0, //master intf      ==30  //The interface number of the Communication or Dat a Cl ass interface, designated as the masteror controlling
interface for the union.
      1, //save intf0     ==31  //Interface number of first slave or associated interface in the union. *

  //class descriptor [call mgmt header]
      5, //length of descriptor   ==32
      0x24, //dscriptor type (0x24 == )     ==33
      1, //sub type (1=call mgmt)  ==34
      0, //capabilities         ==35  //device does not handle call management itself
      1, //data interface       ==36  //interface number of data class interface

  //endpoint descriptor
      USB_DESC_ENDPOINT_LEN, //length of descriptor                ==37
      USB_DESC_ENDPOINT_TYPE, //constant ENDPOINT (ENDPOINT 0x05)        ==38
      USB_CDC_COMM_IN_ENDPOINT | 0x80, //endpoint number and direction
      0x03, //transfer type supported (0x03 is interrupt)        ==40
      USB_CDC_COMM_IN_SIZE,0x00, //maximum packet size supported              ==41,42
      250,  //polling interval, in ms.  (cant be smaller than 10)     ==43

  //interface descriptor 1 (data class interface)
      USB_DESC_INTERFACE_LEN, //length of descriptor     =44
      USB_DESC_INTERFACE_TYPE, //constant INTERFACE (INTERFACE 0x04)      =45
      0x01, //number defining this interface (IF we had more than one interface)   ==46
      0x00, //alternate setting    ==47
      2, //number of endpoints  ==48
      0x0A, //class code, 0A = Data Interface Class    ==49
      0x00, //subclass code     ==50
      0x00, //protocol code     ==51
      0x00, //index of string descriptor for interface     ==52

  //endpoint descriptor
      USB_DESC_ENDPOINT_LEN, //length of descriptor                ==60
      USB_DESC_ENDPOINT_TYPE, //constant ENDPOINT (ENDPOINT 0x05)        ==61
      USB_CDC_DATA_OUT_ENDPOINT, //endpoint number and direction (0x02 = EP2 OUT)      ==62
      0x02, //transfer type supported (0x02 is bulk)        ==63
//        make8(USB_CDC_DATA_OUT_SIZE,0),make8(USB_CDC_DATA_OUT_SIZE,1), //maximum packet size supported                ==64,
65
      USB_CDC_DATA_OUT_SIZE  &  0xFF,  (USB_CDC_DATA_OUT_SIZE  >>  8)  &  0xFF,  //maximum  packet  size  supported
==64, 65
      250,  //polling interval, in ms.  (cant be smaller than 10)     ==66

  //endpoint descriptor
      USB_DESC_ENDPOINT_LEN, //length of descriptor                ==53
      USB_DESC_ENDPOINT_TYPE, //constant ENDPOINT (ENDPOINT 0x05)        ==54
      USB_CDC_DATA_IN_ENDPOINT | 0x80, //endpoint number and direction (0x82 = EP2 IN)      ==55
      0x02, //transfer type supported (0x02 is bulk)        ==56
//        make8(USB_CDC_DATA_IN_SIZE,0),make8(USB_CDC_DATA_IN_SIZE,1), //maximum packet size supported               ==57, 58
      USB_CDC_DATA_IN_SIZE & 0xFF, (USB_CDC_DATA_IN_SIZE >> 8) & 0xFF, //maximum packet size supported               ==64, 65
      250,  //polling interval, in ms.  (cant be smaller than 10)     ==59
  };

  //****** BEGIN CONFIG DESCRIPTOR LOOKUP TABLES ********
  //since we can't make pointers to constants in certain pic16s, this is an offset table to find
  //  a specific descriptor in the above table.

  //the maximum number of interfaces seen on any config
  //for example, if config 1 has 1 interface and config 2 has 2 interfaces you must define this as 2
  #define USB_MAX_NUM_INTERFACES   2
```

```
  //define how many interfaces there are per config.  [0] is the first config, etc.
  const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={2};

  //define where to find class descriptors
  //first dimension is the config number
  //second dimension specifies which interface
  //last dimension specifies which class in this interface to get, but most will only have 1 class per interface
  //if a class descriptor is not valid, set the value to 0xFFFF
  const int16 USB_CLASS_DESCRIPTORS[USB_NUM_CONFIGURATIONS][USB_MAX_NUM_INTERFACES][4]=
  {
  //config 1
    //interface 0
      //class 1-4
      18,23,27,32,
    //interface 1
      //no classes for this interface
      0xFFFF,0xFFFF,0xFFFF,0xFFFF
  };

  #if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
    #error USB_TOTAL_CONFIG_LEN not defined correctly
  #endif


//////////////////////////////////////////////////////////////////
///
///   start device descriptors
///
//////////////////////////////////////////////////////////////////

  const char USB_DEVICE_DESC[USB_DESC_DEVICE_LEN] ={
    //starts of with device configuration. only one possible
      USB_DESC_DEVICE_LEN, //the length of this report   ==0
      0x01, //the constant DEVICE (DEVICE 0x01)  ==1
      0x10,0x01, //usb version in bcd  ==2,3
      0x02, //class code. 0x02=Communication Device Class ==4
      0x00, //subclass code ==5
      0x00, //protocol code ==6
      USB_MAX_EP0_PACKET_LENGTH, //max packet size for endpoint 0. (SLOW SPEED SPECIFIES 8) ==7

      0xD8,0x04,          //vendor id (0x04D8 is Microchip)
      0x0A,0x00,          //product id

// RR2 cambiado para       0x61,0x04, //vendor id (0x04D8 is Microchip, or is it 0x0461 ??)  ==8,9
// compatibilidad con .inf 0x33,0x00, //product id   ==10,11
// de Microchip

      0x00,0x01, //device release number  ==12,13
      0x01, //index of string description of manufacturer. therefore we point to string_1 array (see below)  ==14
      0x02, //index of string descriptor of the product  ==15
      0x00, //index of string descriptor of serial number  ==16
      USB_NUM_CONFIGURATIONS  //number of possible configurations  ==17
  };



//the offset of the starting location of each string.  offset[0] is the start of string 0, offset[1] is the start of string 1, etc.
char USB_STRING_DESC_OFFSET[]={0,4,12};

char const USB_STRING_DESC[]={
  //string 0
      4, //length of string index
      USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
      0x09,0x04,   //Microsoft Defined for US-English
  //string 1
      8,
      USB_DESC_STRING_TYPE,
      112,0,105,0,99,0,
  //string 2
      32,
```

```
        USB_DESC_STRING_TYPE,
        115,0,97,0,110,0,100,0,114,0,111,0,' ',
        0,106,0,117,0,99,0,97,0,' ',
        0,117,0,115,0,98,0
};

#ENDIF
```