8,0
(oito)

ERICK DARÍO LEÓN BUENO DE CAMARGO

DESENVOLVIMENTO DE INTERFACE IRDA MICROPROCESSADA

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro

Área de Engenharia Mecatrônica e Sistemas Mecânicos

Orientador:
Prof. Dr. Jun Okamoto Júnior

São Paulo
2002

Aos meus pais, por sempre me mostrarem qual o caminho certo a seguir, e por me ajudarem sempre que precisei.

AGRADECIMENTOS

Ao professor doutor Jun Okamoto Jr., pela orientação e paciência que tornaram possível a conclusão deste trabalho.

Ao Marcus Grieneisen Pivatto, pela ajuda na programação do microcontrolador, no projeto da parte eletrônica do dispositivo e na fabricação da placa.

À Maria Angélica Cravo Dias Pereira, pela ajuda na programação do microcotrolador.

Ao Valdir Grassi Júnior, pela ajuda na comunicação com o robô e na programação do microcontrolador.

À Graftec Eletronics Sales, Inc., por fornecer gratuitamente amostras do transceiver IrDA para o desenvolvimento deste projeto.

À todos aqueles que ajudaram direta ou indiretamente no desenvolvimento deste trabalho.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

RESUMO

O trabalho visa o desenvolvimento e construção de um dispositivo microcontrolado capaz de receber sinais por uma porta infravermelha, provenientes de um controle remoto de TV, celular, PDA ou outro dispositivo compatível com IrDA, interpretá-los e enviar sinais de comando através de uma porta serial RS232 a um robô desenvolvido no Departamento. Esses sinais podem ser desde simples comandos como "avanço" até seqüências de comandos, como percorrer uma trajetória pré-determinada. Inclui ainda a programação de um software em PalmOS para envio de sinais infravermelhos ao dispositivo.

O dispositivo deverá conter um microcontrolador, que poderá ser conectado diretamente a um transceiver, e uma UART implementada, para fazer a interface RS232.

ABSTRACT

The project consists of development and construction of a microcontrolled device capable of receiving infrared signals, from a remote control, a PDA or another IrDA compatible device, and sending control commands, through a RS232 serial port to a robot developed at USP. These signals can be single commands or a sequence of multiple commands. The project also includes programming a software for PalmOS to send infrared data to the device.

The device will have a microcontroller, witch can be connected directly to a transceiver, and an implemented UART, for the RS232 interface.

I – INTRODUÇÃO

Este trabalho consiste no projeto e construção de uma interface microprocessada entre sinais seriais RS232, padrão nos PCs atuais, e sinais infravermelhos compatíveis com IrDA (Infrared Data Association), amplamente utilizados por celulares, computadores portáteis, impressoras, PDAs (Personal Data Assistants) e outros dispositivos capazes de realizar comunicação wireless, para enviar sinais de comando a um robô desenvolvido no Departamento. Esse robô já possui uma porta RS232 habilitada para receber comandos de direção e velocidade.

Sendo assim, o trabalho envolve o estudo de dois tipos de comunicação amplamente utilizados na industria atual, a escolha e implementação de um microcontrolador, além das programações do mesmo e de um PDA para envio de sinais infravermelho, e o projeto e construção de um circuito elétrico, atingindo várias áreas de interesse no estudo da engenharia mecatrônica.

II – REVISÃO DA LITERATURA

1 - COMUNICAÇÃO SERIAL – RS232

O padrão RS232 descreve um método de comunicação capaz de operar em diferentes condições de operação, podendo, por exemplo, operar numa grande faixa de voltagens. Na especificação original, as possibilidades técnicas daquela época foram levadas em consideração, como por exemplo, na taxa máxima de transferência, de 20 kbps. Com os componentes atuais, como o 16550A UART, velocidades de 1,5 Mbps são permitidas. As especificações elétricas estão definidas na EIA (Eletronics Industry Association).

1.1 – Sinal:

O nível do sinal dos pinos do RS232 podem apresentar dois estados. O nível 'HIGH' é identificado por uma voltagem negativa, e o nível 'LOW' é identificado por uma voltagem positiva. Os limites de voltagem são mostrados na tabela 1.

Nível	Transmissor (V)	Receptor (V)
Lógico '0'	+5 ... +15	+3 ... +25
Lógico '1'	-5 ... -15	-3 ... -25
Indefinido	-	-3 ... +3

Tabela 1 - Valores de voltagem do RS232

Apesar das altas voltagens presentes, não é possível danificar a porta serial com um pequeno curto-circuito. Apenas aplicando voltagens externas com altas correntes pode-se eventualmente queimar os chips do driver.

1.2 – Velocidades entre DTE / DCE:

Os dispositivos que utilizam cabos seriais para comunicação podem ser divididos em dois grupos: DTE (Data Terminal Equipment) e DCE (Data Communications Equipment). Um exemplo típico de DTE é o computador, e de DCE é o modem. Assim temos dois valores de velocidades: entre DTE e DCE e entre DCE e DCE. A velocidade DTE-DCE é a velocidade de comunicação entre o computador e o modem, e a velocidade DCE-DCE é a velocidade de comunicação entre dois modems, também chamada de velocidade da linha.

Atualmente é comum encontrar dispositivos e softwares de comunicação com compressão de dados. Com isso, em um modem de 28,8 kbps e uma taxa de compressão de aproximadamente 1 : 4, pode-se transmitir um arquivo de texto a uma velocidade de 115,2 kbps, sendo então necessária uma velocidade DTE-DCE de 115,2 kpbs, mesmo com a velocidade DCE-DCE de 28,8 kbps. Por esse motivo a velocidade DTE-DCE deve ser muito maior que a velocidade DCE-DCE.

1.3 – Conector RS232:

O conector RS232 foi originalmente projetado para usar 25 pinos. Nessa pinagem foram feitas provisões para um segundo canal de comunicação. Na prática, apenas um canal de comunicação está presente. Por essa razão, a versão menor, de 9 pinos, é mais comum hoje em dia. As figuras 1 e 2 mostram a pinagem desses dois tipos de conectores, e a função dos pinos está descrita na tabela 2.

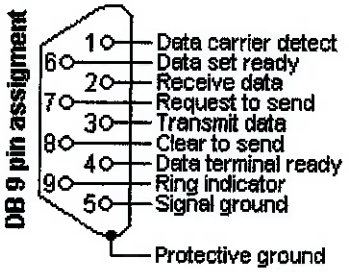


Figura 1 - Conector DB9

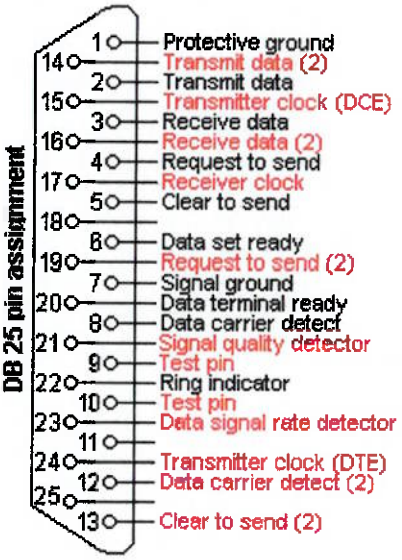


Figura 2 - Conector DB25

Abreviação	Nome Completo	Função
TD	Transmit Data	Saída de Dados Seriais (TXD).
RD	Receive Data	Entrada de Dados Seriais (RXD).
CTS	Clear to Send	Esta linha indica que o modem está pronto para trocar dados.
DCD	Data Carrier Detected	Quando o modem detecta um "carrier" do modem do outro lado da linha de telefone, esta linha se torna ativa.
DSR	Data Set Ready	Esta linha diz à UART que o modem está pronto para estabelecer uma conexão.
DTR	Data Terminal Ready	Esta linha diz ao modem que a UART está pronta para estabelecer uma conexão.
RTS	Request to Send	Esta linha informa ao modem que a UART está pronta para trocar dados.
RI	Ring Indicator	Esta linha se torna ativa quando o modem detecta uma chamada do PSTN (Public Switched Telephone Network).

Tabela 2 - Funções dos Pinos do RS232

1.4 – UART:

Um UART (Universal Asynchronous Receiver / Transmitter) é o responsável por executar as principais tarefas em uma comunicação serial. Esse dispositivo converte a informação recebida em paralelo em informação serial, a qual pode ser enviada para a linha de comunicação. Um segundo UART pode ser usado para receber informações. O UART executa todas as tarefas, temporização, checagem de paridade etc, necessárias para a comunicação. Os únicos dispositivos extras são chips de drivers capazes de transformar sinais de nível TTL em voltagens de linha e vice-versa. A comunicação entre o processador e a UART é controlada completamente por 12 registradores. Estes registradores podem ser lidos e escritos para checar ou mudar o comportamento do dispositivo de comunicação. Cada registrador tem um tamanho de 8 bits.

1.5 – Microcontroladores:

Também é possível utilizar microcontroladores para transmitir e receber dados seriais. Alguns microcontroladores possuem, entre outras coisas, UARTs implementadas. Muitos microcontroladores, como o 68HC05J1A e o PIC16C84, têm um número menor de pinos comparados a UART de 40 pinos. Isso não só torna o projeto menor em tamanho, como também reduz a complexidade do circuito da placa.

1.6 – Caracteres ASCII:

Os caracteres ASCII têm sido adotados como o padrão para troca de informações. Os primeiros 32 caracteres e o último são códigos de controle, e os outros são caracteres imprimíveis. A tabela 3 mostra o conjunto de caracteres ASCII e a tabela 4 mostra o significado dos códigos de controle.

	0	1	2	3	4	5	6	7
0	<nul>	<soh>	<stx>	<etx>	<eot>	<enq>	<ack>	<bel>
8	<bs>	<ht>	<lf>	<vt>	<ff>	<cr>	<so>	<si>
16	<dle>	<dc1>	<dc2>	<dc3>	<dc4>	<nak>	<syn>	<etb>
24	<can>		<sub>	<esc>	<fs>	<gs>	<rs>	<us>
32	<sp>	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7
56	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G
72	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W
88	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g
104	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	<del>

Tabela 3 - Conjunto de caracteres ASCII

Valor	Nome	Comentário
0	<nul>	Null
1	<soh>	Start of header
2	<stx>	Start of text
3	<etx>	End of text
4	<eot>	End of transmission
5	<enq>	Enquiry
6	<ack>	Acknowledgment
7	<bel>	Bell
8	<bs>	Backspace
9	<ht>	Horizontal tab
10	<lf>	Line feed
11	<vt>	Vertical tab
12	<ff>	Form feed
13	<cr>	Carriage return
14	<so>	Shift out
15	<si>	Shift in
16	<dle>	Data link escape
17	<dc1>	Device control 1
18	<dc2>	Device control 2
19	<dc3>	Device control 3
20	<dc4>	Device control 4
21	<nak>	Negative acknowledgment
22	<syn>	Synchronous idle
23	<etb>	End of transmission block
24	<can>	Cancel
25		End of medium
26	<sub>	Substitute character

27	<esc>	Escape
28	<fs>	File separator
29	<gs>	Group separator
30	<rs>	Record separator
31	<us>	Unit separator
127	<del>	Delete

Tabela 4 - Códigos de Controle

2 – COMUNICAÇÃO POR INFRAVERMELHO – IrDA

2.1 – Introdução

2.1.1 – IrDA:

Assim que comunicação de dados por infravermelho, baseada no padrão da IrDA (*Infrared Data Association*), se tornou amplamente disponível em computadores pessoais e periféricos, passou a existir uma grande oportunidade de comunicação sem fio em sistemas incorporados e dispositivos de todos os tipos que seja efetiva e barata.

A *Infrared Data Association* (IrDA) é um grupo industrial de mais de 160 companhias que desenvolveram padrões de comunicação especialmente para baixo custo, baixo alcance, plataforma cruzada, comunicação ponto a ponto em uma grande faixa de velocidades. Esses padrões têm sido implementados em várias plataformas computacionais e, mais recentemente, estão disponíveis em várias aplicações incorporadas.

O padrão atual da camada física do IrDA está na versão 1.3 (março de 2001). A versão 1.4 está prevista para breve e adicionará o padrão de 16 Mbit/s o qual, atualmente, é coberto por outra documentação. A versão 1.4 substitui a versão 1.3. As versões 1.0 a 1.2 são versões obsoletas e hoje apenas descrevem passos históricos do desenvolvimento da IrDA.

2.1.2 – Funcionamento da IrDA:

A transmissão em um modo compatível com IrDA (algumas vezes chamada SIR, serial infrared) usa, no caso mais simples, a porta RS232, um padrão incluído em todos os computadores compatíveis com PC. Com uma interface simples, encurtando o comprimento do pulso para um máximo de 3/16 do comprimento original para atender requisitos de baixo consumo, um diodo emissor de infravermelho é usado para transmitir um sinal óptico para o receptor.

Esse tipo de comunicação cobre a faixa de transmissão de dados de até 115,2 kbit/s, a qual é a máxima taxa de transmissão suportada por uma UART padrão. A velocidade de transmissão mínima é de 9600 bit/s. Todas as transmissões devem ser começadas nesta freqüência para habilitar compatibilidade. Velocidades mais altas são uma questão de negociação entre as portas depois de estabelecida a conexão.

Velocidades mais altas necessitam de interfaces especiais que operam a 1.152 Mbit/s e usam um processo de encurtamento do pulso similar ao do modo relacionado a RS232, mas com uma redução do pulso de 1/4 do comprimento original. A taxa máxima de dados suportada pela IrDA é de 4.0 Mbit/s (geralmente chamada de FIR, fast infrared), operando com pulsos de 125 ns em um modo 4 PPM (Pulse-Position Modulation). Uma interface típica para os vários modos é mostrada na Figura 3.

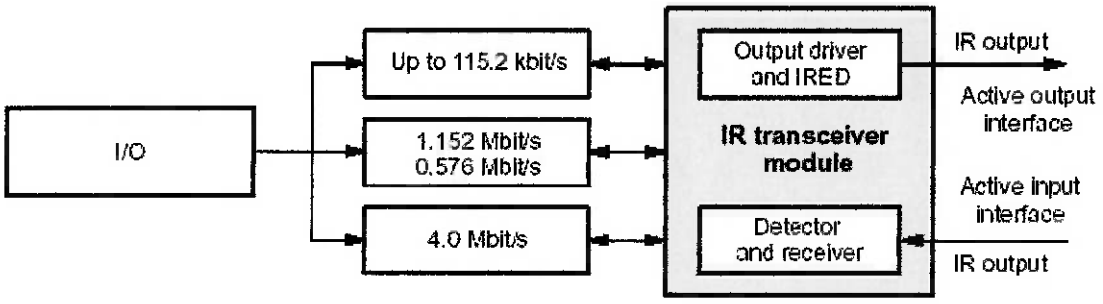


Figura 3 - Diagrama de bloco de uma ponta da conexão para taxas de até 4 Mbit/s

A potência de saída e a sensibilidade de entrada são configuradas para um nível no qual uma atividade apontar-e-disparar (± 15°) é suficiente para comunicação ponto a ponto, mas previne a poluição do ambiente não desperdiçando energia. Transmissão até uma distância de 1 m é garantida. O detector recebe o sinal transmitido, reformata o sinal e o manda para a porta. O sistema trabalha num modo half-duplex, o qual permite apenas uma direção de transmissão estar ativa por vez.

Para freqüências de até 115,2 kbit/s, a intensidade de saída mínima é de 40 mW/sr. Para velocidades mais altas, uma intensidade mínima maior, de 100 mW/sr, é usada. Conseqüentemente, as sensitividades de 40 mW/m^2 e 100 mW/m^2 são usadas, respectivamente, para SIR e FIR.

O comprimento de onda escolhido para o padrão é entre 850 nm e 900 nm.

O padrão VFIR (Very Fast Infrared, 16 Mbit/s para mais de 1 m) foi estabelecido em 1999.

O modo mais simples de interfaceamento óptico no modo SIR é mostrado na Figura 4. Deve ser usado um dispositivo para reformatar e recuperar pulsos, como o TOIM3000 ou TOIM3232 da Vishay Telefunken. A transmissão e recepção devem ser realizadas, por exemplo, por um transceiver, como o TFDS4500. Um amplificador é usado no receptor para amplificar a entrada. Sua saída é alimentada na entrada de um comparador, cujo nível de referência é ajustado para suprimir ruído e interferência do ambiente eficientemente.

Adicionalmente, um circuito formatador de pulso deve ser inserido para encurtar o pulso para ser emitido em 1.6 µs (3/16 do comprimento do bit a 115 kbit/s) e recuperar pulsos do sinal detectado para atender o padrão IrDA. Apenas os bits 'LOW' (0) são transmitidos. Em alguns transceivers, esses circuitos já vem implementados, como é o caso do ZHX1810, da Zilog, mostrado na Figura 5. Com esse transceiver é possível montar um circuito ligando-o diretamente no microcontrolador, como mostra a Figura 6.

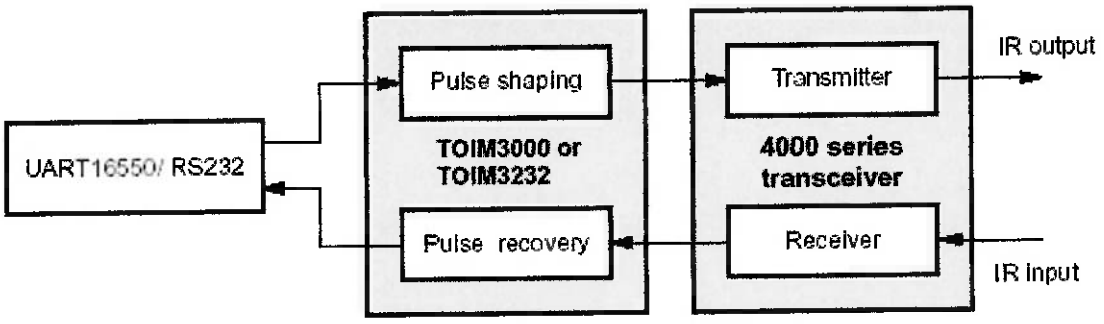


Figura 4 - Diagrama de bloco de uma ponta de uma conexão

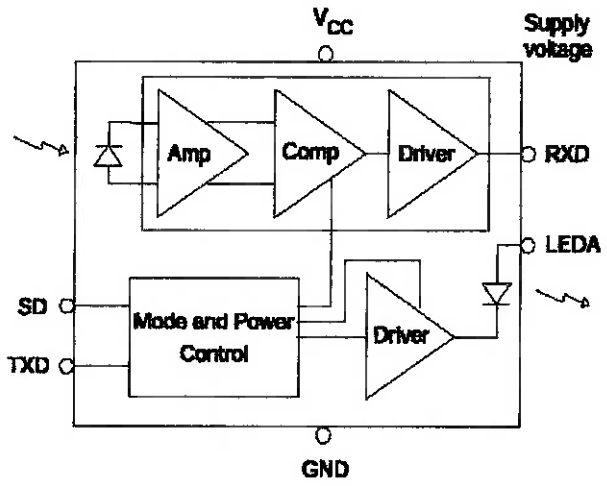


Figura 5 - Diagrama de bloco do ZHX1810

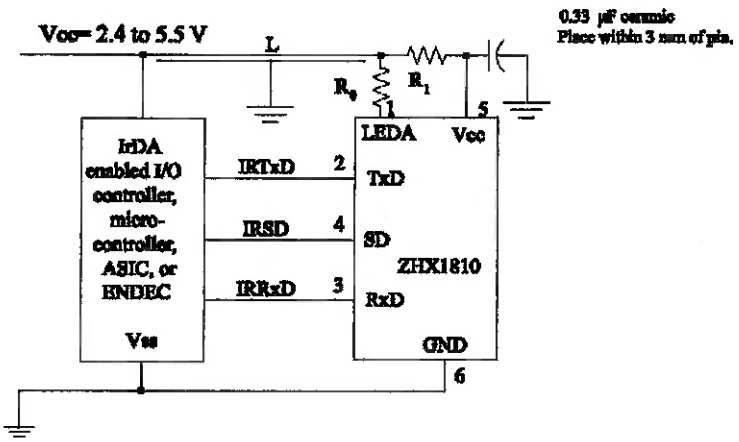


Figura 6 - Circuito com um ZHX1810

2.2 – Especificações

Os protocolos de comunicação lidam com vários assuntos, e por isso são geralmente divididos em camadas, onde cada camada lida com um conjunto de responsabilidades e fornece capacidades necessárias às camadas abaixo e acima. Quando as camadas são colocadas uma sobre as outras, temos o que é chamada uma pilha de protocolos. Uma pilha de protocolos da IrDA é dividida em camadas de conjuntos de protocolos destinados particularmente à comunicação infravermelho ponto a ponto e às aplicações necessárias nesse ambiente. Na Figura 7 temos as camadas de protocolos da IrDA.

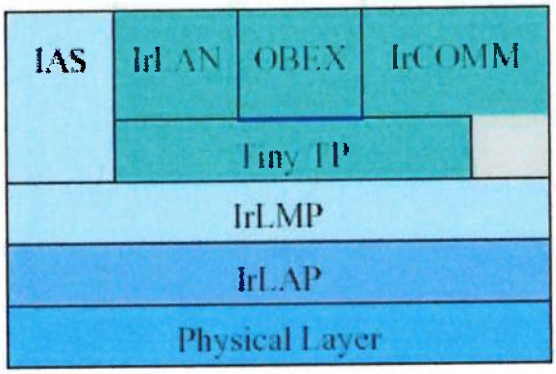


Figura 7 - Protocolos da IrDA

Os protocolos desta pilha podem ser divididos em dois grupos: protocolos necessários e opcionais.

Os protocolos necessários incluem os seguintes:

- ✓ Camada física: especifica características ópticas, codificação de dados, e separação em blocos de dados para transmissão em várias velocidades;
- ✓ IrLAP: Link Access Protocol. Estabelece a conexão básica;
- ✓ IrLMP: Link Management Protocol. Multiplexa serviços e aplicações na conexão IrLAP;
- ✓ IAS: Information Access Service. Fornece uma "páginas amarelas" de serviços em um dispositivo.

Os protocolos opcionais são:

- ✓ TinyTP: Tiny Transport Protocol. Adiciona controle de fluxo por canal. Esta é uma função muito importante e é necessária em muitos casos;
- ✓ IrOBEX: Object Exchange Protocol. Transferência fácil de arquivos e outros objetos de dados;
- ✓ IrCOMM: emulação de portas paralela e serial, habilitando aplicações existentes que usam portas paralela e serial a usar infravermelho sem mudanças.
- ✓ IrLAN: acesso à LAN (Local Area Network).

Quando essas camadas são integradas a um sistema incorporado, elas podem ser dispostas como mostra a Figura 8.

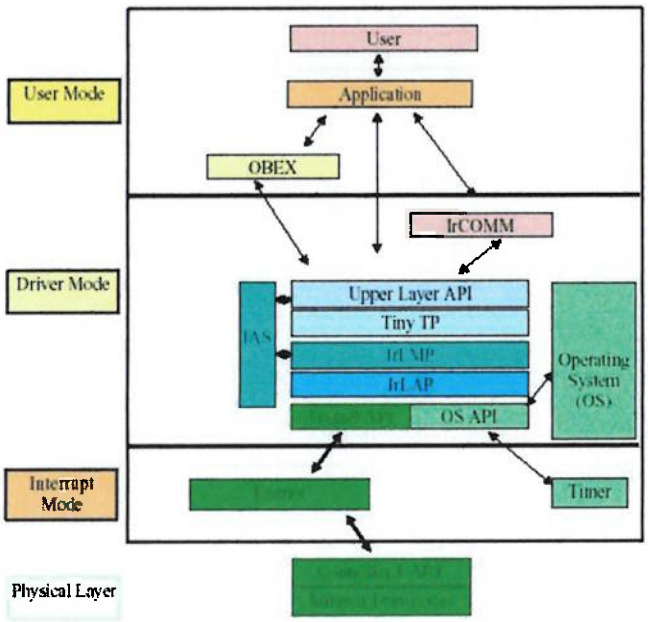


Figura 8 - Disposição das camadas em um sistema

2.2.1 – Camada física:

No modo SIR, os dados são representados por pulsos ópticos entre 1.6 μs e 3/16 do tamanho do bit de um pulso de uma RS232, como mostra a Figura 9. Essa redução de pulsos também é aplicada a modos de freqüências mais altas, e os limites da duração dos pulsos são mostrados na Tabela 5. Uma interface serial infravermelha IrDA deve operar a 9.6 kbit/s. As velocidades adicionais são opcionais.

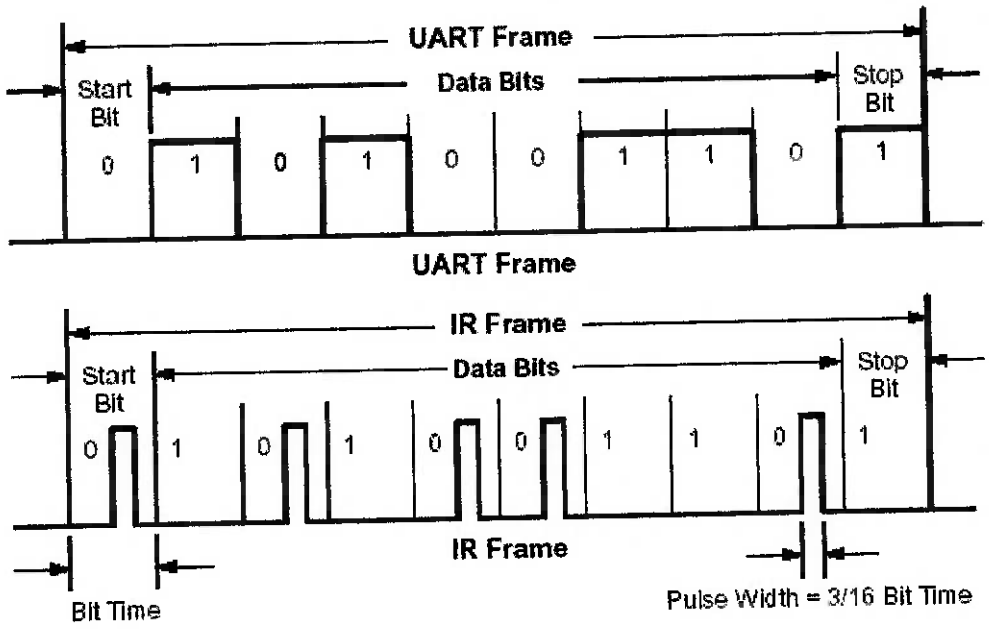


Figura 9 - Formato do frame em UART e infravermelho

Freqüência do Sinal	Tolerância % da freqüência	Duração mínima do pulso	Duração Nominal do pulso	Duração máxima do pulso
2.4 kbit/s	±0.87	1.41 µs	78.13 µs	88.55 µs
9.6 kbit/s	**±0.87**	**1.41 µs**	**19.53 µs**	**22.13 µs**
19.2 kbit/s	±0.87	1.41 µs	9.77 µs	11.07 µs
38.4 kbit/s	±0.87	1.41 µs	4.88 µs	5.96 µs
57.6 kbit/s	±0.87	1.41 µs	3.26 µs	4.34 µs
115.2 kbit/s	±0.87	1.41 µs	1.63 µs	2.23 µs
0.576 Mbit/s	±0.1	295.2 ns	434.0 ns	520.8 ns
1.152 Mbit/s	±0.1	147.6 ns	217.0 ns	260.4 ns

Tabela 5 - Especificação da freqüência e duração do sinal

A intensidade da radiação e a sensibilidade são ajustadas para garantir uma transmissão ponto-a-ponto em um cone de ±15° a uma distância de pelo menos 1m. O BER (Bit Error Ratio) não deve ultrapassar 10^{-8}, e é medido conforme teste

definido na Especificação da Camada Física da IrDA. A conexão deve operar e obedecer à especificação para o BER no alcance especificado. A intensidade de radiação óptica é limitada a um máximo de 500 mW/sr e um ângulo de ±30° a fim de habilitar operações independentes de mais de um sistema em uma sala. O campo de tolerância da emissão de um transmissor é mostrada na Figura 10.

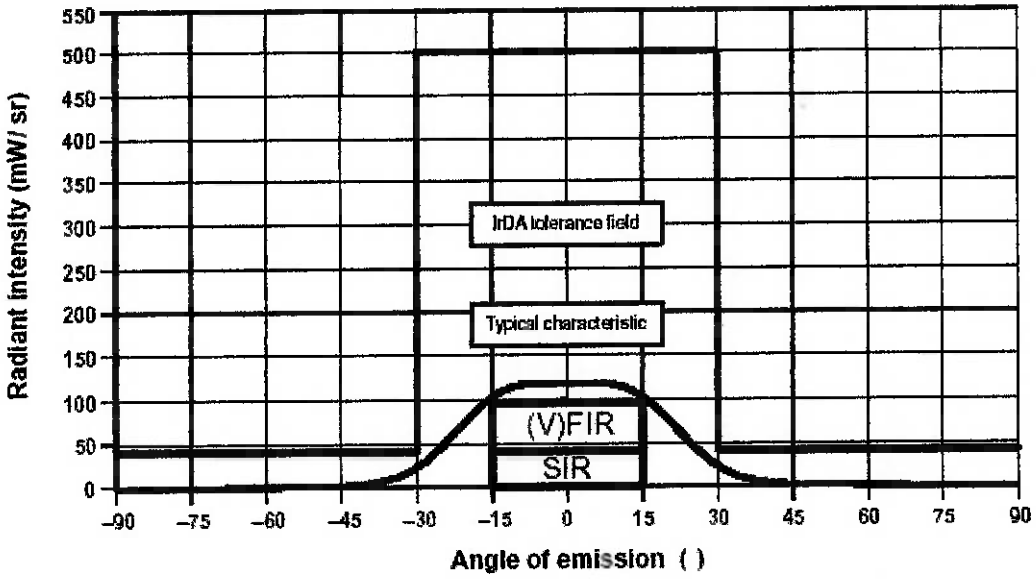


Figura 10 - Campo de tolerância de uma emissão angular

A distância máxima da transmissão como função da sensibilidade (irradiação necessária no detector) é mostrada na Figura 11. Por exemplo, uma sensibilidade de 40 mW/m^2, combinada com uma intensidade de 40 mW/sr, resulta em uma distância de emissão de 1 metro. A combinação de um detector com uma irradiação mínima de 10 mW/m^2 e um emissor com 250 mW/sr pode transmitir por quase 5 metros.

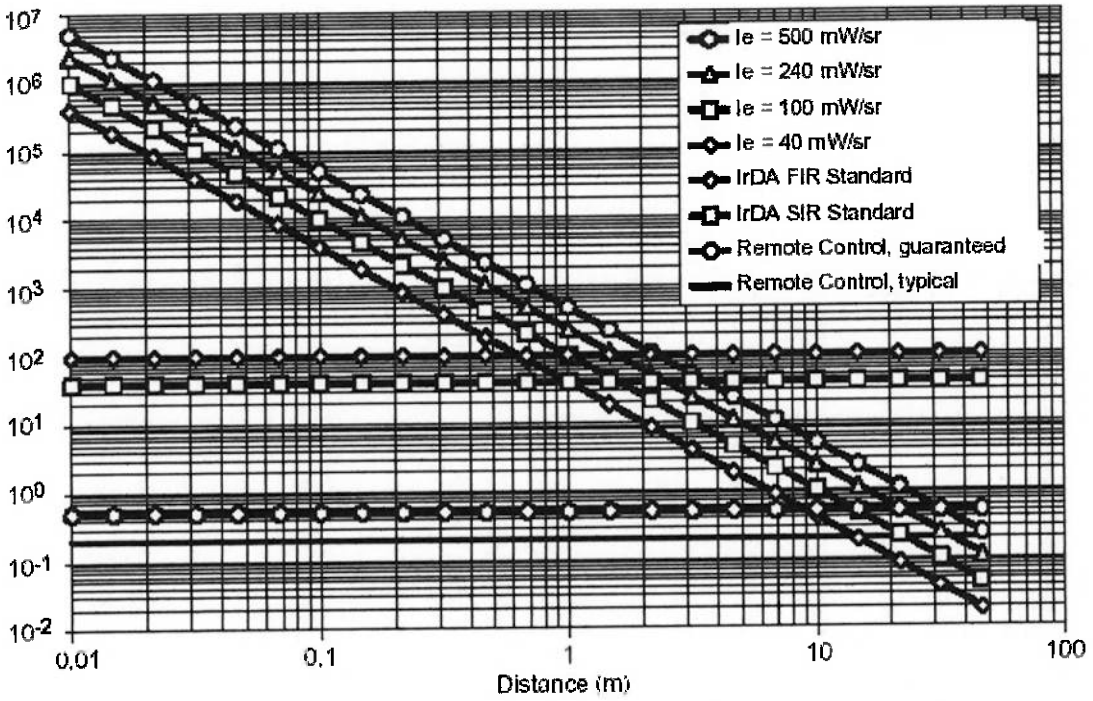


Figura 11 - Distância de transmissão da IrDA

2.2.2 – IrLAP – Link Access Protocol:

O *Link Access Protocol* fornece caminhos para uma conexão na qual o software procura por outras máquinas para conectar (sniffing), reconhece as outras máquinas (discovery), resolve conflitos de endereço, inicia uma conexão e troca de informações, e gerencia a desconexão. Durante a transferência de dados, o IrLAP é responsável em prover uma detecção de erro confiável, retransmissão e controle de fluxo.

A conexão opera essencialmente da seguinte maneira:

Um dispositivo (primário) deseja conectar-se a outro dispositivo (por reconhecimento automático ou por pedido direto do usuário). Uma conexão de dados envolve pelo menos uma estação primária e uma ou mais secundárias, as quais o IrLAP tem a responsabilidade de gerenciar. A estação primária é responsável pela conexão de dados. Toda transmissão em uma conexão de dados ou provêm ou vão

para a estação primária e podem ser ponto-a-ponto ou ponto-a-multipontos. Sempre há apenas uma estação primária, enquanto todas as outras estações são secundárias.

Depois de obedecer às regras de conexão, o primário manda informações de pedido de conexão a 9600 bit/s para os outros dispositivos, como seu endereço e outras capacidades como velocidades de transmissão possíveis etc. O dispositivo que irá responder assume posição de secundário e, depois de obedecer as regras de conexão, retorna informações que contém seu endereço e outras capacidades. Então o primário e o secundário irão mudar suas velocidades de transmissão e outras especificações para um valor comum definido pela troca de informações. O primário então transmite dados para o secundário confirmando a velocidade de transmissão e capacidades. Após isso, os dois dispositivos estão conectados e os dados são transmitidos entre primário e secundário sobre controle completo do secundário.

Qualquer estação capaz pode assumir o papel de primário, o qual é determinado dinamicamente quando a conexão é estabelecida e continua até que a conexão é terminada.

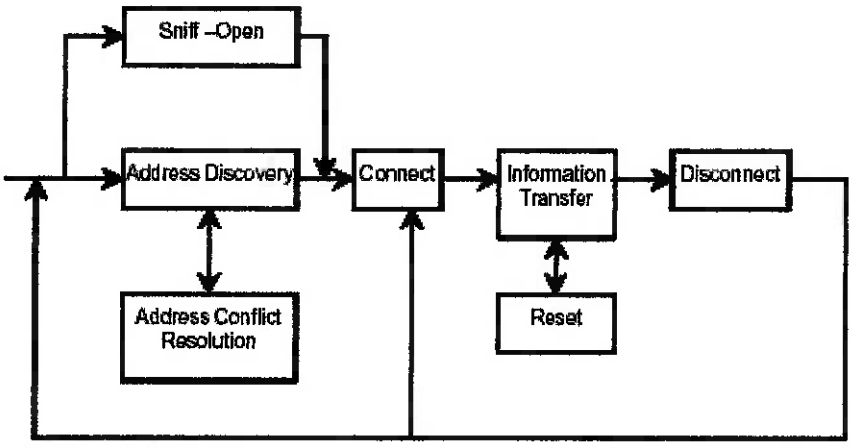


Figura 12 - Procedimentos de operação do IrLAP

2.2.3 – IrLMP – Link Management Protocol:

Uma vez que a conexão é estabelecida, o IrLMP gerencia as funções disponíveis e aplicações entre os dispositivos de comunicação. O IrLMP determina

as capacidades dos dispositivos conectados e gerencia a negociação de parâmetros da conexão e transferência correta de dados.

O IrLMP é responsável por:

✓ Multiplexar serviços na conexão, através do *IrLMP Multiplexer* (LM-MUX);

✓ Localização de componentes suportados pelo *IrLMP Information Access Service* (LM-IAS).

O LM-MUX fornece canais multiplexados no topo de uma conexão IrLAP. Cada uma dessas conexões, chamadas de *Link Service Access Point*, ou conexões LSAP, carrega separadamente uma informação lógica.

O LM-IAS é um cliente direto do LM-MUX . Componentes de aplicativos que desejam estar disponíveis para outros dispositivos se "descrevem" criando um objeto cujos atributos carregam os parâmetros essenciais necessários para sua utilização. Um componente de aplicativo procurando por conexões as procura dentre os objetos do LM-IAS remoto. Assumindo que um objeto adequado existe, os parâmetros da conexão são trocados e ficam acessíveis no topo do LM-IAS.

3 – Microcontroladores da família MCS-51

3.1 – Introdução

O microcontrolador 8051 é o membro original da família MCS-51, e está em produção desde 1981. Todos os membros da família MCS-51 executam o mesmo conjunto de instruções, as quais são otimizadas para aplicações de controle 8-bit.

Todos os microcontroladores MCS-51 possuem um oscilador *on-chip*, que pode ser usado caso desejado como fonte de clock para a CPU. O gerador de clock interno define a seqüência de estados que monta um ciclo de máquina.

Um ciclo de máquina consiste de uma seqüência de seis estados, onde cada estado tem duração de dois períodos do oscilador. Logo, um ciclo de máquina tem duração de 12 períodos do oscilador.

3.2 – Interrupções

O 8051 possui cinco fontes de interrupção: duas externas, duas de timer e a interrupção da porta serial. Cada uma das interrupções pode ser habilitada ou desabilitada individualmente acionando um bit da palavra de controle chama IE (Interrupt Enable). Esse registrador possui ainda um bit para desabilitar todas as interrupções.

(MSB) (LSB)

EA	-	-	ES	ET1	EX1	ET0	EX0

Figura 13 - Registrador IE (Interrupt Enable)

EA – desabilita todas as interrupções;

ES – habilita ou desabilita a interrupção da porta serial/

ET1 – habilita ou desabilita o TIMER1;

EX1 – habilita ou desabilita a Interrupção Externa 1;

ET0 – habilita ou desabilita o TIMER0;

EX0 – habilita ou desabilita a Interrupção Externa 0.

3.3 – Timers/Contadores

Todos os timers dos microcontroladores MCS-51 podem ser configurados para serem usados ou como timers ou como contadores.

Funcionando como timer, o registrador é incrementado a cada ciclo de máquina. Logo, podemos pensar nele como contando ciclos de máquina.

Já funcionando como contador, o registrador é incrementado em resposta a uma transição 1-para-0 em seu pino externo correspondente. Nessa função, a entrada externa é adquirida a cada ciclo de máquina.

Esses timers possuem quatro modos de operação:

- Modo 0:

Nesse modo o timer funciona como um contador de 8-bits. Nesse modo, o registrador é configurado como um registrador de 13-bits. Quando a contagem passa de todos 1s para todos 0s, a bandeira de interrupção do timer TF1 é acionada.

- Modo 1:

O modo 1 é idêntico ao modo 0, porém nesse modo o registrador do timer funciona com todos os 16-bits.

- Modo 2:

O modo 2 configura o registrador do timer como um contador de 8-bits com recontagem automática. Um *overflow* não só define TF1, como também reinicia a contagem.

- Modo 3:

Esse modo estabelece TL0 e TH0 como dois contadores separados. Ele é destinado para aplicações que requerem um timer ou contador de 8-bits extra.

O funcionamento é selecionado nas palavras de controle TMOD e TCON.

(MSB) (LSB)

GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0
TIMER1				TIMER0			

Figura 14 - Registrador TMOD

GATE – habilita controle pela porta "INTx";

$C/\overline{T}$ - Seletor de Timer ou Contador;

M1 M0 – Seletor de modo.

(MSB) (LSB)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Figura 15 - Registrador TCON

TF1 – Bandeira de overflow do Timer 1. Acionada por hardware;

TR1 – Bit de controle para ligar o Timer 1. Acionada por software;

TF0 – Bandeira de overflow do Timer 0. Acionada por hardware;

TR0 – Bit de controle para ligar o Timer 0. Acionada por software;

IE1 – Bandeira da Interrupção Externa 1. Ligada por hardware quando a interrupção é detectada. Desligada quando a interrupção é processada;

IT1 – Bit de controle de tipo de interrupção. Acionada por software para determinar interrupção por nível ou por borda;

IE0 – Bandeira da Interrupção Externa 0. Ligada por hardware quando a interrupção é detectada. Desligada quando a interrupção é processada;

IT0 – Bit de controle de tipo de interrupção. Acionada por software para determinar interrupção por nível ou por borda.

3.4 – Interface Serial

A porta serial é *full duplex*, ou seja, pode transmitir e receber dados simultaneamente. Ela também possui um buffer de recebimento, de modo que ela pode iniciar o recebimento de um segundo byte antes que o primeiro tenha sido lido.

A porta serial pode operar em 4 modos:

- Modo 0:

Os dados seriais entram e saem através do RXD. TXD fornece o sinal de clock. A taxa de transmissão é fixada em 1/12 da freqüência do oscilador.

- Modo 1:

10 bits são transmitidos, pelo TXD, ou recebidos, pelo RXD: um *start bit* (0), os 8 bits do dado (o bit menos significativo primeiro), e um *stop bit* (1). A taxa de transmissão é variada.

- Modo 2:

11 bits são transmitidos e recebidos: um *start bit* (0), os 8 bits do dado (o bit menos significativo primeiro), um nono bit programável, e um *stop bit* (1). A taxa de transmissão é de 1/32 ou 1/64 da freqüência do oscilador.

- Modo 3:

11 bits são transmitidos e recebidos: um *start bit* (0), os 8 bits do dado (o bit menos significativo primeiro), um nono bit programável, e um *stop bit* (1). De fato, esse modo é idêntico ao modo 2, exceto pelo fato de que a taxa de transmissão é variada.

O controle da porta serial é feito através do registrador SCON.

(MSB) (LSB)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Figura 16 - Registrador SCON

SM0 SM1 – seleção de modo da porta serial;

SM2 – habilita a característica de comunicação multiprocessada nos modos 2 e 3;

REN – habilita comunicação serial. Acionada por software;

TB8 – é o 9° bit que será transmitido nos modos 2 e 3;

RB8 – é o 9° bit que será recebido nos modos 2 e 3;

TI – Bandeira de interrupção de transmissão. Ligada por hardware. Deve ser desligada por software;

RI – Bandeira de interrupção de recepção. Ligada por hardware. Deve ser desligada por software;

Para utilizar um baud rate específico deve-se utilizar o timer 1. Para isso, a interrupção desse timer deve ser desabilitado.

III – MATERIAIS E MÉTODOS

O dispositivo é composto por duas portas de comunicação: uma serial RS232 e uma serial infravermelha. A porta serial RS232 é ligada a um conector molex que por sua vez é ligado a um conector DB9, para comunicação com o robô. A porta infravermelha receberá sinais de um dispositivo IrDA, como por exemplo um controle remoto ou um PDA, através de um transceiver.

Para o desenvolvimento deste projeto, foram adquiridas gratuitamente amostras de transceivers IrDA ZHX10Xi/1810 da Zilog, junto a Graftec Eletronics Sales, Inc., distribuidor oficial da Zilog em São Paulo.

Devido à diferença na quantidade de bytes enviados por instrução entre marcas de controle remoto, bem como seus valores, o dispositivo possui um modo de se escolher qual controle remoto, ou PDA, será usado para envio de sinais infravermelho. Esses sinais foram tabelados previamente, quando foram especificadas teclas que correspondem aos comandos desejados para execução pelo robô. Esses comandos são: parar, aumentar ou diminuir a velocidade, e mudar a direção do movimento. Essa escolha é feita através de um botão no dispositivo, e a opção escolhida é sinalizada através de LEDs. Um segundo botão é usado para escolha entre comunicação com um computador ou com o robô.

1 – TESTE DE BANCADA COM O TRANSCEIVER

O circuito do transceiver ZHX10Xi/1810 está esquematizado na Figura 17.

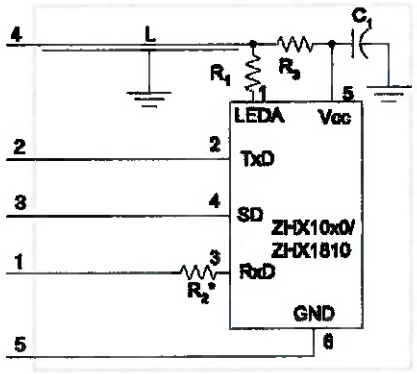


Figura 17 - Circuito do Transceiver ZHX10Xi/1810

Nesse circuito, a resistência R_1 original era de 2,7 ohms, responsável pelo controle de corrente no emissor de infravermelho com uma alimentação de 3 volts. Como no dispositivo a alimentação é de 5 volts, foi necessário substituí-la por uma resistência de 8,2 ohms.

No teste do transceiver analisou-se o sinal de saída do transceiver IrDA ao receber sinais de infravermelho provenientes de um PDA (Palm m105) e de controles remotos de diferentes marcas.

O sinal de saída está nas figuras abaixo.

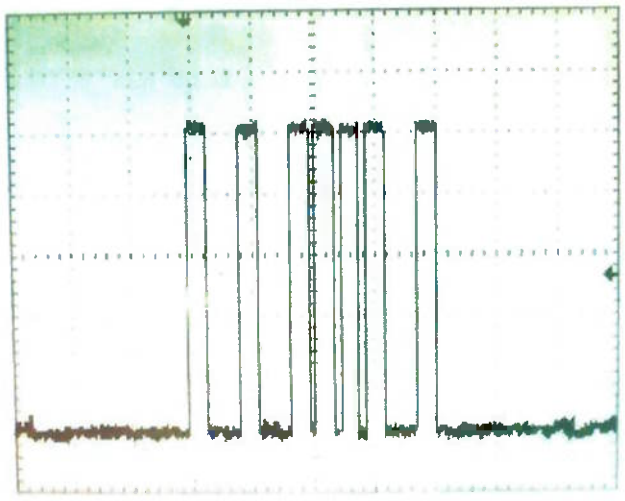


Figura 18 - Sinal de saída do transceiver ao receber sinais de um Palm

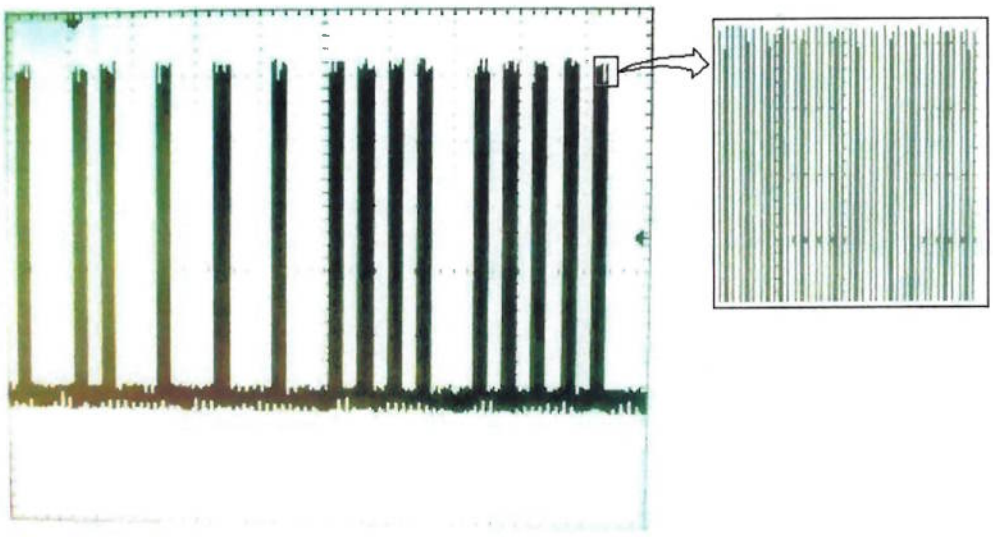


Figura 19 - Sinal de saída do transceiver ao receber sinais de um controle remoto

Como podemos notar, o sinal proveniente do Palm é exatamente como especificado pela IrDA. Já o sinal proveniente de alguns controles remotos vem modulado. Porém, como veremos mais adiante, esse fato não influencia no projeto, pois o sinal será analisado pela sua primeira borda de descida.

2 – ESTUDO DAS FUNÇÕES QUE SERÃO IMPLEMENTADAS NO MICROCONTROLADOR

A fim de se determinar qual microcontrolador será utilizado neste projeto, foram feitos algoritmos das funções que serão implementadas, de modo a se determinar o número de interrupções, timers e tamanho aproximado do programa que serão necessários no microcontrolador. Para isso, optou-se por utilizar um microcontrolador compatível com um 8051, com uma UART implementada, para a comunicação serial com o robô.

Uma vez que o menor pulso possível de se receber dura apenas 1,41 μs, não é possível analisá-lo por uma porta de I/O. Isso porque um ciclo de máquina (tempo mínimo necessário para executar uma instrução) de um microcontrolador que opera a 24 MHz é de 0,5 μs (12 ciclos do oscilador), fazendo com que não seja sempre possível a detecção desse pulso pelo programa. Então, para garantir que todos os pulsos serão detectados, a saída do transceiver será ligada ao microcontrolador acionando uma interrupção externa, que por sua vez será configurada para ser sensível a borda de descida do sinal.

Para garantir que não sejam detectados possíveis ruídos entre um pulso e outro, será usado um timer que funcionará ciclicamente toda vez que um novo byte for recebido, a uma freqüência um pouco maior que a máxima freqüência do sinal. Desse modo o programa consegue determinar quantos bits estão chegando, eliminando ruídos e o problema referente à modulação do sinal dos controles remotos.

Os bytes que chegam são colocados em um buffer, que possui dois índices, como mostra a Figura 20: um com a posição do último byte recebido, e outro com a posição do último byte lido. Para saber quantos bytes chegaram basta então analisar a diferença entre esses dois índices.

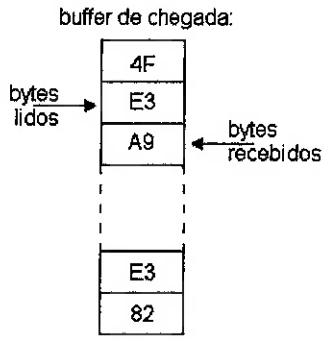


Figura 20 - Esquema do buffer de chegada

Assim, um possível algoritmo do programa principal é descrito a seguir:

- Inicializa as variáveis do programa, as interrupções e os timers;
- Mostra a opção padrão do dispositivo IrDA através de LEDs;
- Entra em LOOP verificando existência de novos dados no buffer de chegada, ou mudança no dispositivo IrDA através de um botão ligado a um bit de I/O;
- Na chegada de novos dados, pára o LOOP;
- Compara os novos dados com valores tabelados de acordo com o dispositivo selecionado;
- Em caso de dados válidos, envia o respectivo comando pela porta serial para o robô (8 bits de direção e 8 bits de velocidade a 9600 bits/s);
- Volta para o LOOP.

A interrupção acionada externamente pelo transceiver (IE0) é a responsável por montar o byte recebido. Um possível algoritmo é:

- verifica número de bits já recebidos (cont):
    - se cont = 0:
        - habilita TIMER0;
        - byte_recebido = FF;
        - desabilita IE0.
    - se cont > 0:
        - grava um 0 em byte_recebido[cont];
        - desabilita IE0.

Um timer (TIMER0), é responsável por guardar os bytes recebidos no buffer:

- habilita IE0;

- cont = cont + 1;
- se cont = 9:
    - grava byte_recebido no buffer de chegada;
    - cont = 0;
    - incrementa índice de bytes recebidos;
    - desabilita TIMER0.

3 – ESCOLHA DO MICROCONTROLADOR

De modo a ser possível implementar os algoritmos descritos no item anterior, foram levadas em consideração as seguintes características na escolha do microcontrolador:

- Compatível com 8051;
- Interface serial (UART) implementada, e um timer para gerar o baud rate;
- para recepção de dados do transceiver: uma interrupção acionada externamente, e uma acionada por timer;
- tamanho da memória flash;
- número de linhas de I/O.

Assim, o microcontrolador escolhido foi o AT89C4051, da Atmel. Esse microcontrolador, já disponível no laboratório, possui as seguintes características:

- compatível com MCS 51;
- memória flash de 4 kbytes;
- memória RAM de 128 bytes;
- trabalha até a 24 MHz;
- 15 linhas de I/O programáveis;
- Canal serial UART programável;
- 6 fontes de interrupção;
- 2 timers/counters de 16 bits.

4 – PROJETO DO HARDWARE

O dispositivo proposto neste trabalho deve fornecer uma interface entre dispositivos IrDA e RS232. Assim sendo, deve conter um microcontrolador, um transceiver infravermelho capaz de receber sinais compatíveis com IrDA, e botões e LEDs para permitir uma interface com o usuário. O diagrama de blocos do sistema está na Figura 21.

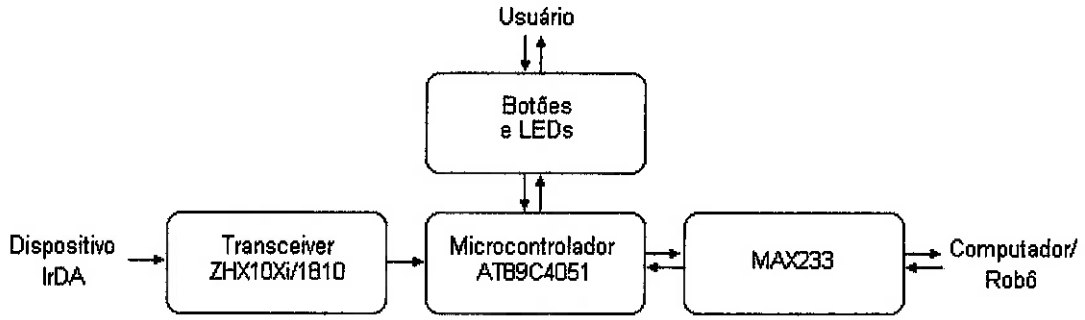


Figura 21 - Diagrama de Blocos do Sistema

A partir desse diagrama de blocos, foi feito um esquema do hardware. Então, o hardware foi montado em uma protoboard, onde foram realizados alguns testes, e, com o auxílio de um software de CAD eletrônico, foi feito o projeto da placa, que foi fabricada em seguida.

4.1 – Desenho esquemático do hardware:

A partir do Diagrama de Blocos da Figura 21, foi feito um desenho esquemático do hardware, mostrado na Figura 22, no software EAGLE Layout Editor, da CadSoft.

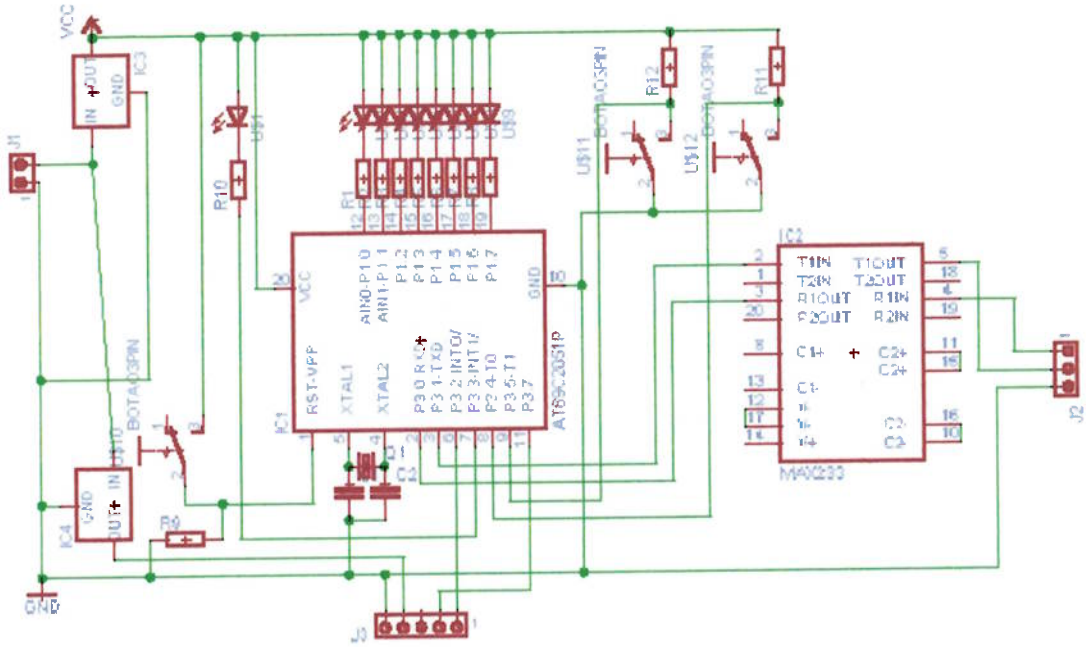


Figura 22 - Desenho esquemático do Hardware

4.2 – Montagem em protoboard e testes:

Com o esquema do hardware pronto, o mesmo foi montado em uma protoboard, para a realização de alguns testes, antes da fabricação da placa.

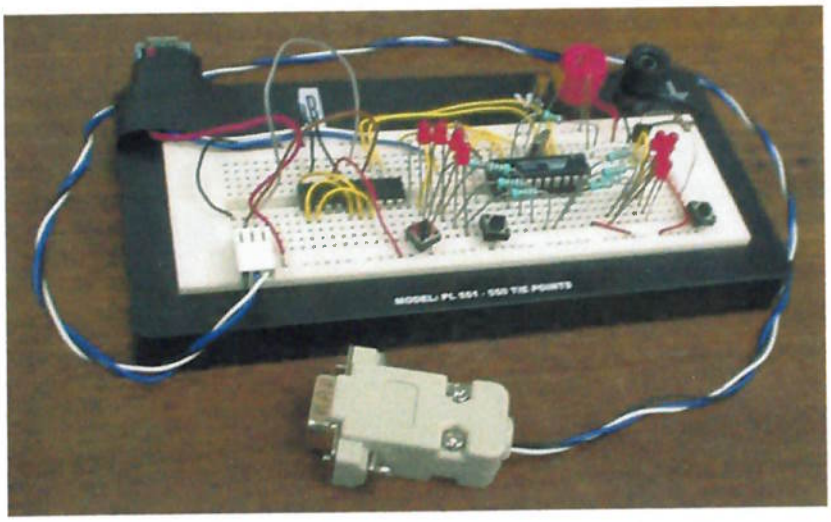


Figura 23 - Montagem do hardware em protoboard

Os testes realizados incluem recepção e envio pela porta serial RS232 do dispositivo, e recepção pela porta IR.

4.3 – Projeto da placa:

O projeto da placa foi feito a partir do esquema do hardware, também com o auxílio do software EAGLE Layout Editor, da CadSoft.

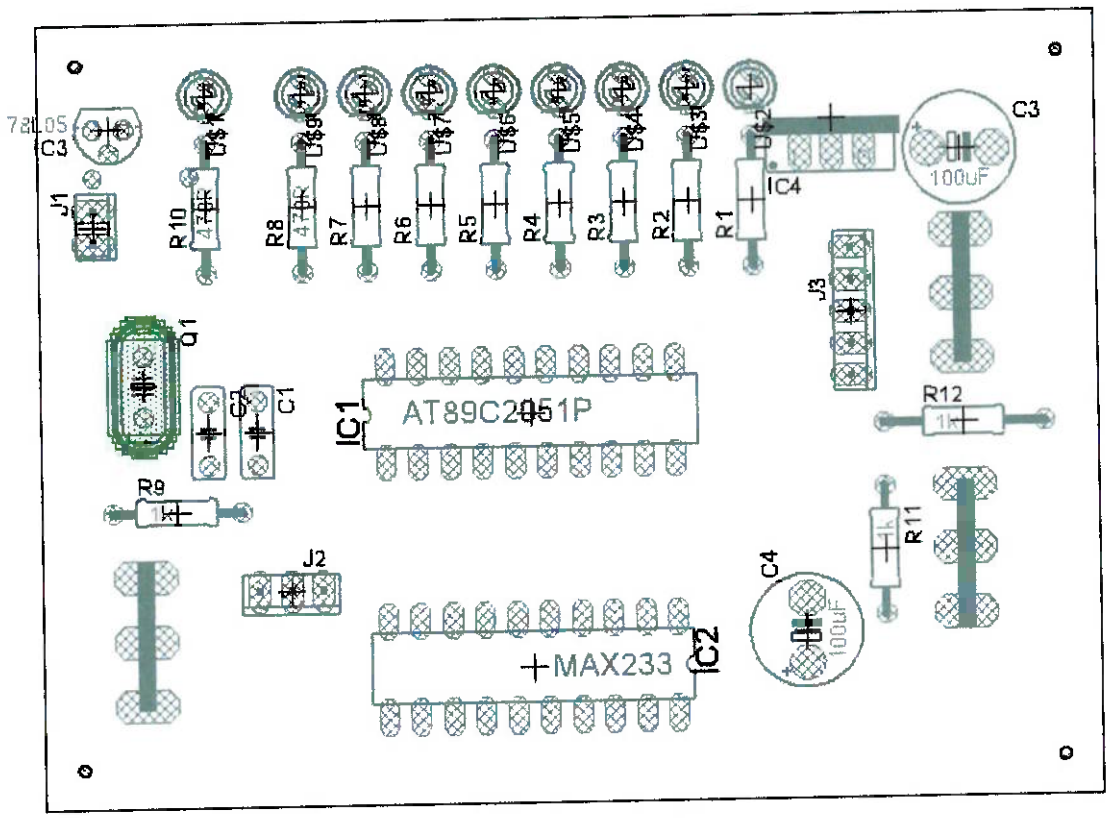


Figura 24 - Projeto da placa

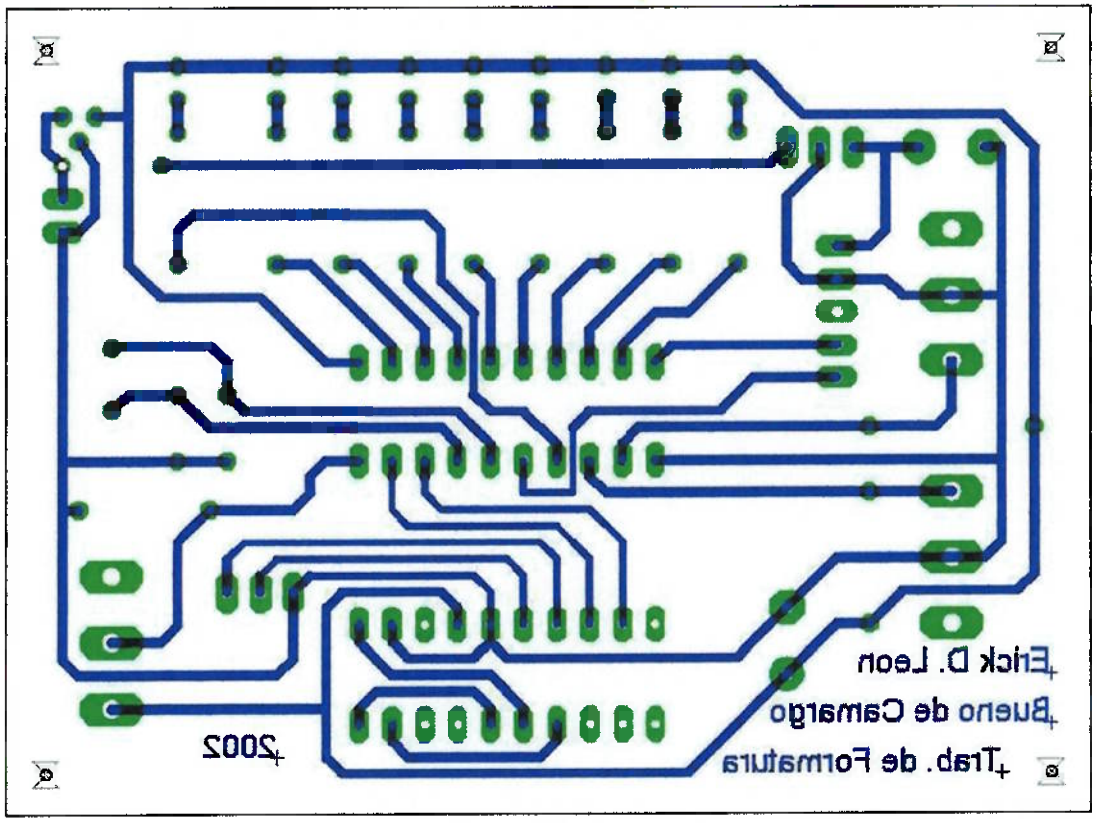


Figura 25 - Projeto do circuito

Paralelamente ao projeto e fabricação da placa, foram mapeados botões de diferentes controles remotos com a montagem em protoboard.

4.4 – Fabricação da placa:

Para a fabricação da placa o circuito foi impresso em uma transparência, e então colocado acima de uma placa de silício com um material photoresist aplicado. O conjunto foi então exposto à luz ultra-violeta durante certo tempo, e banhada com material corrosivo em seguida. Com isso obteve-se o circuito da Figura 26.

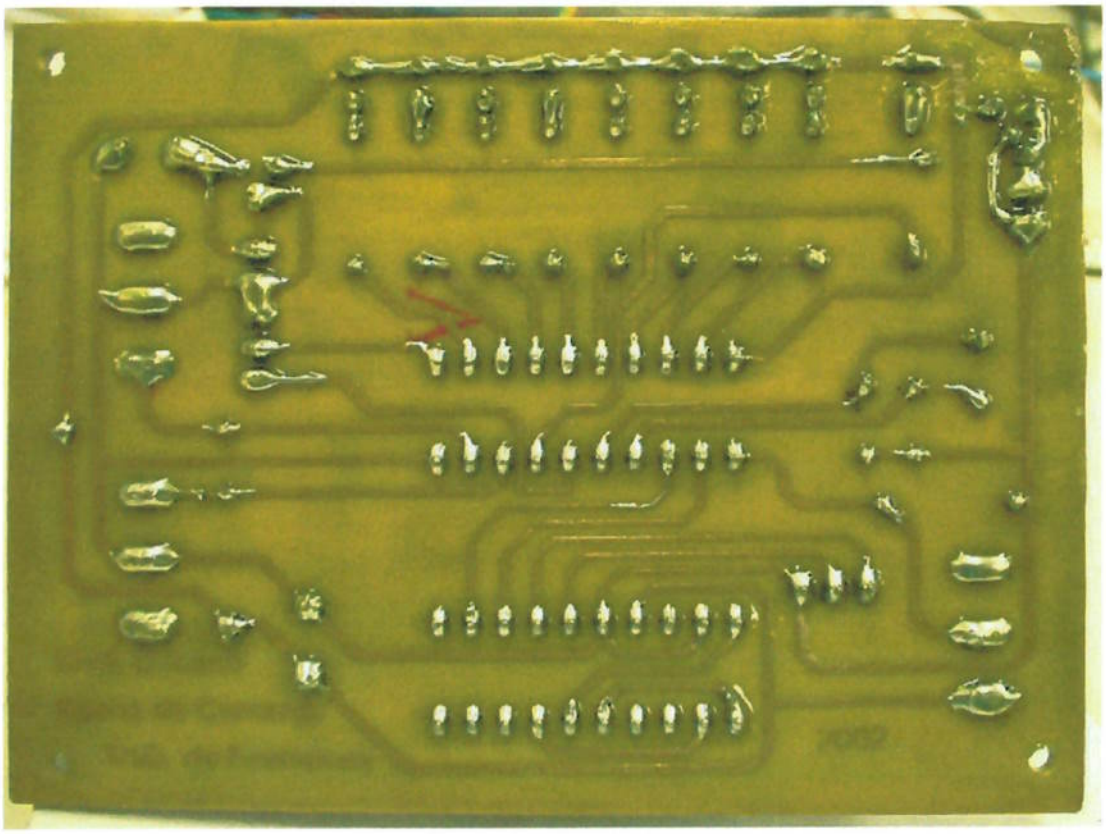


Figura 26 - Foto do circuito

Nesse circuito foram soldados os componentes, originando a placa da Figura 27. Após a soldagem foi aplicado um verniz próprio para circuitos, para evitar a oxidação das trilhas de cobre.

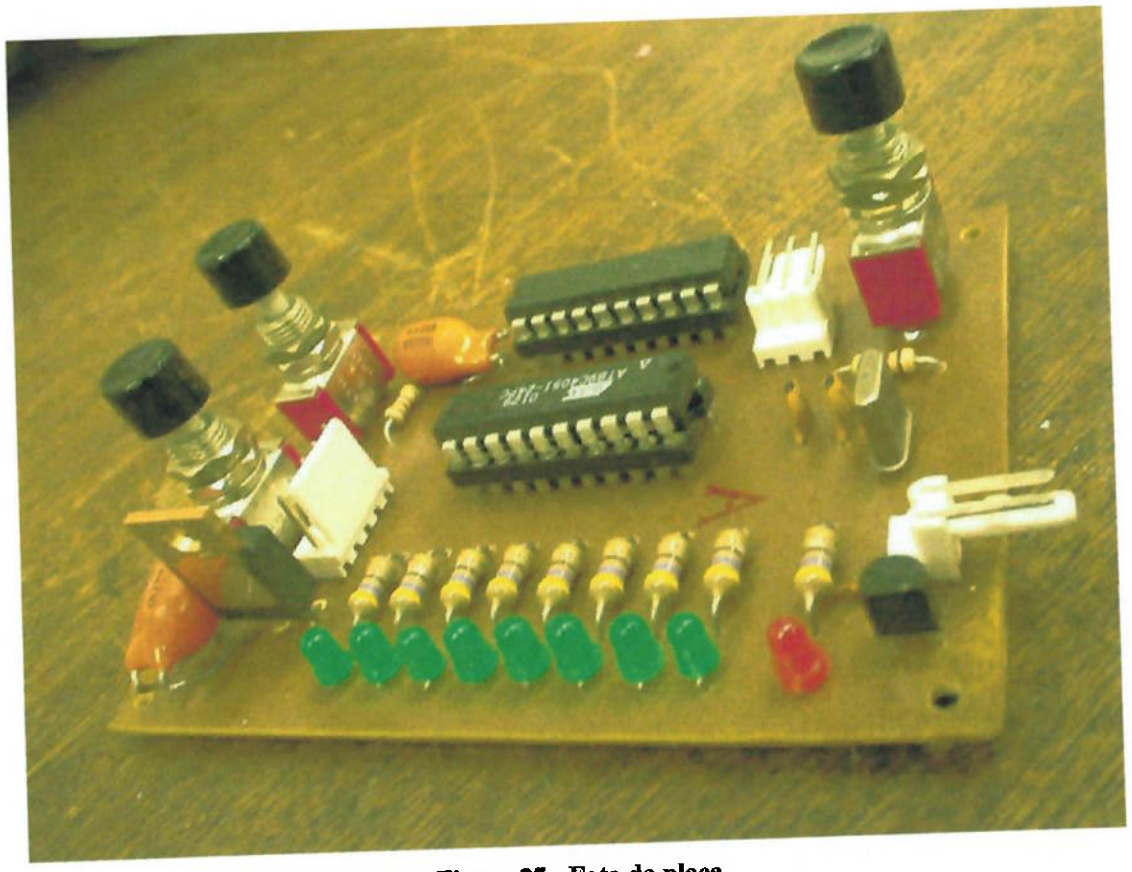


Figura 27 - Foto da placa

4.5 – Montagem no robô:

O dispositivo foi montado no robô Base Móvel para controlá-lo ao receber comandos de um dispositivo IrDA (Figura 28). Para esse projeto foram usados um controle remoto e um Palm m105.

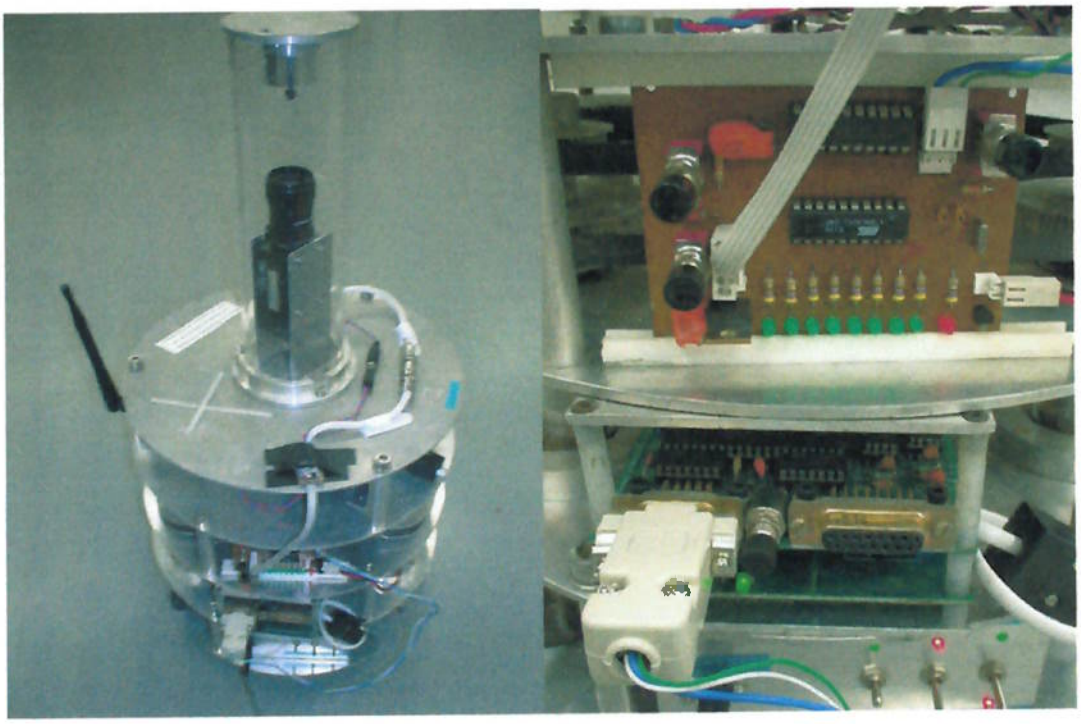


Figura 28 - Fotos do dispositivo no robô

5 – PROGRAMAÇÃO DO SOFTWARE PARA PALMOS

5.1 – Introdução:

Programar aplicações para PDAs é um pouco diferente de programar aplicações para computadores de mesa, uma vez que a plataforma do PalmOS é diferente da de um computador de mesa. Além disso, os usuários interagem diferentemente com um PDA do que com um computador de mesa.

Alguns dos aspectos importantes a serem observados são apresentados a seguir.

Tamanho da tela: a tela de um Palm possui apenas 160x160 pixels, logo a interface deve ser projetada cuidadosamente com diferentes prioridades e objetivos que as usadas em uma tela maior.

Tempo de resposta: em um computador de mesa o usuário não se importa em esperar alguns segundos para obter uma resposta do sistema. Entretanto, aplicações para PDA são usadas freqüentemente, várias vezes ao dia, em espaços de tempo muito curtos. Por esse motivo a velocidade do aplicativo se torna uma parte crítica do projeto de aplicações para PDAs.

Entrada de dados: uma vez que a entrada de dados em um PDA se dá através de uma caneta, não é conveniente requerer que o usuário entre com uma quantidade grande de dados.

Memória: devido à limitação de energia e memória em um PDA, a otimização da aplicação se torna outro fator crítico do projeto.

Sistema de arquivos: por causa da capacidade limitada de armazenamento, o PalmOS não usa um sistema de arquivos convencional. Os dados são guardados em bancos de dados, e não em arquivos. Logo, é recomendável editar bancos de dados alocados em memória ao invés de criá-los em RAM e então armazená-los, de modo a economizar espaço de memória.

Existem várias ferramentas de programação para construir e testar aplicativos para PalmOS. A ferramenta mais comumente utilizada é o *CodeWarrior Interactive Development Environment (IDE)*, que utiliza C e C++ como linguagem de

programação. Por ser essa a ferramenta de programação oficial da Palm, possui uma ampla documentação na internet. Por esses motivos essa foi a ferramenta utilizada para o projeto do aplicativo.

5.2 – Comunicação Infravermelho:

O PalmOS possui três níveis de suporte para comunicação infravermelho:

- O *Exchange Manager* fornece uma interface de alto-nível que lida com todos os detalhes da comunicação transparentemente;
- O *Serial Manager* fornece um drive virtual que implementa o protocolo *IrComm*;
- A *IR Library* fornece capacidades de comunicação de baixo-nível, e direta à interface infravermelha. É direcionada para aplicações que precisem de um acesso direto à porta infravermelha e por esse motivo é a que foi utilizada nesse projeto. O suporte infravermelho fornecido pelo PalmOS está de acordo com as especificações IrDA.

5.3 – Interface com o usuário:

O programa é composto de duas partes: a primeira destina-se ao estudo da comunicação infravermelho, onde se pode receber e enviar caracteres em ASCII ou hexadecimal pela porta infravermelha, em diversos baud rates (2400, 4800, 9600, 19200, 38400, 57600 e 115200 bps), e a segunda destina-se ao controle à distância do robô, onde se pode escolher a velocidade do robô, bem como a direção de seu movimento.

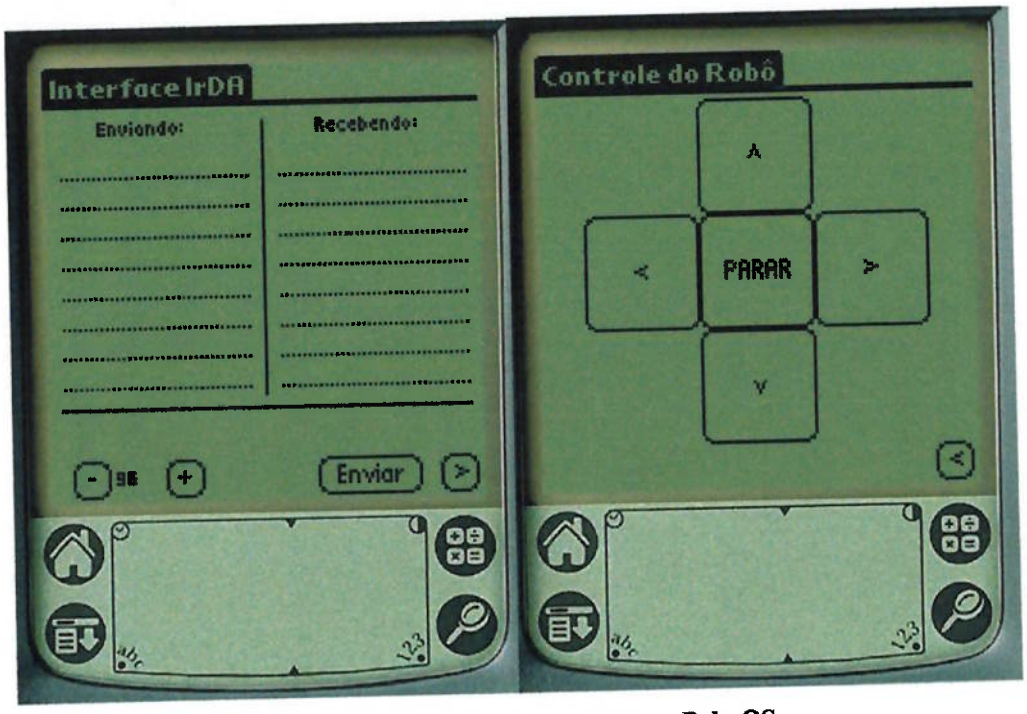


Figura 29 - Telas do programa para PalmOS

6 – PROGRAMAÇÃO DO MICROCONTROLADOR

O controle do 8051 é feito, na maior parte, por Registros de Função Especial (*SFRs – Special Function Registers*).

0F8H								0FFH
0F0H	B 00000000							0F7H
0E8H								0EFH
0E0H	ACC 00000000							0E7H
0D8H								0DFH
0D0H	PSW 00000000							0D7H
0C8H								0CFH
0C0H								0C7H
0B8H	IP XXX00000							0BFH
0B0H	P3 11111111							0B7H
0A8H	IE 0XX00000							0AFH
0A0H								0A7H
98H	SCON 00000000	SBUF XXXXXXXX						9FH
90H	P1 11111111							97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000		8FH
80H		SP 00000111	DPL 00000000	DPH 00000000				PCON 0XXX0000	87H

Figura 30 - Mapa de endereços dos *SFRs* do AT89C4051

P1 e **P3**: posições na RAM que contém os dados das portas de I/O do microcontrolador, caso as mesmas sejam utilizadas para esse fim. Se for feita uma escrita em um desses registros, o conteúdo nos pinos correspondentes à porta será automaticamente alterado. Já uma leitura dos mesmos verifica o estado dos pinos.

SP: é o apontador de pilha (*Stack Pointer*), que aponta para o alto da pilha.

DPL e **DPH**: em conjunto formam o DPTR, utilizado para endereçamento indireto de 16 bits.

PCON: o registro PCON (*Power Control*) permite adaptar o microcontrolador para redução de consumo com segurança.

TCON e **TMOD**: registros dos Temporizadores/Contadores. Permitem a programação dos mesmos.

TL0, **TH0**, **TL1** e **TH1**: são os registros dos dados dos dois Temporizadores/Contadores (T0 e T1).

SCON e **SBUF**: registros para uso da porta de comunicação serial.

IE e **IP**: registro para programação (habilitação/desabilitação, prioridade etc.) das interrupções.

PSW: o PSW (*Program Status Word* – palavra de status do programa) é o registro dos *Flags* do 8051.

ACC: é o acumulador.

B: registro auxiliar B.

O programa é composto de uma rotina principal, rotinas de inicialização das variáveis e das interrupções, rotinas de envio de caracteres em ASCII e em hexadecimal através da porta serial RS232, e as rotinas das interrupções, externa, serial, e timer 0. Foram utilizadas as linguagens C e assembler para a programação do microcontrolador.

6.1 – A comunicação serial RS232:

A porta serial é configurada para funcionar como uma UART de 8 bits no modo 1, com baud rate (taxa de transmissão) variado, através do Timer 1.

A recepção pela porta serial RS232 é feita por interrupção, onde os bytes recebidos são guardados em um buffer, e a transmissão é feita por pooling (wait-for-flag). Apesar de o robô não enviar dados pela porta serial, a recepção é utilizada no dispositivo para a comunicação com um computador. Assim, foi possível tabelar os caracteres enviados por um controle remoto quando determinados botões são enviados, bem como analisar a melhor freqüência do Timer 0 na recepção dos caracteres por infravermelho, sem a necessidade de reprogramar o microcontrolador.

O Timer 1 é configurado para funcionar a uma freqüência o mais próximo possível de 9600 Hz, de modo a obter uma comunicação de 9600 bps.

6.2 – A comunicação serial Infravermelha:

A comunicação infravermelha é feita através da interrupção externa 0 e do timer 0. O timer 0 é responsável pela velocidade de recepção e contagens dos bits, podendo-se escolher entre qualquer divisor de 19200bps, ou seja, 19200bps, 9600bps, 4800bps, 2400bps etc.

Para identificação do byte recebido tanto o timer 0 como a interrupção externa 0 funcionam como nos algoritmos descritos na seleção do microcontrolador.

6.3 – A rotina principal:

A rotina principal inicializa as variáveis e as interrupções utilizadas no programa, e entra em um LOOP infinito.

Dentro desse LOOP, o programa fica verificando continuamente a chegada de dados pela porta serial RS232, por infravermelho ou se algum botão está sendo pressionado. Um botão é utilizado para mudar o dispositivo IrDA que será usado com o dispositivo, o qual é mostrado através de LEDs, e outro botão é utilizado para a selecionar entre comunicação com um computador ou com o robô. Na comunicação com o robô, caso algum dado seja recebido, ele é comparado com os dados previamente tabelados na memória, e em caso de correspondência com algum comando, esse comando é enviado pela porta serial RS232 ao robô. Na comunicação com o computador, pode-se ver os dados no buffer de recebimento infravermelho, verificar caracteres IrDA enquanto são recebidos, ou ainda mudar o nível lógico de um pino da porta 1 de I/O do microcontrolador.

IV – CONCLUSÕES

Foram adquiridas gratuitamente amostras do transceiver IrDA junto a Graftec Eletronics Sales, Inc.

Foi projetado e construído no Laboratório de Percepção Avançada da Escola Politécnica um dispositivo capaz de receber sinais infravermelhos compatíveis com IrDA, compreendê-los, e receber e enviar sinais por uma porta RS232.

O dispositivo foi testado com sucesso na recepção de dados em infravermelho e na comunicação com um computador, e no controle do robô Base Móvel do Laboratório de Percepção Avançada (LPA).

O dispositivo pode ser usado para o controle de qualquer equipamento capaz de receber comandos por RS232, através de um dispositivo IrDA.

V – REFERÊNCIAS

[1] INFRARED DATA ASSOCIATION (IrDA). Site oficial da IrDA. Disponível em <http://www.irda.org/>. Acesso em janeiro de 2002;

[2] INTERFACING THE PC. Site sobre interfaces de computadores. Disponível em <http://www.beyondlogic.org/>. Acesso em janeiro de 2002;

[3] LAMMERT BIES: RS232 CABLE INFORMATION. Site com informações sobre RS232 e UART. Disponível em <http://www.lammertbies.nl/comm/cable/RS-232.html>. Acesso em janeiro de 2002;

[4] WILLIAMS, S.; MILLAR, I. **The IrDA Plataform**. HP Laboratóries, Bristol;

[5] MEGOWAN, P. et al. **IrDA Infrared Communications: An Overview**. Counterpoint Systems Foundry, Inc.;

[6] KNUTSON, C. D. **Infrared Data Communications with IrDA**. Counterpoint Systems Foundry, Inc.;

[7] Vishay Telefunken. **About IrDA: IrDA-Compatible Data Transmission, IrDA-Standart-Physical Layer**;

[8] Vishay Telefunken. **About IrDA: Software Protocol**;

[9] **Embedeed Controller Handbook**. Volume I, 8-bit, Intel, 1988;

[10] **PalmOS Programmer´s Companion**. Volume I, Palm, Inc., 2001;

[11] **PalmOS Programmer´s Companion**. Volume II – Communications, Palm, Inc., 2001.

ANEXO I –

Datasheet do Microcontrolador AT89C4051

Features

- Compatible with MCS-51™ Products
- 4K Bytes of Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- 3.0V to 6V Operating Range
- Fully Static Operation: 0 Hz to 24 MHz
- Two-level Program Memory Lock
- 128 x 8-bit Internal RAM
- 15 Programmable I/O Lines
- Two 16-bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial UART Channel
- Direct LED Drive Outputs
- On-chip Analog Comparator
- Low-power Idle and Power-down Modes
- Brown-out Detection

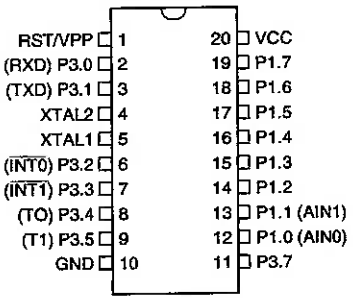
Description

The AT89C4051 is a low-voltage, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard MCS-51 instruction set. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C4051 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

The AT89C4051 provides the following standard features: 4K bytes of Flash, 128 bytes of RAM, 15 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, a precision analog comparator, on-chip oscillator and clock circuitry. In addition, the AT89C4051 is designed with static logic for operation down to zero frequency and supports two software-selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. The power-down mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

Pin Configuration

PDIP/SOIC

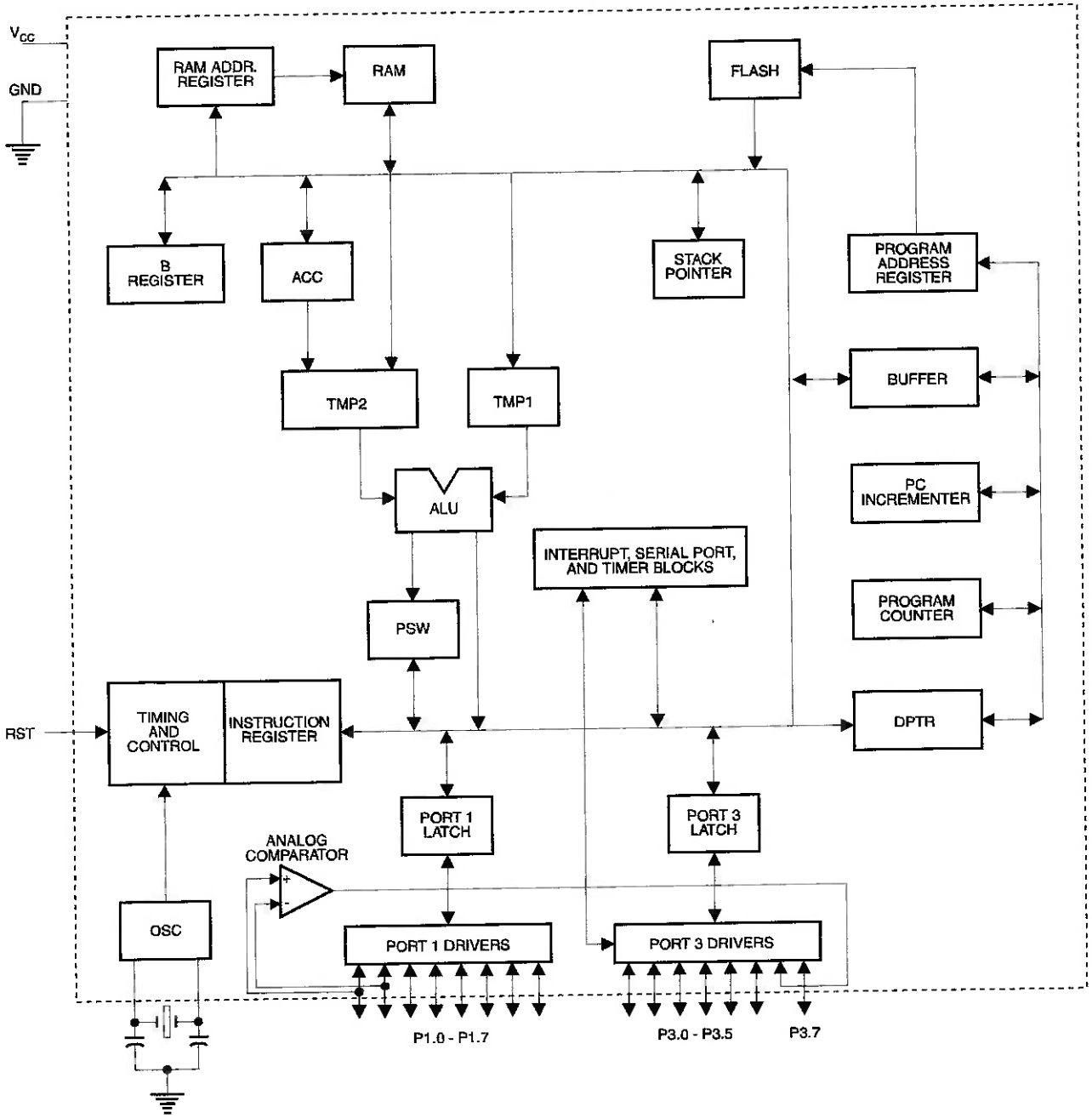


8-bit Microcontroller with 4K Bytes Flash

AT89C4051

Rev. 1001C–02/00

Block Diagram



Pin Description

VCC

Supply voltage.

GND

Ground.

Port 1

Port 1 is an 8-bit bidirectional I/O port. Port pins P1.2 to P1.7 provide internal pullups. P1.0 and P1.1 require external pullups. P1.0 and P1.1 also serve as the positive input (AIN0) and the negative input (AIN1), respectively, of the on-chip precision analog comparator. The Port 1 output buffers can sink 20 mA and can drive LED displays directly. When 1s are written to Port 1 pins, they can be used as inputs. When pins P1.2 to P1.7 are used as inputs and are externally pulled low, they will source current (I_{IL}) because of the internal pullups.

Port 1 also receives code data during Flash programming and verification.

Port 3

Port 3 pins P3.0 to P3.5, P3.7 are seven bidirectional I/O pins with internal pullups. P3.6 is hard-wired as an input to the output of the on-chip comparator and is not accessible as a general purpose I/O pin. The Port 3 output buffers can sink 20 mA. When 1s are written to Port 3 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C4051 as listed below:

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)

Port 3 also receives some control signals for Flash programming and verification.

RST

Reset input. All I/O pins are reset to 1s as soon as RST goes high. Holding the RST pin high for two machine cycles while the oscillator is running resets the device.

Each machine cycle takes 12 oscillator or clock cycles.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

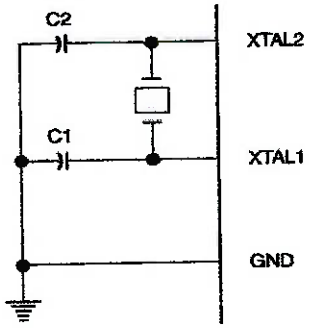
XTAL2

Output from the inverting oscillator amplifier.

Oscillator Characteristics

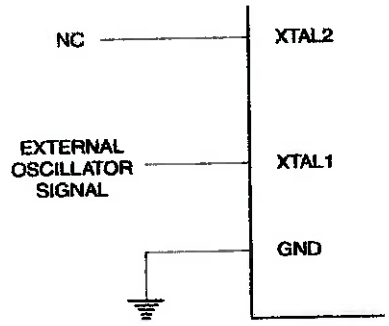
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Figure 1. Oscillator Connections



Note: C1, C2= 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Figure 2. External Clock Drive Configuration



Special Function Registers

A map of the on-chip memory area called the Special Function Register (SFR) space is shown in the table below.

Note that not all of the addresses are occupied, and unoccupied addresses may not be implemented on the chip. Read accesses to these addresses will in general return random data, and write accesses will have an indeterminate effect.

User software should not write 1s to these unlisted locations, since they may be used in future products to invoke new features. In that case, the reset or inactive values of the new bits will always be 0.

Table 1. AT89C4051 SFR Map and Reset Values

0F8H								0FFH
0F0H	B 00000000							0F7H
0E8H								0EFH
0E0H	ACC 00000000							0E7H
0D8H								0DFH
0D0H	PSW 00000000							0D7H
0C8H								0CFH
0C0H								0C7H
0B8H	IP XXX00000							0BFH
0B0H	P3 11111111							0B7H
0A8H	IE 0XX00000							0AFH
0A0H								0A7H
98H	SCON 00000000	SBUF XXXXXXXX						9FH
90H	P1 11111111							97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000		8FH
80H		SP 00000111	DPL 00000000	DPH 00000000			PCON 0XXX0000	87H

Restrictions on Certain Instructions

The AT89C4051 is an economical and cost-effective member of Atmel's growing family of microcontrollers. It contains 4K bytes of flash program memory. It is fully compatible with the MCS-51 architecture, and can be programmed using the MCS-51 instruction set. However, there are a few considerations one must keep in mind when utilizing certain instructions to program this device.

All the instructions related to jumping or branching should be restricted such that the destination address falls within the physical program memory space of the device, which is 4K for the AT89C4051. This should be the responsibility of the software programmer. For example, LJMP 0FE0H would be a valid instruction for the AT89C4051 (with 4K of memory), whereas LJMP 1000H would not.

1. Branching instructions:

LCALL, LJMP, ACALL, AJMP, SJMP, JMP @A+DPTR. These unconditional branching instructions will execute correctly as long as the programmer keeps in mind that the destination branching address must fall within the physical boundaries of the program memory size (locations 00H to FFFH for the 89C4051). Violating the physical space limits may cause unknown program behavior.

CJNE [...], DJNZ [...], JB, JNB, JC, JNC, JBC, JZ, JNZ With these conditional branching instructions the same rule above applies. Again, violating the memory boundaries may cause erratic execution.

For applications involving interrupts the normal interrupt service routine address locations of the 80C51 family architecture have been preserved.

2. MOVX-related instructions, Data Memory:

The AT89C4051 contains 128 bytes of internal data memory. Thus, in the AT89C4051 the stack depth is limited to 128 bytes, the amount of available RAM. External DATA memory access is not supported in this device, nor is external PROGRAM memory execution. Therefore, no MOVX [...] instructions should be included in the program.

A typical 80C51 assembler will still assemble instructions, even if they are written in violation of the restrictions mentioned above. It is the responsibility of the controller user to know the physical features and limitations of the device being used and adjust the instructions used correspondingly.

Program Memory Lock Bits

On the chip are two lock bits which can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the following table:

Lock Bit Protection Modes[1]

Program Lock Bits			
	LB1	LB2	Protection Type
1	U	U	No program lock features.
2	P	U	Further programming of the Flash is disabled.
3	P	P	Same as mode 2, also verify is disabled.

Note: 1. The Lock Bits can only be erased with the Chip Erase operation.

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

P1.0 and P1.1 should be set to "0" if no external pullups are used, or set to "1" if external pullups are used.

It should be noted that when idle is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

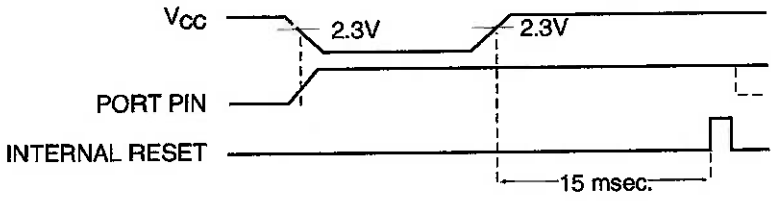
Power-down Mode

In the power-down mode the oscillator is stopped, and the instruction that invokes power-down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the power-down mode is terminated. The only exit from power-down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

P1.0 and P1.1 should be set to "0" if no external pullups are used, or set to "1" if external pullups are used.

Brown-out Detection

When V_{CC} drops below the detection threshold, all port pins except P1.0 and P1.1) are weakly pulled high. When V_{CC} goes back up again, an internal Reset is automatically generated after a delay of typically 15 msec. The nominal brown-out detection threshold is 2.3V ± 10%.



Programming The Flash

The AT89C4051 is shipped with the 4K bytes of on-chip PEROM code memory array in the erased state (i.e., contents = FFH) and ready to be programmed. The code memory array is programmed one byte at a time. *Once the array is programmed, to re-program any non-blank byte, the entire memory array needs to be erased electrically.*

Internal Address Counter: The AT89C4051 contains an internal PEROM address counter which is always reset to 000H on the rising edge of RST and is advanced by applying a positive going pulse to pin XTAL1.

Programming Algorithm: To program the AT89C4051, the following sequence is recommended.

1. Power-up sequence:
   Apply power between VCC and GND pins
   Set RST and XTAL1 to GND
2. Set pin RST to "H"
   Set pin P3.2 to "H"
3. Apply the appropriate combination of "H" or "L" logic levels to pins P3.3, P3.4, P3.5, P3.7 to select one of the programming operations shown in the PEROM Programming Modes table.

To Program and Verify the Array:

4. Apply data for Code byte at location 000H to P1.0 to P1.7.
5. Raise RST to 12V to enable programming.
6. Pulse P3.2 once to program a byte in the PEROM array or the lock bits. The byte-write cycle is self-timed and typically takes 1.2 ms.
7. To verify the programmed data, lower RST from 12V to logic "H" level and set pins P3.3 to P3.7 to the appropriate levels. Output data can be read at the port P1 pins.
8. To program a byte at the next address location, pulse XTAL1 pin once to advance the internal address counter. Apply new data to the port P1 pins.
9. Repeat steps 5 through 8, changing data and advancing the address counter for the entire 4K bytes array or until the end of the object file is reached.
10. Power-off sequence:
    set XTAL1 to "L"
    set RST to "L"
    Turn V_{CC} power off

$\overline{\text{Data}}$ Polling: The AT89C4051 features $\overline{\text{Data}}$ Polling to indicate the end of a write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written data on P1.7. Once the write cycle has been completed, true data is valid on all outputs, and the next cycle may begin. $\overline{\text{Data}}$ Polling may begin any time after a write cycle has been initiated.

Ready/$\overline{\text{Busy}}$: The Progress of byte programming can also be monitored by the RDY/$\overline{\text{BSY}}$ output signal. Pin P3.1 is pulled low after P3.2 goes High during programming to indicate BUSY. P3.1 is pulled High again when programming is done to indicate READY.

Program Verify: If lock bits LB1 and LB2 have not been programmed code data can be read back via the data lines for verification:

1. Reset the internal address counter to 000H by bringing RST from "L" to "H".
2. Apply the appropriate control signals for Read Code data and read the output data at the port P1 pins.
3. Pulse pin XTAL1 once to advance the internal address counter.
4. Read the next code data byte at the port P1 pins.
5. Repeat steps 3 and 4 until the entire array is read.

The lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

Chip Erase: The entire PEROM array (4K bytes) and the two Lock Bits are erased electrically by using the proper combination of control signals and by holding P3.2 low for 10 ms. The code array is written with all "1"s in the Chip Erase operation and must be executed before any non-blank memory byte can be re-programmed.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 000H, 001H, and 002H, except that P3.5 and P3.7 must be pulled to a logic low. The values returned are as follows.

(000H) = 1EH indicates manufactured by Atmel

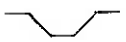
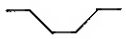
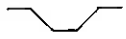
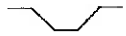
(001H) = 41H indicates 89C4051

Programming Interface

Every code byte in the Flash array can be written and the entire array can be erased by using the appropriate combination of control signals. The write operation cycle is self-timed and once initiated, will automatically time itself to completion.

All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision.

Flash Programming Modes

Mode		RST/V_{PP}	P3.2/$\overline{PROG}$	P3.3	P3.4	P3.5	P3.7
Write Code Data[1][3]		12V		L	H	H	H
Read Code Data[1]		H	H	L	L	H	H
Write Lock	Bit - 1	12V		H	H	H	H
	Bit - 2	12V		H	H	L	L
Chip Erase		12V	 (2)	H	L	L	L
Read Signature Byte		H	H	L	L	L	L

Notes:
1. The internal PEROM address counter is reset to 000H on the rising edge of RST and is advanced by a positive pulse at XTAL 1 pin.
2. Chip Erase requires a 10 ms $\overline{PROG}$ pulse.
3. P3.1 is pulled Low during programming to indicate RDY/$\overline{BSY}$.

Figure 3. Programming the Flash Memory

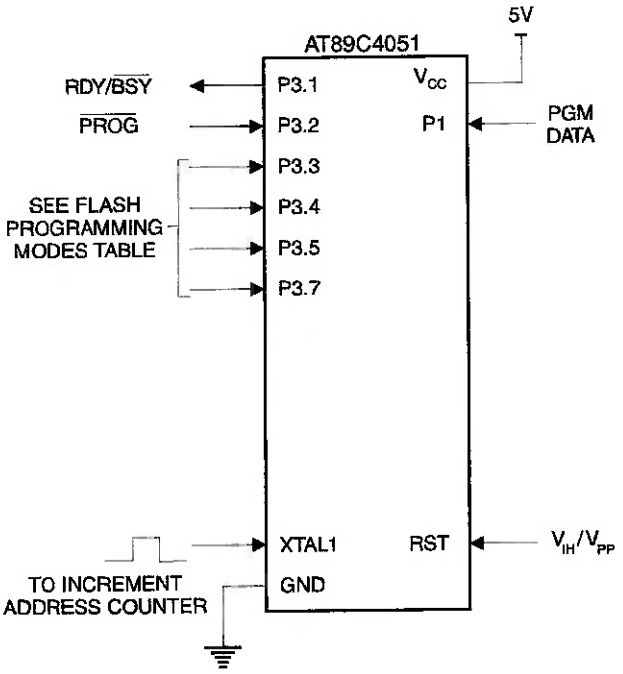
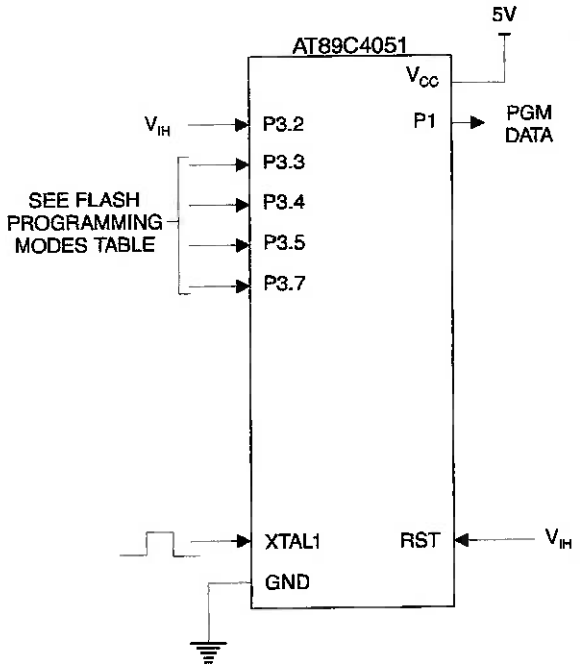


Figure 4. Verifying the Flash Memory



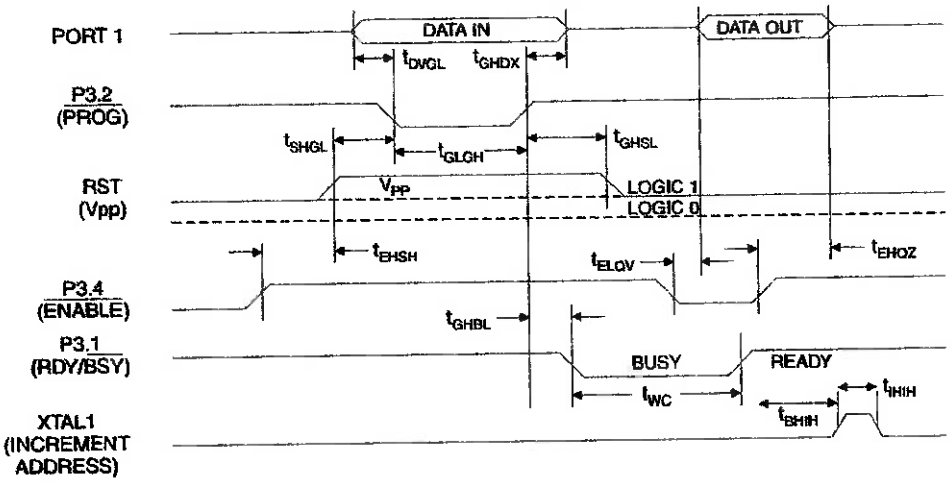
Flash Programming and Verification Characteristics

T_A = 0°C to 70°C, V_{CC} = 5.0 ± 10%

Symbol	Parameter	Min	Max	Units
V_{PP}	Programming Enable Voltage	11.5	12.5	V
I_{PP}	Programming Enable Current		250	µA
t_{DVGL}	Data Setup to $\overline{PROG}$ Low	1.0		µs
t_{GHDX}	Data Hold after $\overline{PROG}$	1.0		µs
t_{EHSH}	P3.4 ($\overline{ENABLE}$) High to V_{PP}	1.0		µs
t_{SHGL}	V_{PP} Setup to $\overline{PROG}$ Low	10		µs
t_{GHSL}	V_{PP} Hold after $\overline{PROG}$	10		µs
t_{GLGH}	$\overline{PROG}$ Width	1	110	µs
t_{ELQV}	$\overline{ENABLE}$ Low to Data Valid		1.0	µs
t_{EHQZ}	Data Float after $\overline{ENABLE}$	0	1.0	µs
t_{GHBL}	$\overline{PROG}$ High to $\overline{BUSY}$ Low		50	ns
t_{WC}	Byte Write Cycle Time		2.0	ms
t_{BHIH}	RDY/$\overline{BSY}$\ to Increment Clock Delay	1.0		µs
t_{IHIL}	Increment Clock High	200		ns

Note: 1. Only used in 12-volt programming mode.

Flash Programming and Verification Waveforms



Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-1.0V to +7.0V
Maximum Operating Voltage	6.6V
DC Output Current	25.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

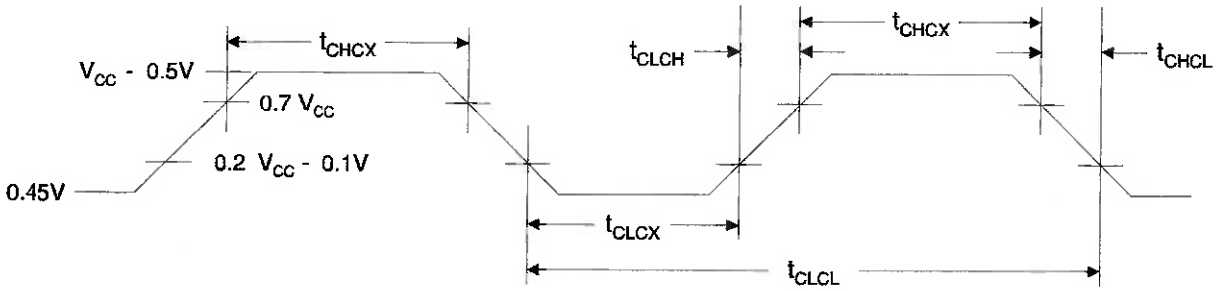
DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 3.0V to 6.0V (unless otherwise noted)

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low-voltage		-0.5	0.2 V_{CC} - 0.1	V
V_{IH}	Input High-voltage	(Except XTAL1, RST)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V
V_{IH1}	Input High-voltage	(XTAL1, RST)	0.7 V_{CC}	V_{CC} + 0.5	V
V_{OL}	Output Low-voltage[1] (Ports 1, 3)	I_{OL} = 20 mA, V_{CC} = 5V I_{OL} = 10 mA, V_{CC} = 2.7V		0.5	V
V_{OH}	Output High-voltage (Ports 1, 3)	I_{OH} = -80 µA, V_{CC} = 5V ± 10%	2.4		V
		I_{OH} = -30 µA	0.75 V_{CC}		V
		I_{OH} = -12 µA	0.9 V_{CC}		V
I_{IL}	Logical 0 Input Current (Ports 1, 3)	V_{IN} = 0.45V		-50	µA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1, 3)	V_{IN} = 2V, V_{CC} = 5V ± 10%		-750	µA
I_{LI}	Input Leakage Current (Port P1.0, P1.1)	0 < V_{IN} < V_{CC}		±10	µA
V_{OS}	Comparator Input Offset Voltage	V_{CC} = 5V		20	mV
V_{CM}	Comparator Input Common Mode Voltage		0	V_{CC}	V
RRST	Reset Pulldown Resistor		50	300	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz, V_{CC} = 6V/3V		15/5.5	mA
		Idle Mode, 12 MHz, V_{CC} = 6V/3V P1.0 & P1.1 = 0V or V_{CC}		5/1	mA
	Power-down Mode[2]	V_{CC} = 6V P1.0 & P1.1 = 0V or V_{CC}		100	µA
		V_{CC} = 3V P1.0 & P1.1 = 0V or V_{CC}		20	µA

Notes:
1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:
   Maximum I_{OL} per port pin: 20 mA
   Maximum total I_{OL} for all output pins: 80 mA
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
2. Minimum V_{CC} for Power-down is 2V.

External Clock Drive Waveforms



External Clock Drive

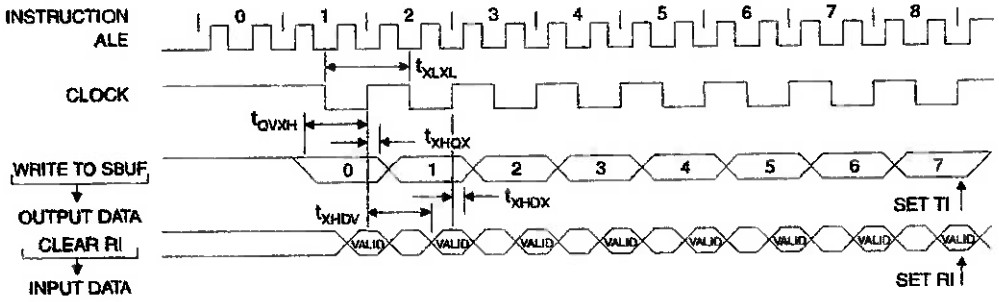
Symbol	Parameter	V_{CC} = 3.0V to 6.0V		V_{CC} = 4.0V to 6.0V		Units
		Min	Max	Min	Max	
$1/t_{CLCL}$	Oscillator Frequency	0	12	0	24	MHz
t_{CLCL}	Clock Period	83.3		41.6		ns
t_{CHCX}	High Time	30		15		ns
t_{CLCX}	Low Time	30		15		ns
t_{CLCH}	Rise Time		20		20	ns
t_{CHCL}	Fall Time		20		20	ns

Serial Port Timing: Shift Register Mode Test Conditions

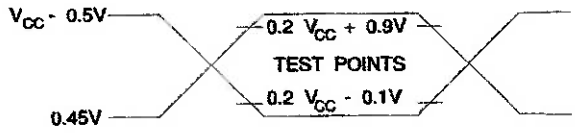
V_{CC} = 5.0V ± 20%; Load Capacitance = 80 pF

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
t_{XLXL}	Serial Port Clock Cycle Time	1.0		$12t_{CLCL}$		µs
t_{QVXH}	Output Data Setup to Clock Rising Edge	700		$10t_{CLCL}$-133		ns
t_{XHQX}	Output Data Hold after Clock Rising Edge	50		$2t_{CLCL}$-117		ns
t_{XHDX}	Input Data Hold after Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		700		$10t_{CLCL}$-133	ns

Shift Register Mode Timing Waveforms



AC Testing Input/Output Waveforms[1]



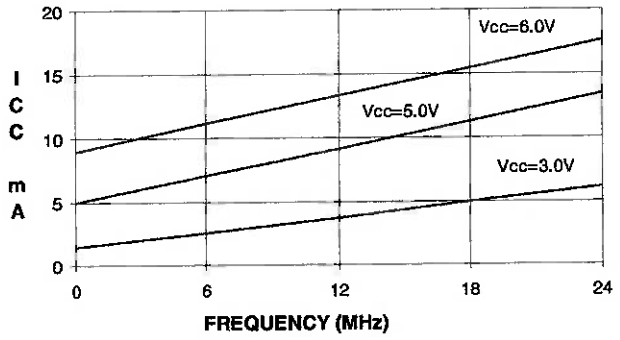
Note: 1. AC Inputs during testing are driven at V_{CC} - 0.5V for a logic 1 and 0.45V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

Float Waveforms[1]

Note: 1. For timing purposes, a port pin is no longer floating when a 100 mV change from load voltage occurs. A port pin begins to float when 100 mV change frothe loaded V_{OH}/V_{OL} level occurs.

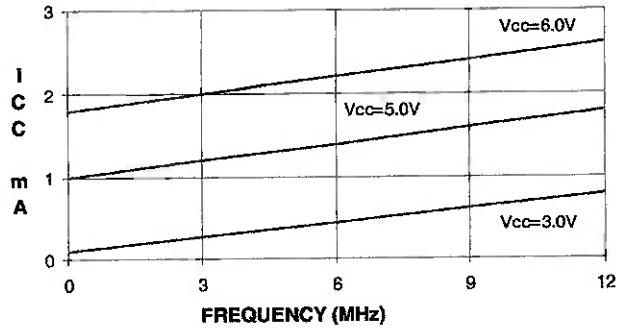
AT89C4051

TYPICAL ICC - ACTIVE (85°C)



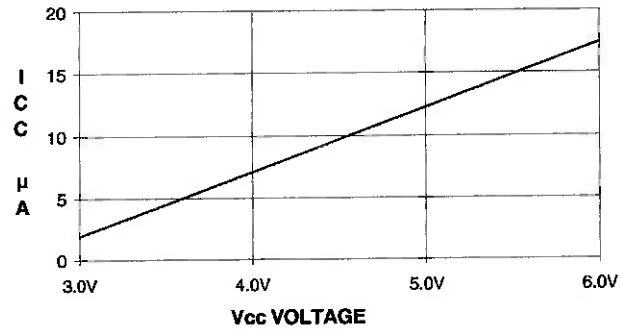
AT89C4051

TYPICAL ICC - IDLE (85°C)



AT89C4051

TYPICAL ICC vs. VOLTAGE- POWER DOWN (85°C)



Notes:
1. XTAL1 tied to GND for I_{CC} (power-down)
2. P1.0 and P1.1 = V_{CC} or GND
3. Lock bits programmed

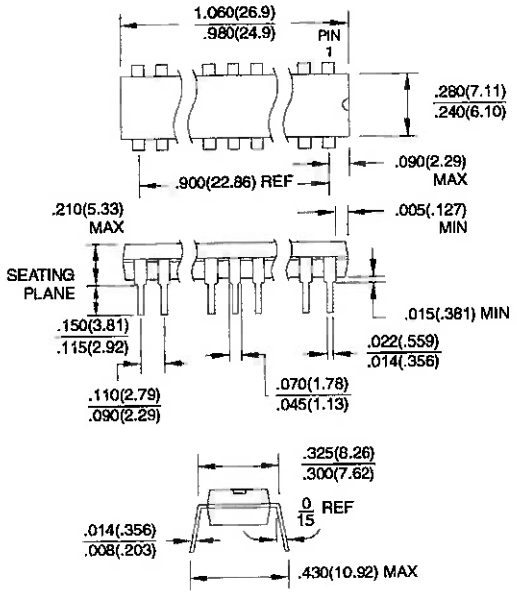
Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
12	3.0V to 6.0V	AT89C4051-12PC AT89C4051-12SC	20P3 20S	Commercial (0°C to 70°C)
		AT89C4051-12PI AT89C4051-12SI	20P3 20S	Industrial (-40°C to 85°C)
24	4.0V to 6.0V	AT89C4051-24PC AT89C4051-24SC	20P3 20S	Commercial (0°C to 70°C)
		AT89C4051-24PI AT89C4051-24SI	20P3 20S	Industrial (-40°C to 85°C)

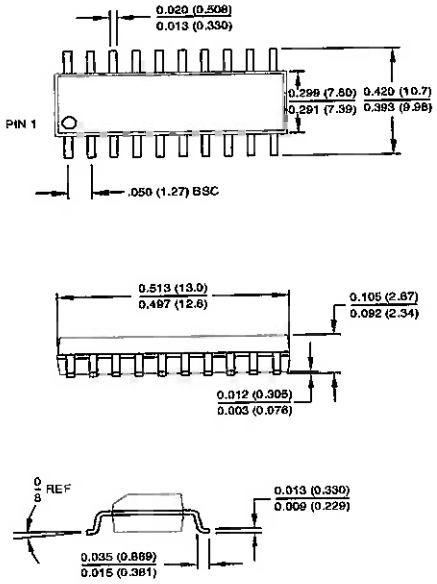
Package Type	
20P3	20-lead, 0.300" Wide, Plastic Dual In-line Package (PDIP)
20S	20-lead, 0.300" Wide, Plastic Gull Wing Small Outline (SOIC)

Packaging Information

20P3, 20-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)
Dimensions in Inches and (Millimeters)
JEDEC STANDARD MS-001 AD



20S, 20-lead, 0.300" Wide, Plastic Gull Wing Small Outline (SOIC)
Dimensions in Inches and (Millimeters)



Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686-677
FAX (44) 1276-686-697

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

1001C–02/00/xM

ANEXO II –

Datasheet do componente MAX233

19-4323; Rev 9; 5/00

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers

Low-Power Modems

Interface Translation

Battery-Powered RS-232 Systems

Multidrop RS-232 Networks

Features

Superior to Bipolar

- **Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)**
- **Low-Power Receive Mode in Shutdown (MAX223/MAX242)**
- **Meet All EIA/TIA-232E and V.28 Specifications**
- **Multiple Drivers and Receivers**
- **3-State Driver and Receiver Outputs**
- **Open-Line Detection (MAX243)**

Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

**Contact factory for dice specifications.*

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	0.1	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✔	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✔	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1488
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✔	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✔	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✔	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✔	120	Available in quad flatpack package

ABSOLUTE MAXIMUM RATINGS—MAX220/222/232A/233A/242/243

Supply Voltage (V_{CC}) ..... -0.3V to +6V
Input Voltages
T_{IN} ..... -0.3V to (V_{CC} - 0.3V)
R_{IN} (Except MAX220) ..... ±30V
R_{IN} (MAX220) ..... ±25V
T_{OUT} (Except MAX220) (Note 1) ..... ±15V
T_{OUT} (MAX220) ..... ±13.2V
Output Voltages
T_{OUT} ..... ±15V
R_{OUT} ..... -0.3V to (V_{CC} + 0.3V)
Driver/Receiver Output Short Circuited to GND ..... Continuous
Continuous Power Dissipation (T_A = +70°C)
16-Pin Plastic DIP (derate 10.53mW/°C above +70°C) 842mW
18-Pin Plastic DIP (derate 11.11mW/°C above +70°C) 889mW
20-Pin Plastic DIP (derate 8.00mW/°C above +70°C) 440mW
16-Pin Narrow SO (derate 8.70mW/°C above +70°C) 696mW
16-Pin Wide SO (derate 9.52mW/°C above +70°C) 762mW
18-Pin Wide SO (derate 9.52mW/°C above +70°C) 762mW
20-Pin Wide SO (derate 10.00mW/°C above +70°C) 800mW
20-Pin SSOP (derate 8.00mW/°C above +70°C) 640mW
16-Pin CERDIP (derate 10.00mW/°C above +70°C) 800mW
18-Pin CERDIP (derate 10.53mW/°C above +70°C) 842mW
Operating Temperature Ranges
MAX2_ _AC_ _, MAX2_ _C_ _ ..... 0°C to +70°C
MAX2_ _AE_ _, MAX2_ _E_ _ ..... -40°C to +85°C
MAX2_ _AM_ _, MAX2_ _M_ _ ..... -55°C to +125°C
Storage Temperature Range ..... -65°C to +160°C
Lead Temperature (soldering, 10sec) ..... +300°C

Note 1: Input voltage measured with T_{OUT} in high-impedance state, $\overline{SHDN}$ or V_{CC} = 0V.
Note 2: For the MAX220, V+ and V- can have a maximum magnitude of 7V, but their absolute difference cannot exceed 13V.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, MAX220, C1 = 0.047µF, C2–C4 = 0.33µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±8		V
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High	All devices except MAX220		2	1.4		V
	MAX220: V_{CC} = 5.0V		2.4			
Logic Pull-Up/Input Current	All except MAX220, normal operation			5	40	µA
	$\overline{SHDN}$ = 0V, MAX222/242, shutdown, MAX220			±0.01	±1	
Output Leakage Current	V_{CC} = 5.5V, $\overline{SHDN}$ = 0V, V_{OUT} = ±15V, MAX222/242			±0.01	±10	µA
	V_{CC} = $\overline{SHDN}$ = 0V, V_{OUT} = ±15V			±0.01	±10	
Data Rate				200	116	kb/s
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V		300	10M		Ω
Output Short-Circuit Current	V_{OUT} = 0V		±7	±22		mA
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±30	V
RS-232 Input Threshold Low	V_{CC} = 5V	All except MAX243 $R2_{IN}$	0.8	1.3		V
		MAX243 $R2_{IN}$ (Note 2)	-3			
RS-232 Input Threshold High	V_{CC} = 5V	All except MAX243 $R2_{IN}$		1.8	2.4	V
		MAX243 $R2_{IN}$ (Note 2)		-0.5	-0.1	
RS-232 Input Hysteresis	All except MAX243, V_{CC} = 5V, no hysteresis in shdn.		0.2	0.5	1	V
	MAX243			1		
RS-232 Input Resistance			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 3.2mA			0.2	0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5	V_{CC} - 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{OUT} = GND		-2	-10		mA
	Shrinking V_{OUT} = V_{CC}		10	30		

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243 (continued)

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, MAX220, C1 = 0.047µF, C2–C4 = 0.33µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

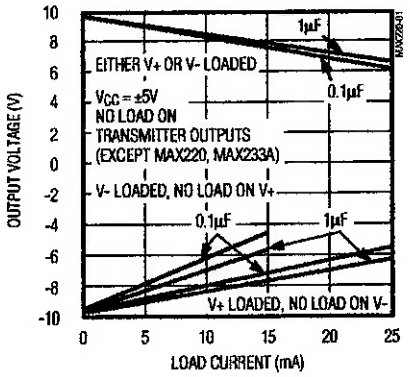
PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
TTL/CMOS Output Leakage Current	$\overline{SHDN}$ = V_{CC} or $\overline{EN}$ = V_{CC} ($\overline{SHDN}$ = 0V for MAX222), 0V ≤ V_{OUT} ≤ V_{CC}			±0.05	±10	µA
$\overline{EN}$ Input Threshold Low	MAX242			1.4	0.8	V
$\overline{EN}$ Input Threshold High	MAX242		2.0	1.4		V
Operating Supply Voltage			4.5		5.5	V
V_{CC} Supply Current ($\overline{SHDN}$ = V_{CC}), Figures 5, 6, 11, 19	No load	MAX220		0.5	2	mA
		MAX222/232A/233A/242/243		4	10	
	3kΩ load both inputs	MAX220		12		
		MAX222/232A/233A/242/243		15		
Shutdown Supply Current	MAX222/242	T_A = +25°C		0.1	10	µA
		T_A = 0°C to +70°C		2	50	
		T_A = -40°C to +85°C		2	50	
		T_A = -55°C to +125°C		35	100	
$\overline{SHDN}$ Input Leakage Current	MAX222/242				±1	µA
$\overline{SHDN}$ Threshold Low	MAX222/242			1.4	0.8	V
$\overline{SHDN}$ Threshold High	MAX222/242		2.0	1.4		V
Transition Slew Rate	C_L = 50pF to 2500pF, R_L = 3kΩ to 7kΩ, V_{CC} = 5V, T_A = +25°C, measured from +3V to -3V or -3V to +3V	MAX222/232A/233A/242/243	6	12	30	V/µs
		MAX220	1.5	3	30	
Transmitter Propagation Delay TLL to RS-232 (normal operation), Figure 1	t_{PHLT}	MAX222/232A/233A/242/243		1.3	3.5	µs
		MAX220		4	10	
	t_{PLHT}	MAX222/232A/233A/242/243		1.5	3.5	
		MAX220		5	10	
Receiver Propagation Delay RS-232 to TLL (normal operation), Figure 2	t_{PHLR}	MAX222/232A/233A/242/243		0.5	1	µs
		MAX220		0.6	3	
	t_{PLHR}	MAX222/232A/233A/242/243		0.6	1	
		MAX220		0.8	3	
Receiver Propagation Delay RS-232 to TLL (shutdown), Figure 2	t_{PHLS}	MAX242		0.5	10	µs
	t_{PLHS}	MAX242		2.5	10	
Receiver-Output Enable Time, Figure 3	t_{ER}	MAX242		125	500	ns
Receiver-Output Disable Time, Figure 3	t_{DR}	MAX242		160	500	ns
Transmitter-Output Enable Time ($\overline{SHDN}$ goes high), Figure 4	t_{ET}	MAX222/242, 0.1µF caps (includes charge-pump start-up)		250		µs
Transmitter-Output Disable Time ($\overline{SHDN}$ goes low), Figure 4	t_{DT}	MAX222/242, 0.1µF caps		600		ns
Transmitter + to - Propagation Delay Difference (normal operation)	t_{PHLT} - t_{PLHT}	MAX222/232A/233A/242/243		300		ns
		MAX220		2000		
Receiver + to - Propagation Delay Difference (normal operation)	t_{PHLR} - t_{PLHR}	MAX222/232A/233A/242/243		100		ns
		MAX220		225		

Note 3: MAX243 $R2_{OUT}$ is guaranteed to be low when $R2_{IN}$ is ≥ 0V or is floating.

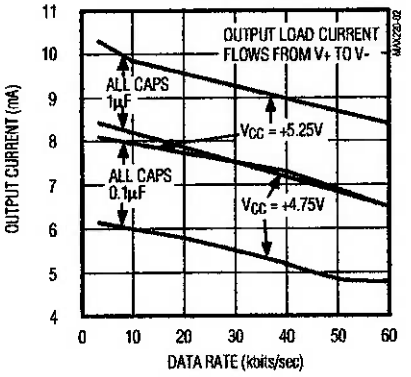
Typical Operating Characteristics

MAX220/MAX222/MAX232A/MAX233A/MAX242/MAX243

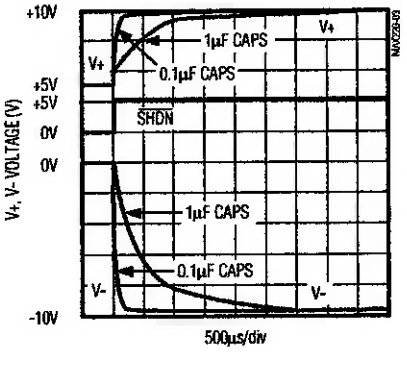
OUTPUT VOLTAGE vs. LOAD CURRENT



AVAILABLE OUTPUT CURRENT vs. DATA RATE



MAX222/MAX242 ON-TIME EXITING SHUTDOWN



ABSOLUTE MAXIMUM RATINGS—MAX223/MAX230–MAX241

V_{CC} ..... -0.3V to +6V
V+ ..... (V_{CC} - 0.3V) to +14V
V- ..... +0.3V to -14V
Input Voltages
T_{IN} ..... -0.3V to (V_{CC} + 0.3V)
R_{IN} ..... ±30V
Output Voltages
T_{OUT} ..... (V+ + 0.3V) to (V- - 0.3V)
R_{OUT} ..... -0.3V to (V_{CC} + 0.3V)
Short-Circuit Duration, T_{OUT} ..... Continuous
Continuous Power Dissipation (T_A = +70°C)
14-Pin Plastic DIP (derate 10.00mW/°C above +70°C)....800mW
16-Pin Plastic DIP (derate 10.53mW/°C above +70°C)....842mW
20-Pin Plastic DIP (derate 11.11mW/°C above +70°C)....889mW
24-Pin Narrow Plastic DIP
(derate 13.33mW/°C above +70°C)..... 1.07W
24-Pin Plastic DIP (derate 9.09mW/°C above +70°C)..... 500mW
16-Pin Wide SO (derate 9.52mW/°C above +70°C)..... 762mW
20-Pin Wide SO (derate 10 00mW/°C above +70°C)..... 800mW
24-Pin Wide SO (derate 11.76mW/°C above +70°C)..... 941mW
28-Pin Wide SO (derate 12.50mW/°C above +70°C) 1W
44-Pin Plastic FP (derate 11.11mW/°C above +70°C) 889mW
14-Pin CERDIP (derate 9.09mW/°C above +70°C)..... 727mW
16-Pin CERDIP (derate 10.00mW/°C above +70°C)..... 800mW
20-Pin CERDIP (derate 11.11mW/°C above +70°C)..... 889mW
24-Pin Narrow CERDIP
(derate 12.50mW/°C above +70°C) 1W
24-Pin Sidebraze (derate 20.0mW/°C above +70°C)..... 1.6W
28-Pin SSOP (derate 9.52mW/°C above +70°C)..... 762mW
Operating Temperature Ranges
MAX2 _ _ C _ _ ..... 0°C to +70°C
MAX2 _ _ E _ _ ..... -40°C to +85°C
MAX2 _ _ M _ _ ..... -55°C to +125°C
Storage Temperature Range ..... -65°C to +160°C
Lead Temperature (soldering, 10sec) ..... +300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX223/MAX230–MAX241

(MAX223/230/232/234/236/237/238/240/241, V_{CC} = +5V ±10; MAX233/MAX235, V_{CC} = 5V ±5%, C1–C4 = 1.0µF; MAX231/MAX239, V_{CC} = 5V ±10%; V+ = 7.5V to 13.2V; T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to ground		±5.0	±7.3		V
V_{CC} Power-Supply Current	No load, T_A = +25°C	MAX232/233		5	10	mA
		MAX223/230/234–238/240/241		7	15	
		MAX231/239		0.4	1	
V+ Power-Supply Current		MAX231		1.8	5	mA
		MAX239		5	15	
Shutdown Supply Current	T_A = +25°C	MAX223		15	50	µA
		MAX230/235/236/240/241		1	10	
Input Logic Threshold Low	T_{IN}; EN, $\overline{SHDN}$ (MAX233); $\overline{EN}$, SHDN (MAX230/235–241)				0.8	V
Input Logic Threshold High	T_{IN}		2.0			V
	EN, $\overline{SHDN}$ (MAX223); $\overline{EN}$, SHDN (MAX230/235/236/240/241)		2.4			
Logic Pull-Up Current	T_{IN} = 0V			1.5	200	µA
Receiver Input Voltage Operating Range			-30		30	V

ELECTRICAL CHARACTERISTICS—MAX223/MAX230–MAX241 (continued)

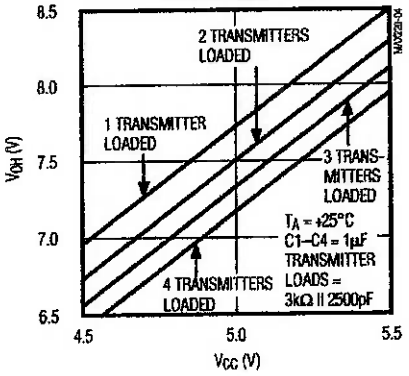
(MAX223/230/232/234/236/237/238/240/241, V_{CC} = +5V ±10; MAX233/MAX235, V_{CC} = 5V ±5%, C1–C4 = 1.0µF; MAX231/MAX239, V_{CC} = 5V ±10%; V+ = 7.5V to 13.2V; T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

PARAMETER	CONDITIONS			MIN	TYP	MAX	UNITS
RS-232 Input Threshold Low	T_A = +25°C, V_{CC} = 5V	Normal operation $\overline{SHDN}$ = 5V (MAX223) SHDN = 0V (MAX235/236/240/241)		0.8	1.2		V
		Shutdown (MAX223) $\overline{SHDN}$ = 0V, EN = 5V ($R4_{IN}$, $R5_{IN}$)		0.6	1.5		
RS-232 Input Threshold High	T_A = +25°C, V_{CC} = 5V	Normal operation $\overline{SHDN}$ = 5V (MAX223) SHDN = 0V (MAX235/236/240/241)			1.7	2.4	V
		Shutdown (MAX223) $\overline{SHDN}$ = 0V, EN = 5V ($R4_{IN}$, $R5_{IN}$)			1.5	2.4	
RS-232 Input Hysteresis	V_{CC} = 5V, no hysteresis in shutdown			0.2	0.5	1.0	V
RS-232 Input Resistance	T_A = +25°C, V_{CC} = 5V			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 1.6mA (MAX231/232/233, I_{OUT} = 3.2mA)					0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1mA			3.5	V_{CC} - 0.4		V
TTL/CMOS Output Leakage Current	0V ≤ R_{OUT} ≤ V_{CC}; EN = 0V (MAX223); $\overline{EN}$ = V_{CC} (MAX235–241)				0.05	±10	µA
Receiver Output Enable Time	Normal operation	MAX223			600		ns
		MAX235/236/239/240/241			400		
Receiver Output Disable Time	Normal operation	MAX223			900		ns
		MAX235/236/239/240/241			250		
Propagation Delay	RS-232 IN to TTL/CMOS OUT, C_L = 150pF	Normal operation			0.5	10	µs
		$\overline{SHDN}$ = 0V (MAX223)	t_{PHLS}		4	40	
			t_{PLHS}		6	40	
Transition Region Slew Rate	MAX223/MAX230/MAX234–241, T_A = +25°C, V_{CC} = 5V, R_L = 3kΩ to 7kΩ, C_L = 50pF to 2500pF, measured from +3V to -3V or -3V to +3V			3	5.1	30	V/µs
	MAX231/MAX232/MAX233, T_A = +25°C, V_{CC} = 5V, R_L = 3kΩ to 7kΩ, C_L = 50pF to 2500pF, measured from +3V to -3V or -3V to +3V				4	30	
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V			300			Ω
Transmitter Output Short-Circuit Current					±10		mA

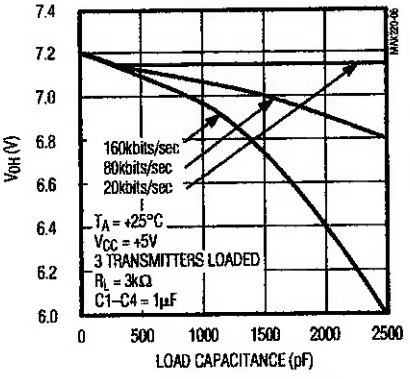
Typical Operating Characteristics

MAX223/MAX230–MAX241

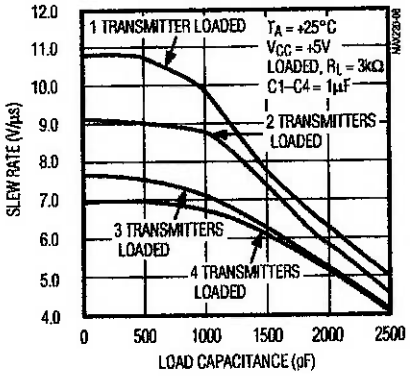
TRANSMITTER OUTPUT VOLTAGE (V_{OH}) vs. V_{CC}



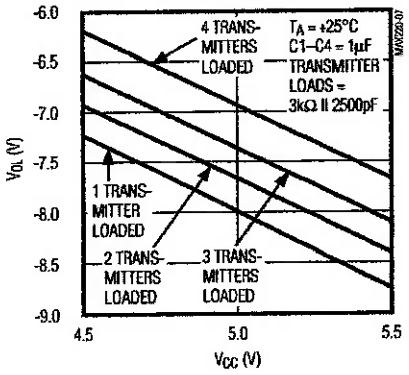
TRANSMITTER OUTPUT VOLTAGE (V_{OH}) vs. LOAD CAPACITANCE AT DIFFERENT DATA RATES



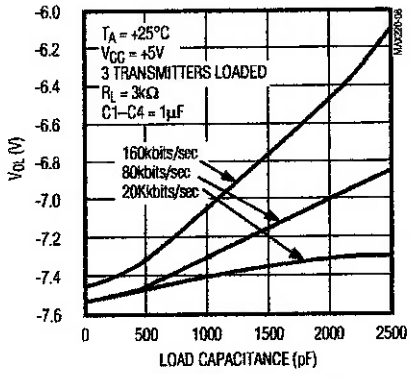
TRANSMITTER SLEW RATE vs. LOAD CAPACITANCE



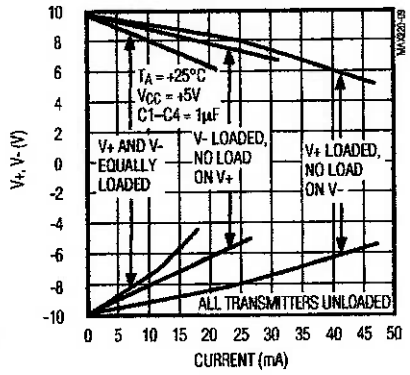
TRANSMITTER OUTPUT VOLTAGE (V_{OL}) vs. V_{CC}



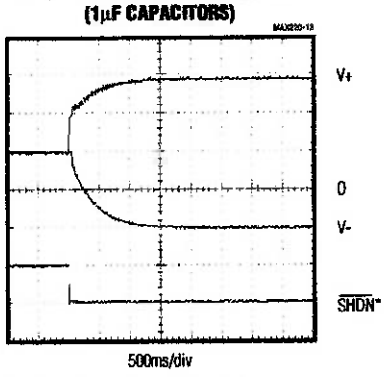
TRANSMITTER OUTPUT VOLTAGE (V_{OL}) vs. LOAD CAPACITANCE AT DIFFERENT DATA RATES



TRANSMITTER OUTPUT VOLTAGE (V+, V-) vs. LOAD CURRENT



V+, V- WHEN EXITING SHUTDOWN (1µF CAPACITORS)



*SHUTDOWN POLARITY IS REVERSED FOR NON MAX241 PARTS

ABSOLUTE MAXIMUM RATINGS—MAX225/MAX244–MAX249

Supply Voltage (V_{CC}) ..... -0.3V to +6V
Input Voltages
T_{IN}, $\overline{ENA}$, $\overline{ENB}$, $\overline{ENR}$, $\overline{ENT}$, $\overline{ENRA}$, $\overline{ENRB}$, $\overline{ENTA}$, $\overline{ENTB}$ ..... -0.3V to (V_{CC} + 0.3V)
R_{IN} ..... ±25V
T_{OUT} (Note 3) ..... ±15V
R_{OUT} ..... -0.3V to (V_{CC} + 0.3V)
Short Circuit (one output at a time)
T_{OUT} to GND ..... Continuous
R_{OUT} to GND ..... Continuous

Continuous Power Dissipation (T_A = +70°C)
28-Pin Wide SO (derate 12.50mW/°C above +70°C) 1W
40-Pin Plastic DIP (derate 11.11mW/°C above +70°C) ... 611mW
44-Pin PLCC (derate 13.33mW/°C above +70°C) 1.07W
Operating Temperature Ranges
MAX225C_ _, MAX24_C_ _ ..... 0°C to +70°C
MAX225E_ _, MAX24_E_ _ ..... -40°C to +85°C
Storage Temperature Range ..... -65°C to +160°C
Lead Temperature (soldering, 10sec) ..... +300°C

Note 4: Input voltage measured with transmitter output in a high-impedance state, shutdown, or V_{CC} = 0V.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX225/MAX244–MAX249

(MAX225, V_{CC} = 5.0V ±5%; MAX244–MAX249, V_{CC} = +5.0V ±10%, external capacitors C1–C4 = 1µF; T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High			2	1.4		V
Logic Pull-Up/Input Current	Tables 1a–1d	Normal operation		10	50	µA
		Shutdown		±0.01	±1	
Data Rate	Tables 1a–1d, normal operation			120	64	kbits/sec
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±7.5		V
Output Leakage Current (shutdown)	Tables 1a–1d	$\overline{ENA}$, $\overline{ENB}$, $\overline{ENT}$, $\overline{ENTA}$, $\overline{ENTB}$ = V_{CC}, V_{OUT} = ±15V		±0.01	±25	µA
		V_{CC} = 0V, V_{OUT} = ±15V		±0.01	±25	
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V (Note 4)		300	10M		Ω
Output Short-Circuit Current	V_{OUT} = 0V		±7	±30		mA
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±25	V
RS-232 Input Threshold Low	V_{CC} = 5V		0.8	1.3		V
RS-232 Input Threshold High	V_{CC} = 5V			1.8	2.4	V
RS-232 Input Hysteresis	V_{CC} = 5V		0.2	0.5	1.0	V
RS-232 Input Resistance			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 3.2mA			0.2	0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5	V_{CC} - 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{OUT} = GND		-2	-10		mA
	Shrinking V_{OUT} = V_{CC}		10	30		
TTL/CMOS Output Leakage Current	Normal operation, outputs disabled, Tables 1a–1d, 0V ≤ V_{OUT} ≤ V_{CC}, $\overline{ENR}$_ = V_{CC}			±0.05	±0.10	µA

ELECTRICAL CHARACTERISTICS—MAX225/MAX244–MAX249 (continued)

(MAX225, V_{CC} = 5.0V ±5%; MAX244–MAX249, V_{CC} = +5.0V ±10%, external capacitors C1–C4 = 1µF; T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

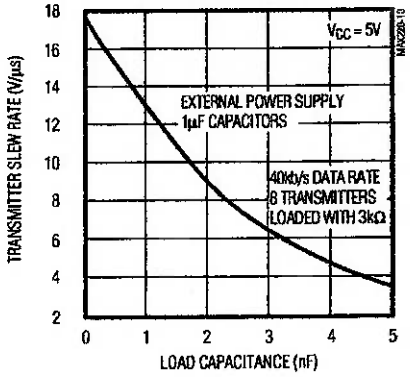
PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
POWER SUPPLY AND CONTROL LOGIC						
Operating Supply Voltage		MAX225	4.75		5.25	V
		MAX244–MAX249	4.5		5.5	
V_{CC} Supply Current (normal operation)	No load	MAX225		10	20	mA
		MAX244–MAX249		11	30	
	3kΩ loads on all outputs	MAX225		40		
		MAX244–MAX249		57		
Shutdown Supply Current	T_A = +25°C			8	25	µA
	T_A = T_{MIN} to T_{MAX}				50	
Control Input	Leakage current				±1	µA
	Threshold low			1.4	0.8	V
	Threshold high		2.4	1.4		
AC CHARACTERISTICS						
Transition Slew Rate	C_L = 50pF to 2500pF, R_L = 3kΩ to 7kΩ, V_{CC} = 5V, T_A = +25°C, measured from +3V to -3V or -3V to +3V		5	10	30	V/µs
Transmitter Propagation Delay TLL to RS-232 (normal operation), Figure 1	t_{PHLT}			1.3	3.5	µs
	t_{PLHT}			1.5	3.5	
Receiver Propagation Delay TLL to RS-232 (normal operation), Figure 2	t_{PHLR}			0.6	1.5	µs
	t_{PLHR}			0.6	1.5	
Receiver Propagation Delay TLL to RS-232 (low-power mode), Figure 2	t_{PHLS}			0.6	10	µs
	t_{PLHS}			3.0	10	
Transmitter + to - Propagation Delay Difference (normal operation)	t_{PHLT} - t_{PLHT}			350		ns
Receiver + to - Propagation Delay Difference (normal operation)	t_{PHLR} - t_{PLHR}			350		ns
Receiver-Output Enable Time, Figure 3	t_{ER}			100	500	ns
Receiver-Output Disable Time, Figure 3	t_{DR}			100	500	ns
Transmitter Enable Time	t_{ET}	MAX246–MAX249 (excludes charge-pump start-up)		5		µs
		MAX225/MAX245–MAX249 (includes charge-pump start-up)		10		ms
Transmitter Disable Time, Figure 4	t_{DT}			100		ns

Note 5: The 300Ω minimum specification complies with EIA/TIA-232E, but the actual resistance when in shutdown mode or V_{CC} = 0V is 10MΩ as is implied by the leakage specification.

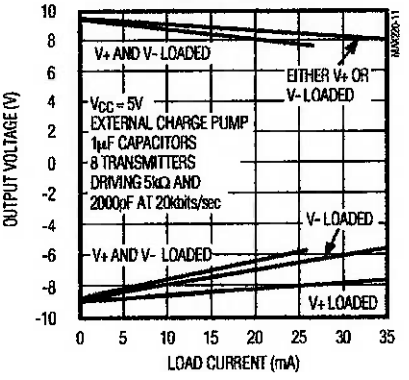
Typical Operating Characteristics

MAX225/MAX244–MAX249

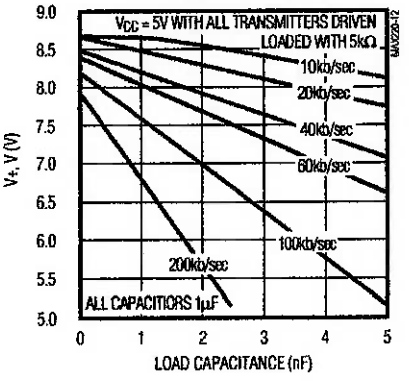
TRANSMITTER SLEW RATE vs. LOAD CAPACITANCE



OUTPUT VOLTAGE vs. LOAD CURRENT FOR V+ AND V-



TRANSMITTER OUTPUT VOLTAGE (V+, V-) vs. LOAD CAPACITANCE AT DIFFERENT DATA RATES



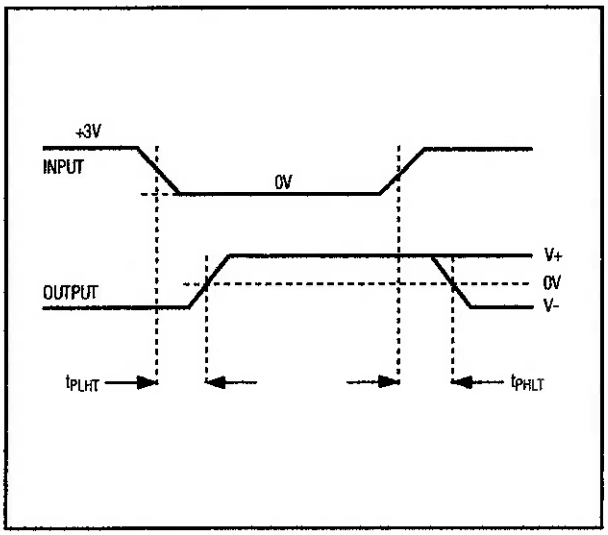


Figure 1. Transmitter Propagation-Delay Timing

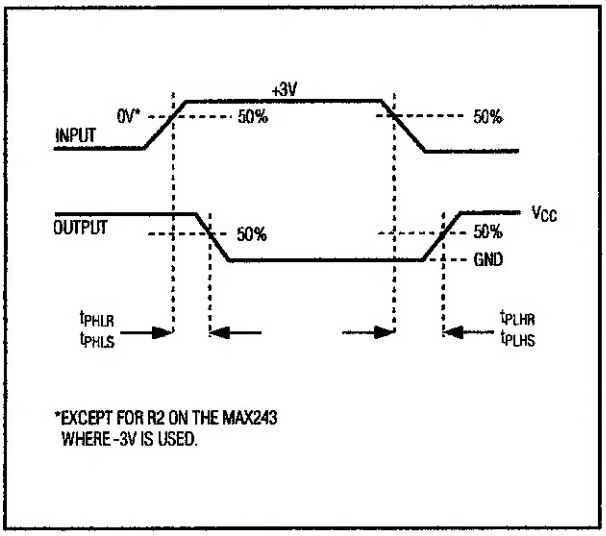


Figure 2. Receiver Propagation-Delay Timing

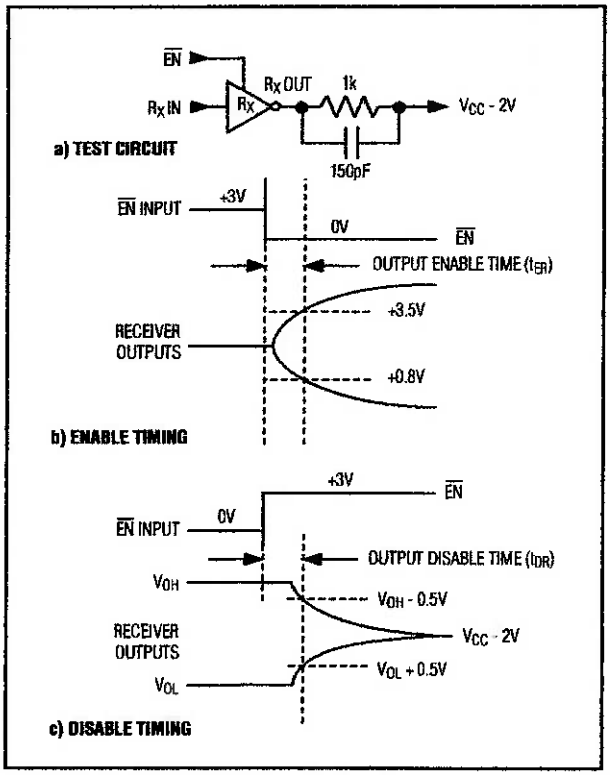


Figure 3. Receiver-Output Enable and Disable Timing

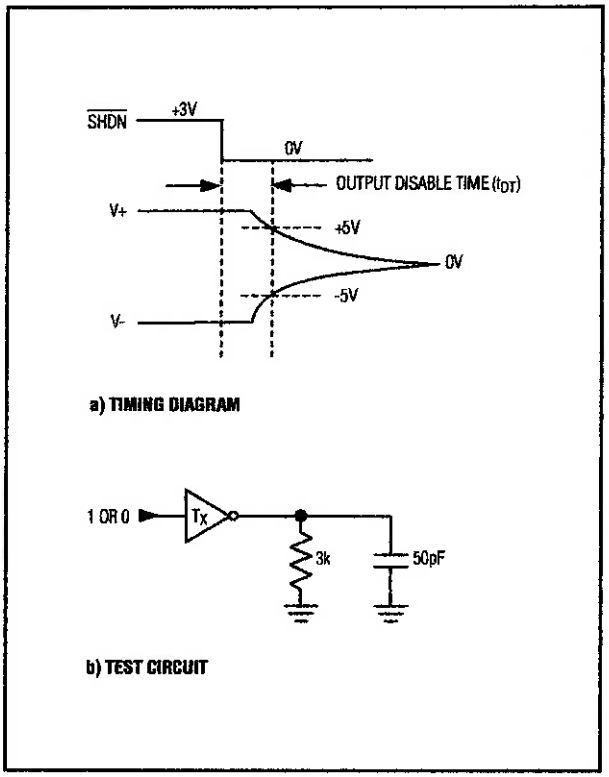


Figure 4. Transmitter-Output Disable Timing

Table 1a. MAX245 Control Pin Configurations

$\overline{ENT}$	$\overline{ENR}$	OPERATION STATUS	TRANSMITTERS	RECEIVERS
0	0	Normal Operation	All Active	All Active
0	1	Normal Operation	All Active	All 3-State
1	0	Shutdown	All 3-State	All Low-Power Receive Mode
1	1	Shutdown	All 3-State	All 3-State

Table 1b. MAX245 Control Pin Configurations

$\overline{ENT}$	$\overline{ENR}$	OPERATION STATUS	TRANSMITTERS		RECEIVERS	
			TA1–TA4	TB1–TB4	RA1–RA5	RB1–RB5
0	0	Normal Operation	All Active	All Active	All Active	All Active
0	1	Normal Operation	All Active	All Active	RA1–RA4 3-State, RA5 Active	RB1–RB4 3-State, RB5 Active
1	0	Shutdown	All 3-State	All 3-State	All Low-Power Receive Mode	All Low-Power Receive Mode
1	1	Shutdown	All 3-State	All 3-State	RA1–RA4 3-State, RA5 Low-Power Receive Mode	RB1–RB4 3-State, RB5 Low-Power Receive Mode

Table 1c. MAX246 Control Pin Configurations

$\overline{ENA}$	$\overline{ENB}$	OPERATION STATUS	TRANSMITTERS		RECEIVERS	
			TA1–TA4	TB1–TB4	RA1–RA5	RB1–RB5
0	0	Normal Operation	All Active	All Active	All Active	All Active
0	1	Normal Operation	All Active	All 3-State	All Active	RB1–RB4 3-State, RB5 Active
1	0	Shutdown	All 3-State	All Active	RA1–RA4 3-State, RA5 Active	All Active
1	1	Shutdown	All 3-State	All 3-State	RA1–RA4 3-State, RA5 Low-Power Receive Mode	RB1–RB4 3-State, RA5 Low-Power Receive Mode

Table 1d. MAX247/MAX248/MAX249 Control Pin Configurations

$\overline{ENTA}$	$\overline{ENTB}$	$\overline{ENRA}$	$\overline{ENRB}$	OPERATION STATUS		TRANSMITTERS		RECEIVERS	
					MAX247	TA1–TA4	TB1–TB4	RA1–RA4	RB1–RB5
					MAX248	TA1–TA4	TB1–TB4	RA1–RA4	RB1–RB4
					MAX249	TA1–TA3	TB1–TB3	RA1–RA5	RB1–RB5
0	0	0	0	Normal Operation		All Active	All Active	All Active	All Active
0	0	0	1	Normal Operation		All Active	All Active	All Active	All 3-State, except RB5 stays active on MAX247
0	0	1	0	Normal Operation		All Active	All Active	All 3-State	All Active
0	0	1	1	Normal Operation		All Active	All Active	All 3-State	All 3-State, except RB5 stays active on MAX247
0	1	0	0	Normal Operation		All Active	All 3-State	All Active	All Active
0	1	0	1	Normal Operation		All Active	All 3-State	All Active	All 3-State, except RB5 stays active on MAX247
0	1	1	0	Normal Operation		All Active	All 3-State	All 3-State	All Active
0	1	1	1	Normal Operation		All Active	All 3-State	All 3-State	All 3-State, except RB5 stays active on MAX247
1	0	0	0	Normal Operation		All 3-State	All Active	All Active	All Active
1	0	0	1	Normal Operation		All 3-State	All Active	All Active	All 3-State, except RB5 stays active on MAX247
1	0	1	0	Normal Operation		All 3-State	All Active	All 3-State	All Active
1	0	1	1	Normal Operation		All 3-State	All Active	All 3-State	All 3-State, except RB5 stays active on MAX247
1	1	0	0	Shutdown		All 3-State	All 3-State	Low-Power Receive Mode	Low-Power Receive Mode
1	1	0	1	Shutdown		All 3-State	All 3-State	Low-Power Receive Mode	All 3-State, except RB5 stays active on MAX247
1	1	1	0	Shutdown		All 3-State	All 3-State	All 3-State	Low-Power Receive Mode
1	1	1	1	Shutdown		All 3-State	All 3-State	All 3-State	All 3-State, except RB5 stays active on MAX247

Detailed Description

The MAX220–MAX249 contain four sections: dual charge-pump DC-DC voltage converters, RS-232 drivers, RS-232 receivers, and receiver and transmitter enable control inputs.

Dual Charge-Pump Voltage Converter

The MAX220–MAX249 have two internal charge-pumps that convert +5V to ±10V (unloaded) for RS-232 driver operation. The first converter uses capacitor C1 to double the +5V input to +10V on C3 at the V+ output. The second converter uses capacitor C2 to invert +10V to -10V on C4 at the V- output.

A small amount of power may be drawn from the +10V (V+) and -10V (V-) outputs to power external circuitry (see the *Typical Operating Characteristics* section), except on the MAX225 and MAX245–MAX247, where these pins are not available. V+ and V- are not regulated, so the output voltage drops with increasing load current. Do not load V+ and V- to a point that violates the minimum ±5V EIA/TIA-232E driver output voltage when sourcing current from V+ and V- to external circuitry.

When using the shutdown feature in the MAX222, MAX225, MAX230, MAX235, MAX236, MAX240, MAX241, and MAX245–MAX249, avoid using V+ and V- to power external circuitry. When these parts are shut down, V- falls to 0V, and V+ falls to +5V. For applications where a +10V external supply is applied to the V+ pin (instead of using the internal charge pump to generate +10V), the C1 capacitor must not be installed and the $\overline{SHDN}$ pin must be tied to V_{CC}. This is because V+ is internally connected to V_{CC} in shutdown mode.

RS-232 Drivers

The typical driver output voltage swing is ±8V when loaded with a nominal 5kΩ RS-232 receiver and V_{CC} = +5V. Output swing is guaranteed to meet the EIA/TIA-232E and V.28 specification, which calls for ±5V minimum driver output levels under worst-case conditions. These include a minimum 3kΩ load, V_{CC} = +4.5V, and maximum operating temperature. Unloaded driver output voltage ranges from (V+ -1.3V) to (V- +0.5V).

Input thresholds are both TTL and CMOS compatible. The inputs of unused drivers can be left unconnected since 400kΩ input pull-up resistors to V_{CC} are built in (except for the MAX220). The pull-up resistors force the outputs of unused drivers low because all drivers invert. The internal input pull-up resistors typically source 12µA, except in shutdown mode where the pull-ups are disabled. Driver outputs turn off and enter a high-impedance state—where leakage current is typically microamperes (maximum 25µA)—when in shutdown mode, in three-state mode, or when device power is removed. Outputs can be driven to ±15V. The power-supply current typically drops to 8µA in shutdown mode. The MAX220 does not have pull-up resistors to force the outputs of the unused drivers low. Connect unused inputs to GND or V_{CC}.

The MAX239 has a receiver three-state control line, and the MAX223, MAX225, MAX235, MAX236, MAX240, and MAX241 have both a receiver three-state control line and a low-power shutdown control. Table 2 shows the effects of the shutdown control and receiver three-state control on the receiver outputs.

The receiver TTL/CMOS outputs are in a high-impedance, three-state mode whenever the three-state enable line is high (for the MAX225/MAX235/MAX236/MAX239–MAX241), and are also high-impedance whenever the shutdown control line is high.

When in low-power shutdown mode, the driver outputs are turned off and their leakage current is less than 1µA with the driver output pulled to ground. The driver output leakage remains less than 1µA, even if the transmitter output is backdriven between 0V and (V_{CC} + 6V). Below -0.5V, the transmitter is diode clamped to ground with 1kΩ series impedance. The transmitter is also zener clamped to approximately V_{CC} + 6V, with a series impedance of 1kΩ.

The driver output slew rate is limited to less than 30V/µs as required by the EIA/TIA-232E and V.28 specifications. Typical slew rates are 24V/µs unloaded and 10V/µs loaded with 3Ω and 2500pF.

RS-232 Receivers

EIA/TIA-232E and V.28 specifications define a voltage level greater than 3V as a logic 0, so all receivers invert. Input thresholds are set at 0.8V and 2.4V, so receivers respond to TTL level inputs as well as EIA/TIA-232E and V.28 levels.

The receiver inputs withstand an input overvoltage up to ±25V and provide input terminating resistors with

Table 2. Three-State Control of Receivers

PART	SHDN	$\overline{SHDN}$	EN	$\overline{EN(R)}$	RECEIVERS
MAX223	—	Low High High	X Low High	—	High Impedance Active High Impedance
MAX225	—	—	—	Low High	High Impedance Active
MAX235 MAX236 MAX240	Low Low High	—	—	Low High X	High Impedance Active High Impedance

nominal 5kΩ values. The receivers implement Type 1 interpretation of the fault conditions of V.28 and EIA/TIA-232E.

The receiver input hysteresis is typically 0.5V with a guaranteed minimum of 0.2V. This produces clear output transitions with slow-moving input signals, even with moderate amounts of noise and ringing. The receiver propagation delay is typically 600ns and is independent of input swing direction.

Low-Power Receive Mode

The low-power receive-mode feature of the MAX223, MAX242, and MAX245–MAX249 puts the IC into shutdown mode but still allows it to receive information. This is important for applications where systems are periodically awakened to look for activity. Using low-power receive mode, the system can still receive a signal that will activate it on command and prepare it for communication at faster data rates. This operation conserves system power.

Negative Threshold—MAX243

The MAX243 is pin compatible with the MAX232A, differing only in that RS-232 cable fault protection is removed on one of the two receiver inputs. This means that control lines such as CTS and RTS can either be driven or left floating without interrupting communication. Different cables are not needed to interface with different pieces of equipment.

The input threshold of the receiver without cable fault protection is -0.8V rather than +1.4V. Its output goes positive only if the input is connected to a control line that is actively driven negative. If not driven, it defaults to the 0 or "OK to send" state. Normally, the MAX243's other receiver (+1.4V threshold) is used for the data line (TD or RD), while the negative threshold receiver is connected to the control line (DTR, DTS, CTS, RTS, etc.).

Other members of the RS-232 family implement the optional cable fault protection as specified by EIA/TIA-232E specifications. This means a receiver output goes high whenever its input is driven negative, left floating, or shorted to ground. The high output tells the serial communications IC to stop sending data. To avoid this, the control lines must either be driven or connected with jumpers to an appropriate positive voltage level.

Shutdown—MAX222–MAX242

On the MAX222, MAX235, MAX236, MAX240, and MAX241, all receivers are disabled during shutdown. On the MAX223 and MAX242, two receivers continue to operate in a reduced power mode when the chip is in shutdown. Under these conditions, the propagation delay increases to about 2.5µs for a high-to-low input transition. When in shutdown, the receiver acts as a CMOS inverter with no hysteresis. The MAX223 and MAX242 also have a receiver output enable input ($\overline{EN}$ for the MAX242 and EN for the MAX223) that allows receiver output control independent of $\overline{SHDN}$ (SHDN for MAX241). With all other devices, $\overline{SHDN}$ (SHDN for MAX241) also disables the receiver outputs.

The MAX225 provides five transmitters and five receivers, while the MAX245 provides ten receivers and eight transmitters. Both devices have separate receiver and transmitter-enable controls. The charge pumps turn off and the devices shut down when a logic high is applied to the ENT input. In this state, the supply current drops to less than 25µA and the receivers continue to operate in a low-power receive mode. Driver outputs enter a high-impedance state (three-state mode). On the MAX225, all five receivers are controlled by the $\overline{ENR}$ input. On the MAX245, eight of the receiver outputs are controlled by the $\overline{ENR}$ input, while the remaining two receivers (RA5 and RB5) are always active. RA1–RA4 and RB1–RB4 are put in a three-state mode when $\overline{ENR}$ is a logic high.

Receiver and Transmitter Enable Control Inputs

The MAX225 and MAX245–MAX249 feature transmitter and receiver enable controls.

The receivers have three modes of operation: full-speed receive (normal active), three-state (disabled), and low-power receive (enabled receivers continue to function at lower data rates). The receiver enable inputs control the full-speed receive and three-state modes. The transmitters have two modes of operation: full-speed transmit (normal active) and three-state (disabled). The transmitter enable inputs also control the shutdown mode. The device enters shutdown mode when all transmitters are disabled. Enabled receivers function in the low-power receive mode when in shutdown.

Tables 1a–1d define the control states. The MAX244 has no control pins and is not included in these tables.

The MAX246 has ten receivers and eight drivers with two control pins, each controlling one side of the device. A logic high at the A-side control input ($\overline{ENA}$) causes the four A-side receivers and drivers to go into a three-state mode. Similarly, the B-side control input ($\overline{ENB}$) causes the four B-side drivers and receivers to go into a three-state mode. As in the MAX245, one A-side and one B-side receiver (RA5 and RB5) remain active at all times. The entire device is put into shutdown mode when both the A and B sides are disabled ($\overline{ENA}$ = $\overline{ENB}$ = +5V).

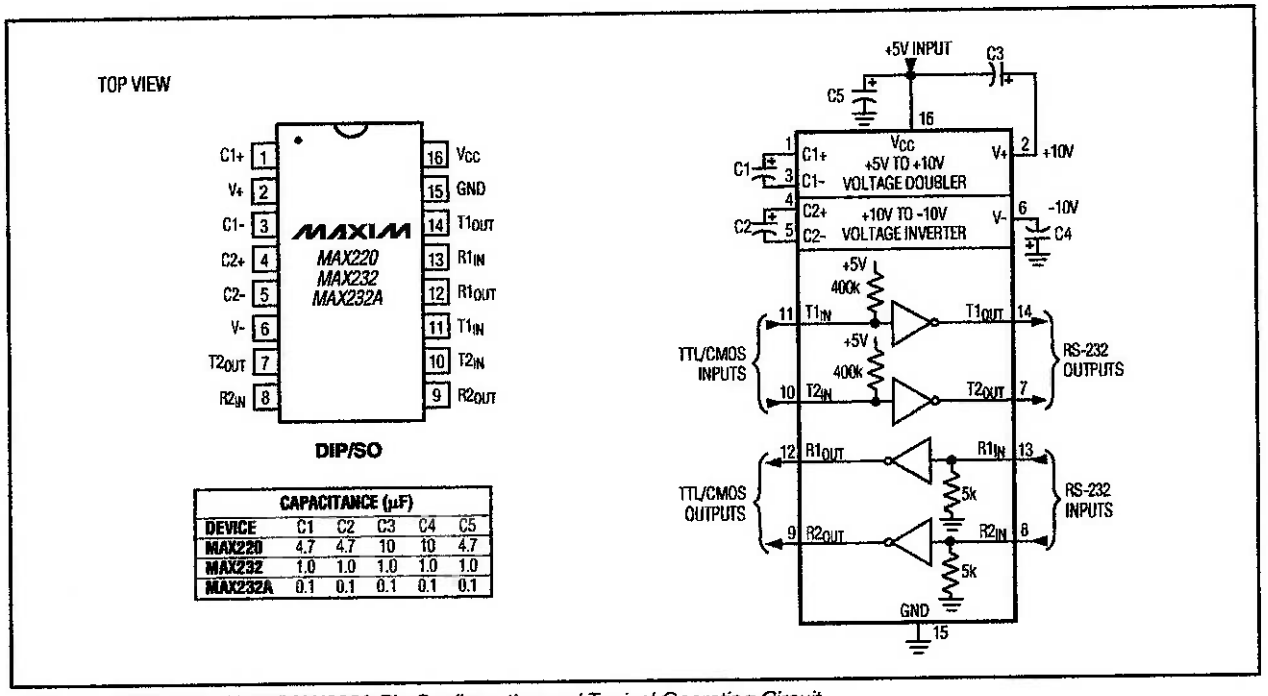
The MAX247 provides nine receivers and eight drivers with four control pins. The $\overline{ENRA}$ and $\overline{ENRB}$ receiver enable inputs each control four receiver outputs. The $\overline{ENTA}$ and $\overline{ENTB}$ transmitter enable inputs each control four drivers. The ninth receiver (RB5) is always active. The device enters shutdown mode with a logic high on both $\overline{ENTA}$ and $\overline{ENTB}$.

The MAX248 provides eight receivers and eight drivers with four control pins. The $\overline{ENRA}$ and $\overline{ENRB}$ receiver enable inputs each control four receiver outputs. The $\overline{ENTA}$ and $\overline{ENTB}$ transmitter enable inputs control four drivers each. This part does not have an always-active receiver. The device enters shutdown mode and transmitters go into a three-state mode with a logic high on both $\overline{ENTA}$ and $\overline{ENTB}$.

The MAX249 provides ten receivers and six drivers with four control pins. The $\overline{ENRA}$ and $\overline{ENRB}$ receiver enable inputs each control five receiver outputs. The $\overline{ENTA}$ and $\overline{ENTB}$ transmitter enable inputs control three drivers each. There is no always-active receiver. The device enters shutdown mode and transmitters go into a three-state mode with a logic high on both $\overline{ENTA}$ and $\overline{ENTB}$. In shutdown mode, active receivers operate in a low-power receive mode at data rates up to 20kbits/sec.

Applications Information

Figures 5 through 25 show pin configurations and typical operating circuits. In applications that are sensitive to power-supply noise, V_{CC} should be decoupled to ground with a capacitor of the same value as C1 and C2 connected as close as possible to the device.



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

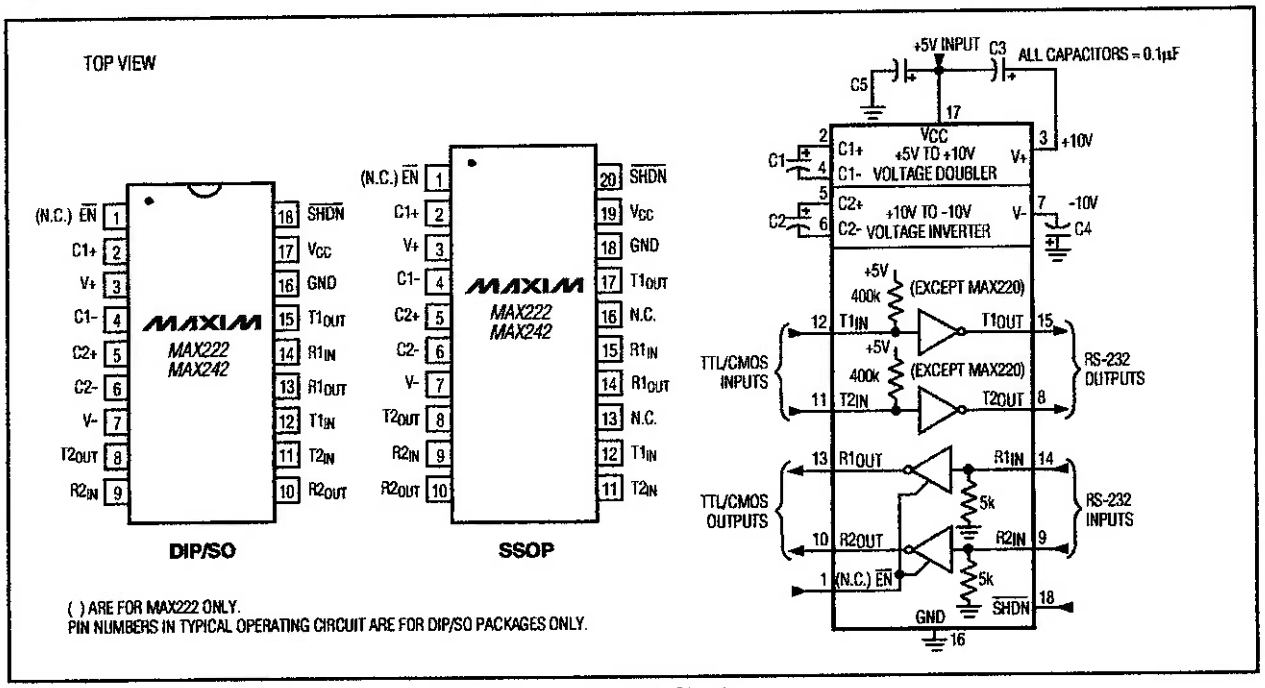
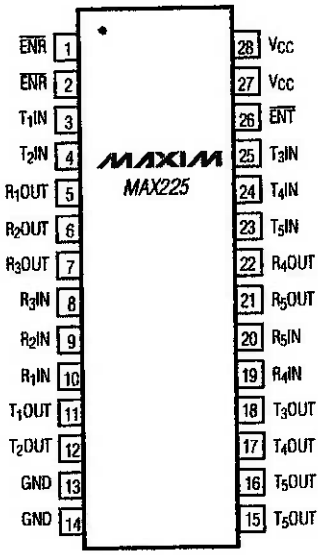


Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

TOP VIEW



SO

MAX225 FUNCTIONAL DESCRIPTION

5 RECEIVERS
5 TRANSMITTERS
2 CONTROL PINS
1 RECEIVER ENABLE ($\overline{ENR}$)
1 TRANSMITTER ENABLE ($\overline{ENT}$)

PINS ($\overline{ENR}$, GND, V_{CC}, T_5OUT) ARE INTERNALLY CONNECTED.
CONNECT EITHER OR BOTH EXTERNALLY. T_5OUT IS A SINGLE DRIVER.

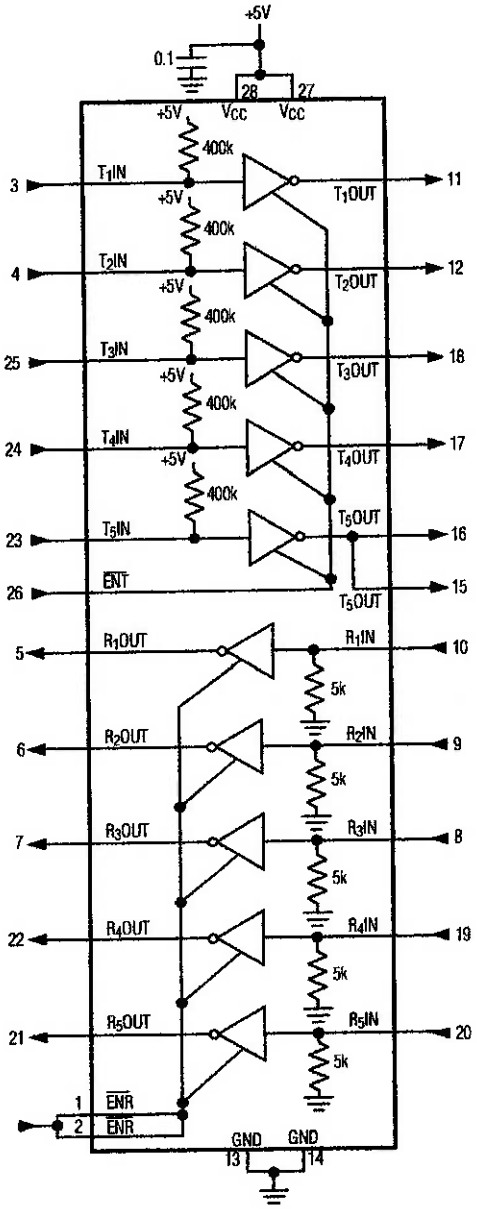


Figure 7. MAX225 Pin Configuration and Typical Operating Circuit

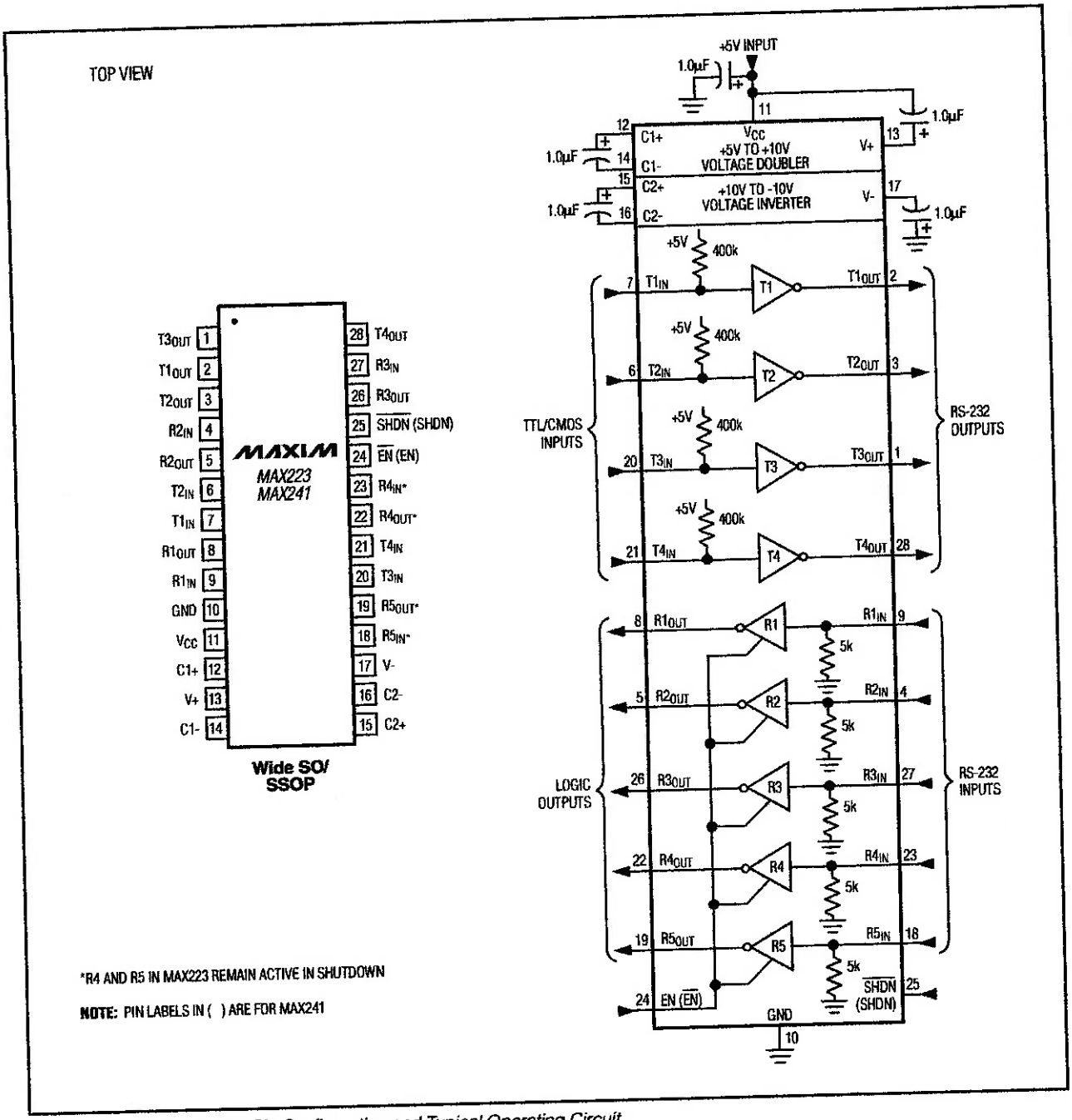


Figure 8. MAX223/MAX241 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

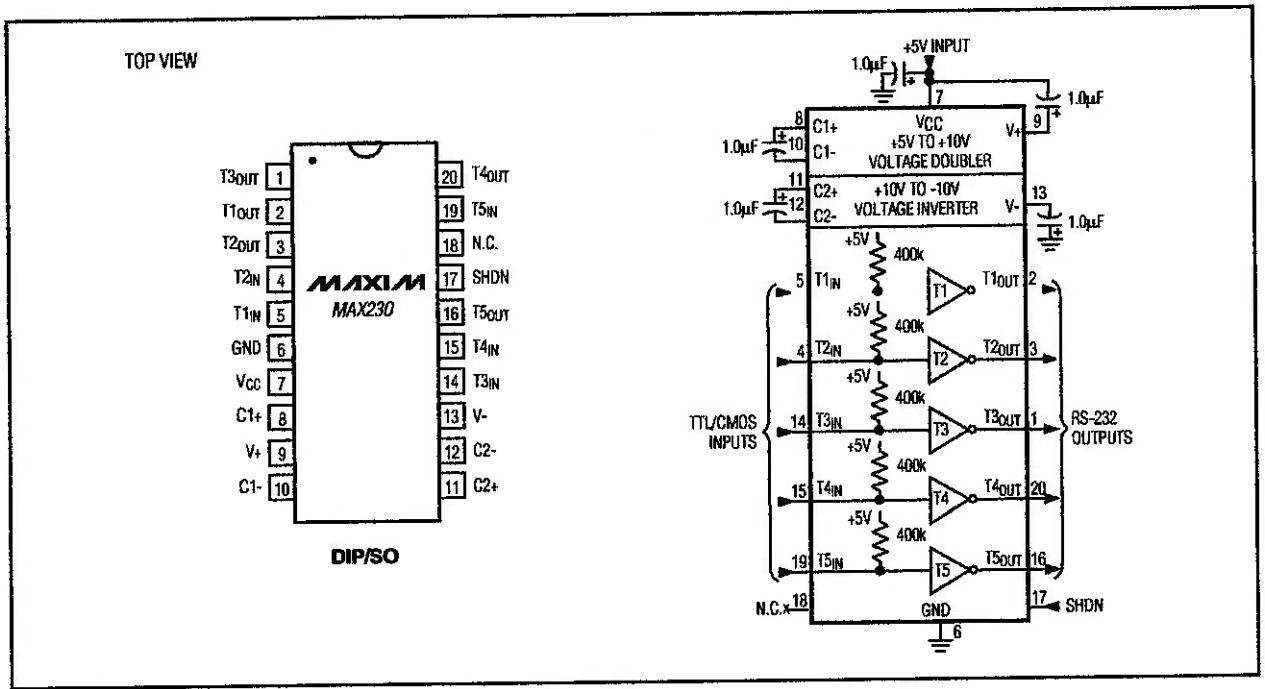


Figure 9. MAX230 Pin Configuration and Typical Operating Circuit

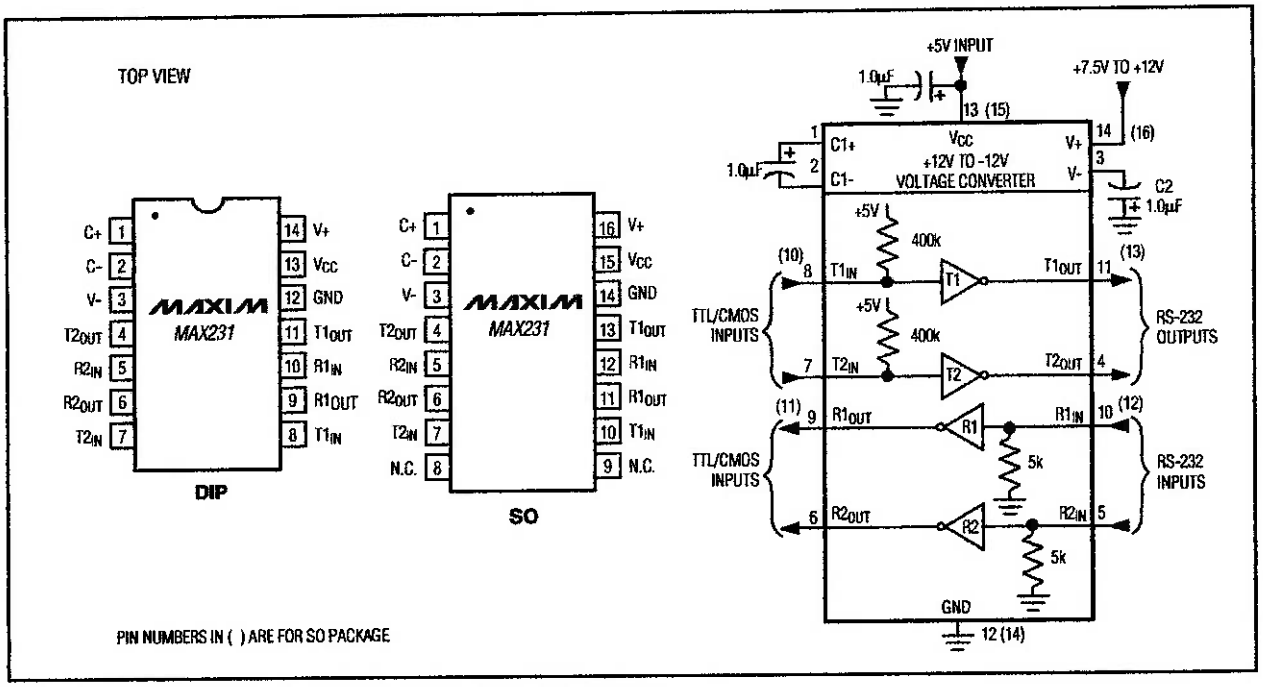


Figure 10. MAX231 Pin Configurations and Typical Operating Circuit

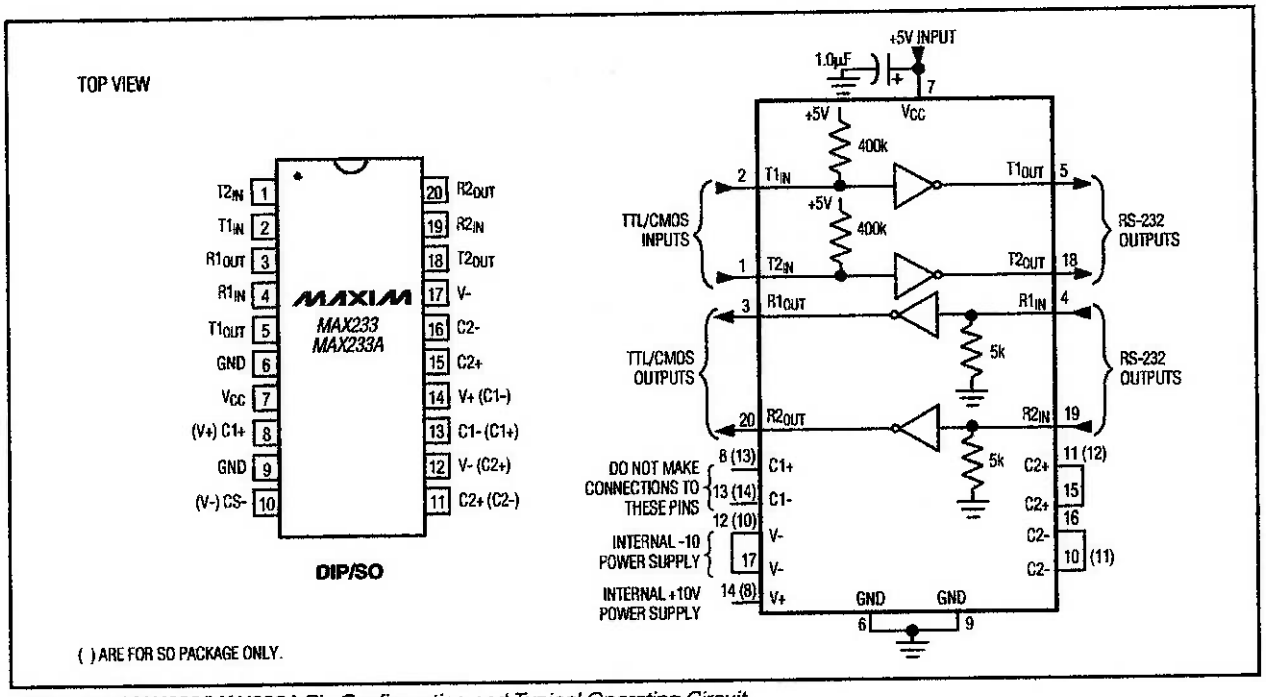


Figure 11. MAX233/MAX233A Pin Configuration and Typical Operating Circuit

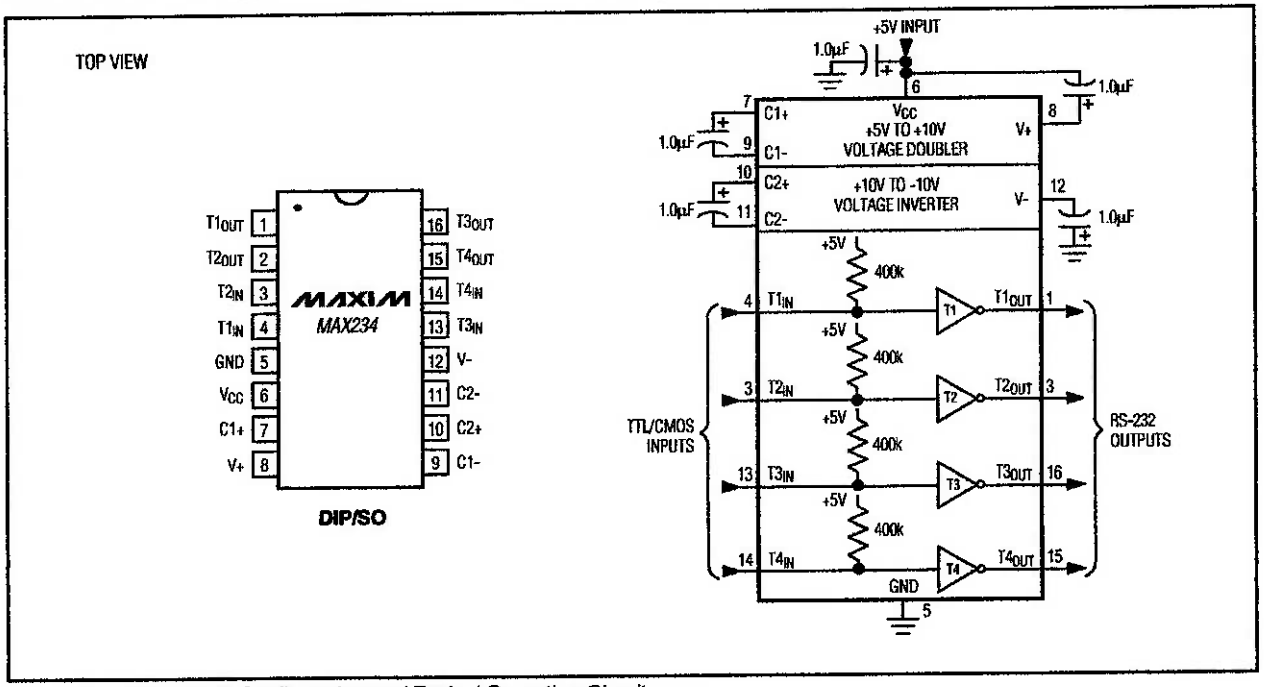


Figure 12. MAX234 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

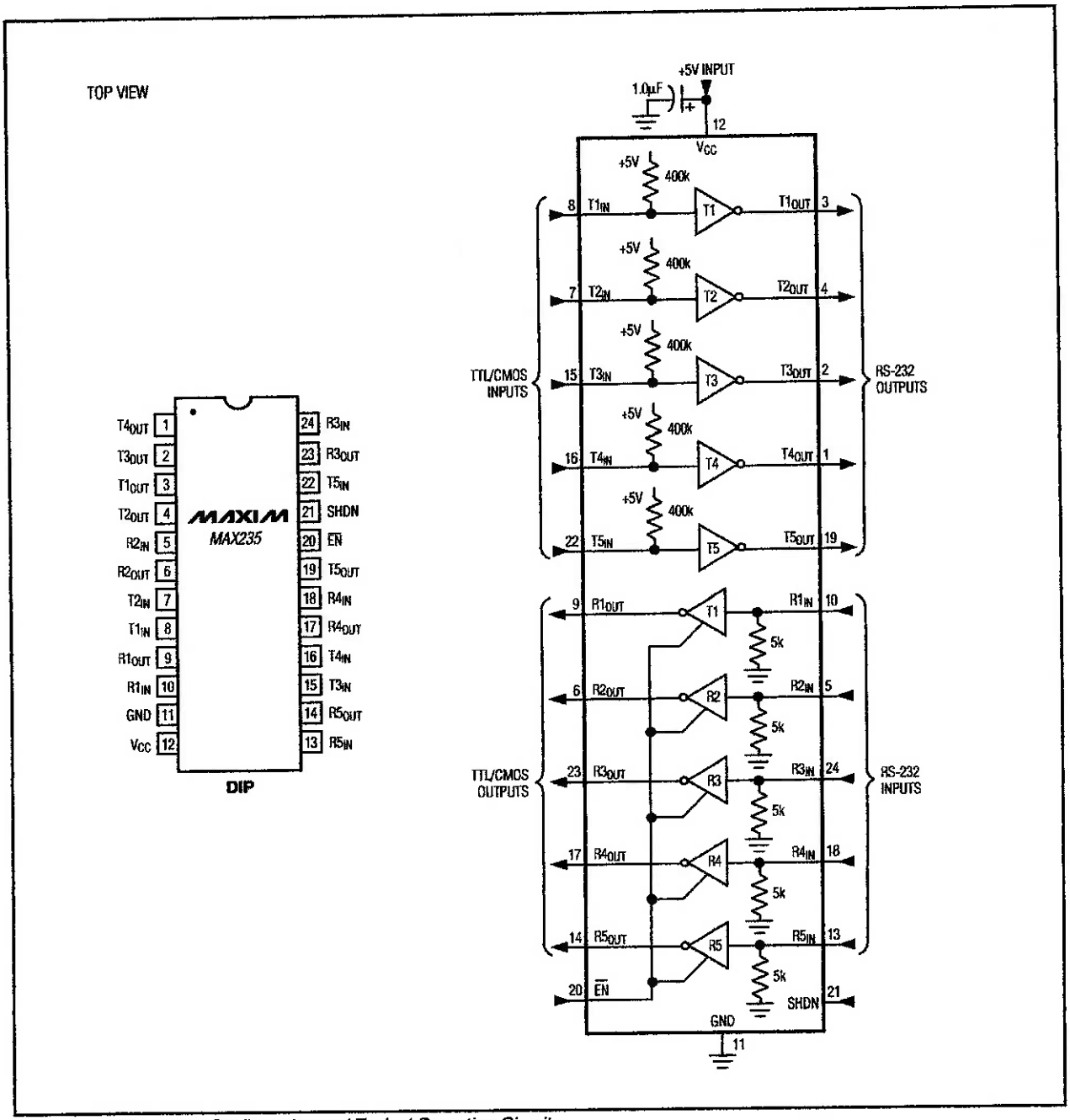


Figure 13. MAX235 Pin Configuration and Typical Operating Circuit

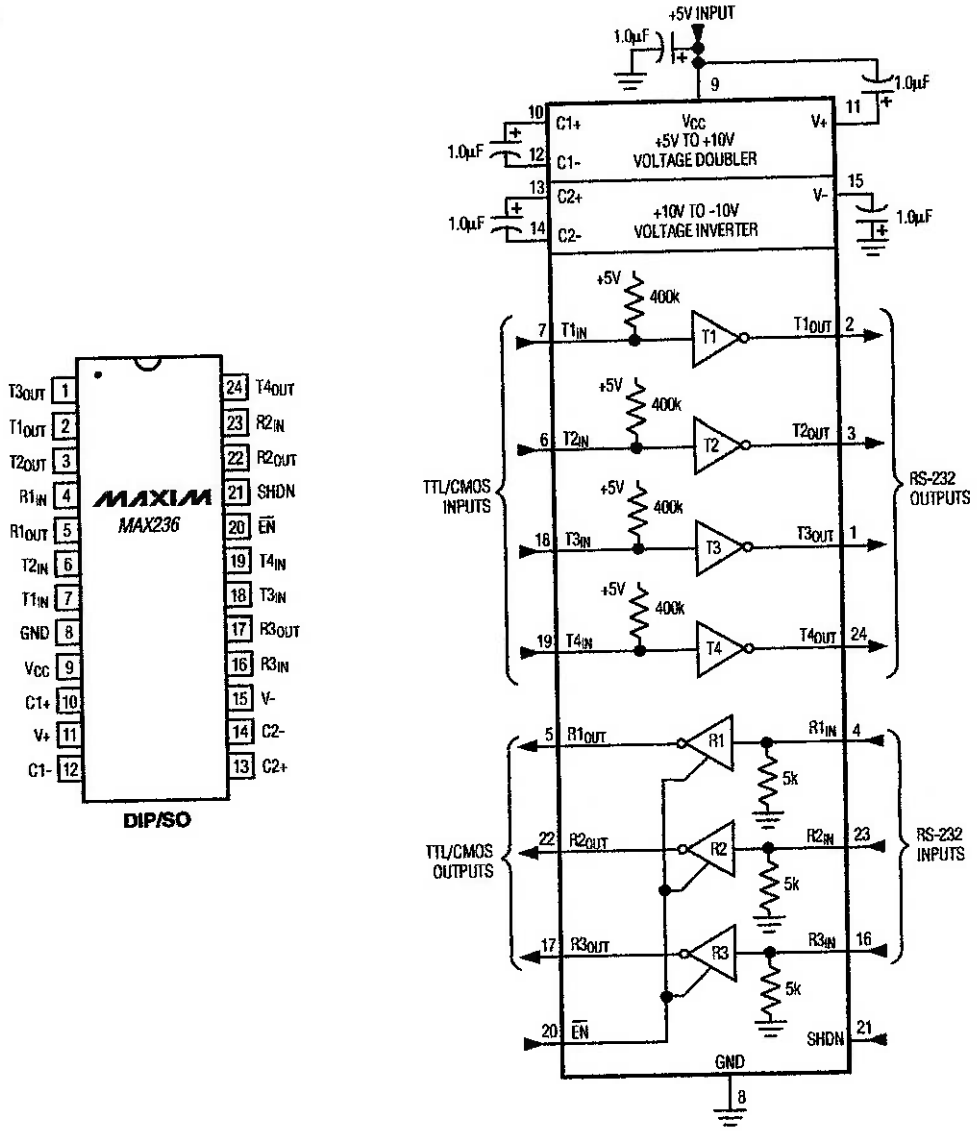


Figure 14. MAX236 Pin Configuration and Typical Operating Circuit

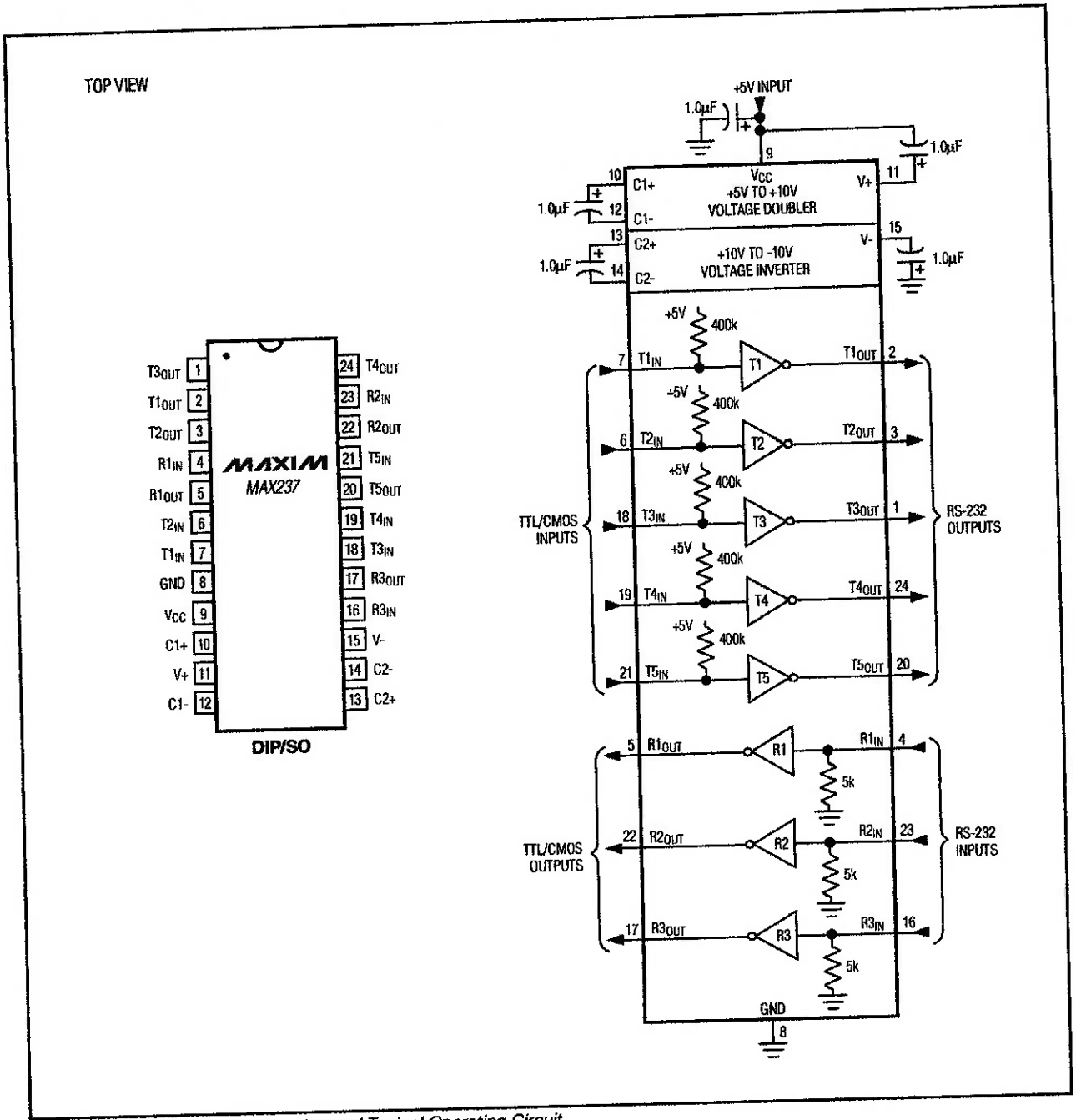


Figure 15. MAX237 Pin Configuration and Typical Operating Circuit

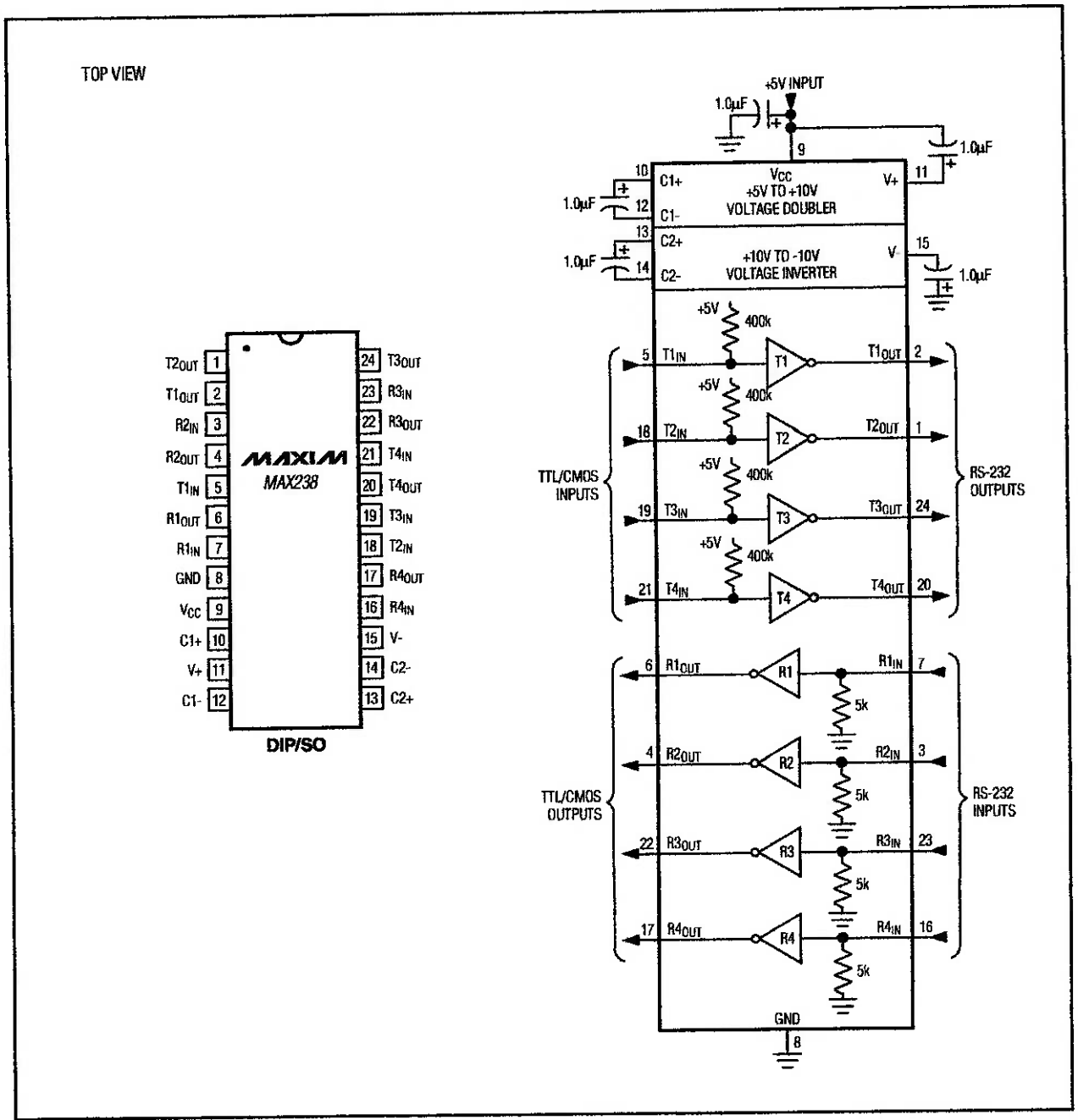


Figure 16. MAX238 Pin Configuration and Typical Operating Circuit

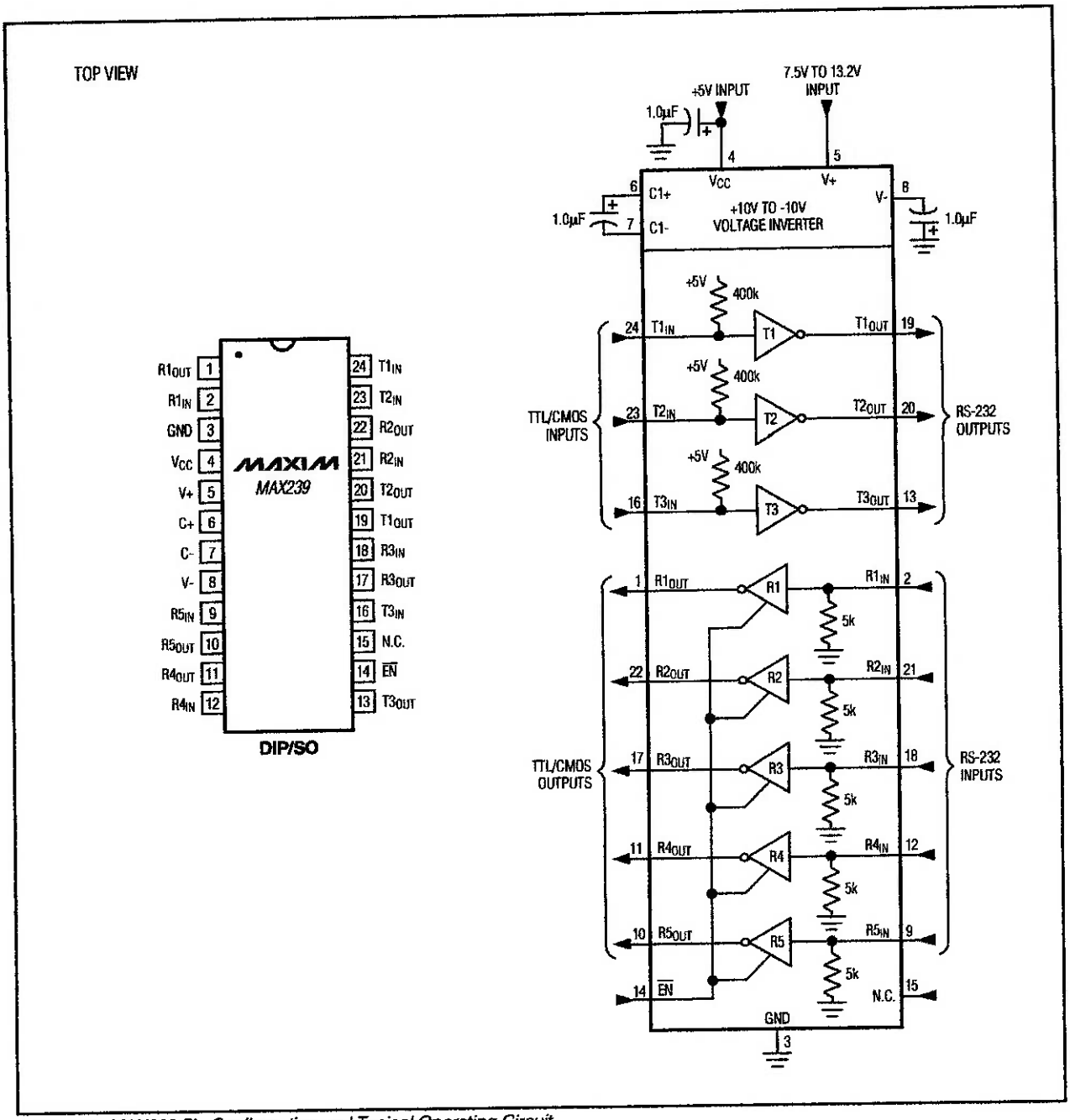


Figure 17. MAX239 Pin Configuration and Typical Operating Circuit

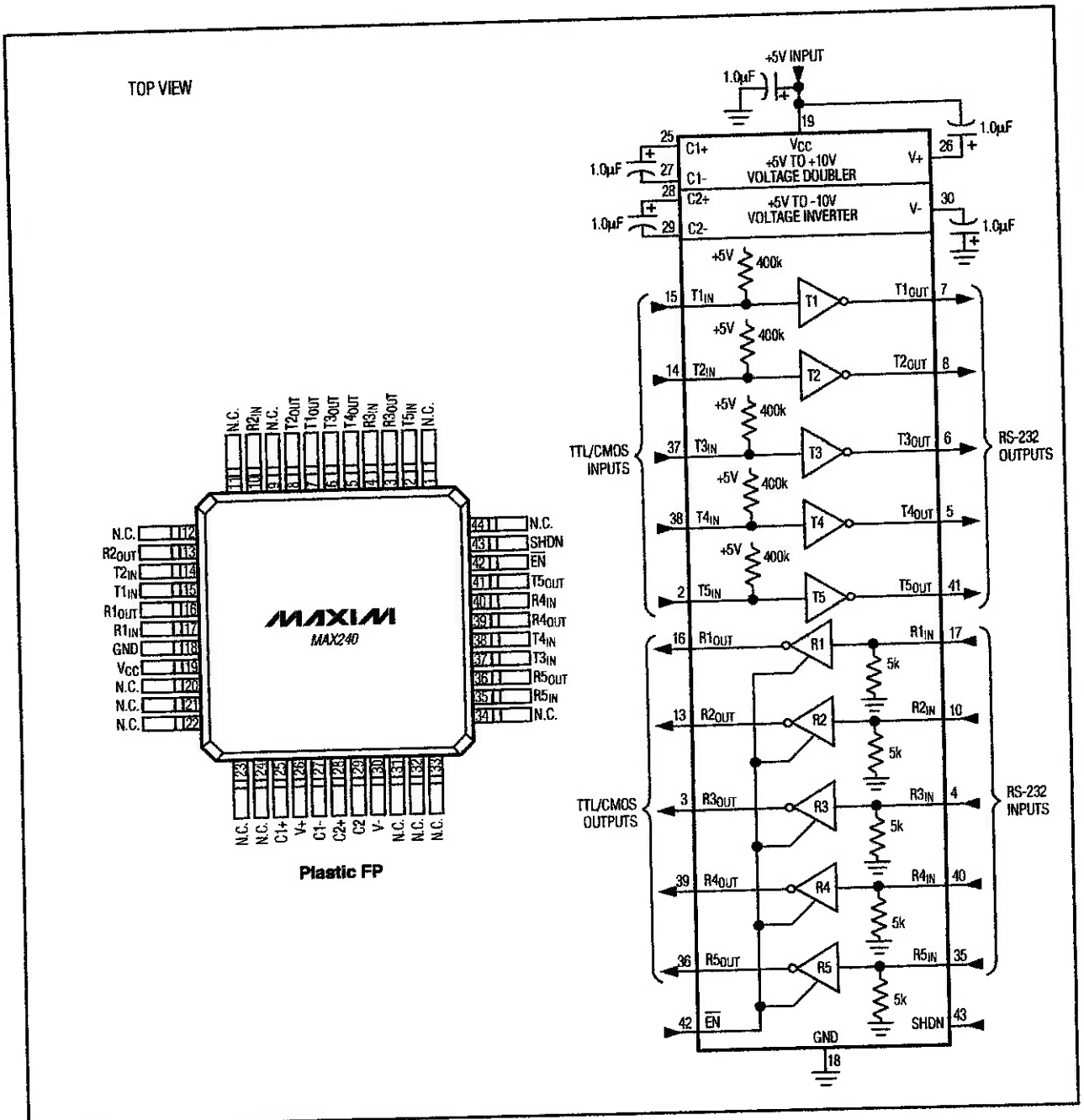
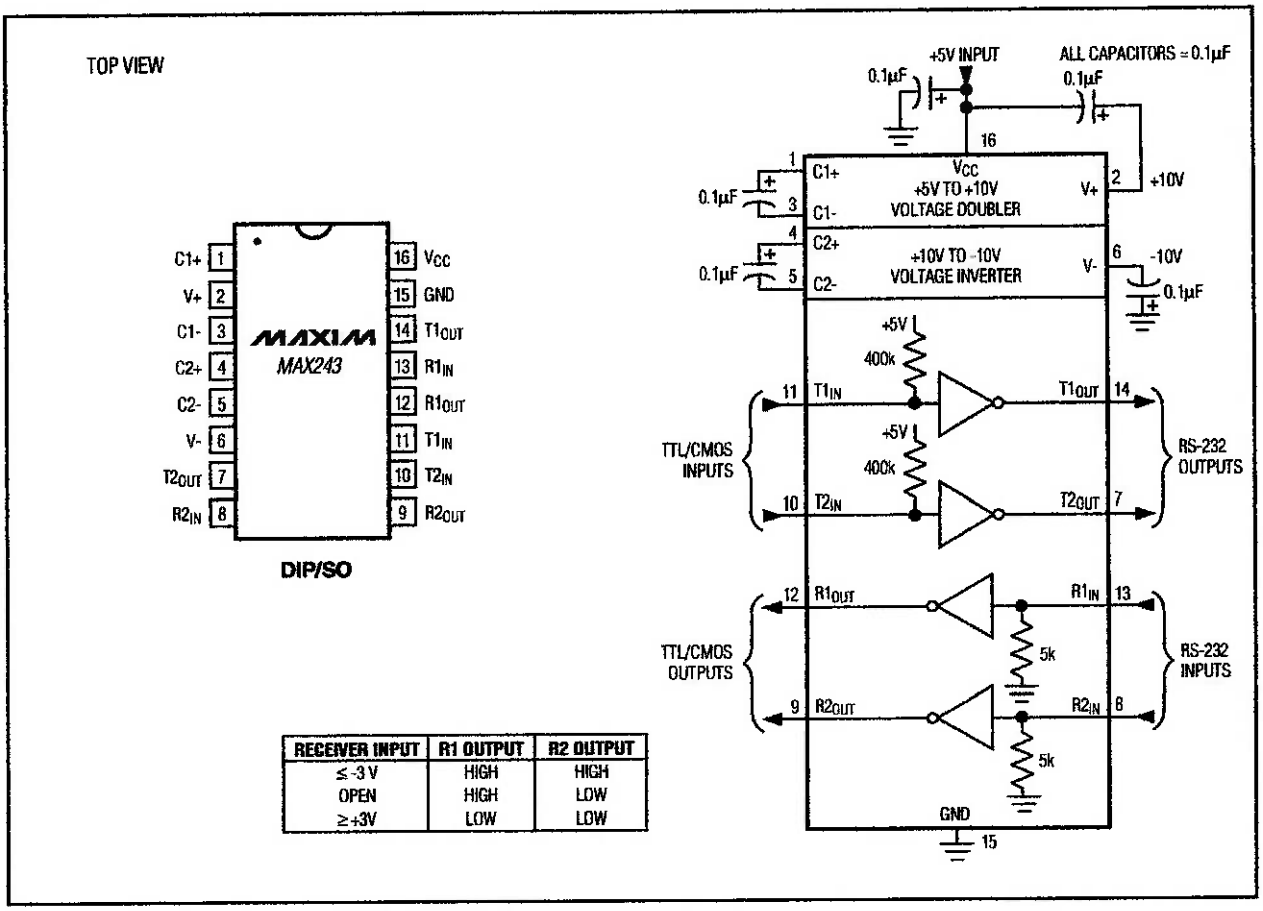
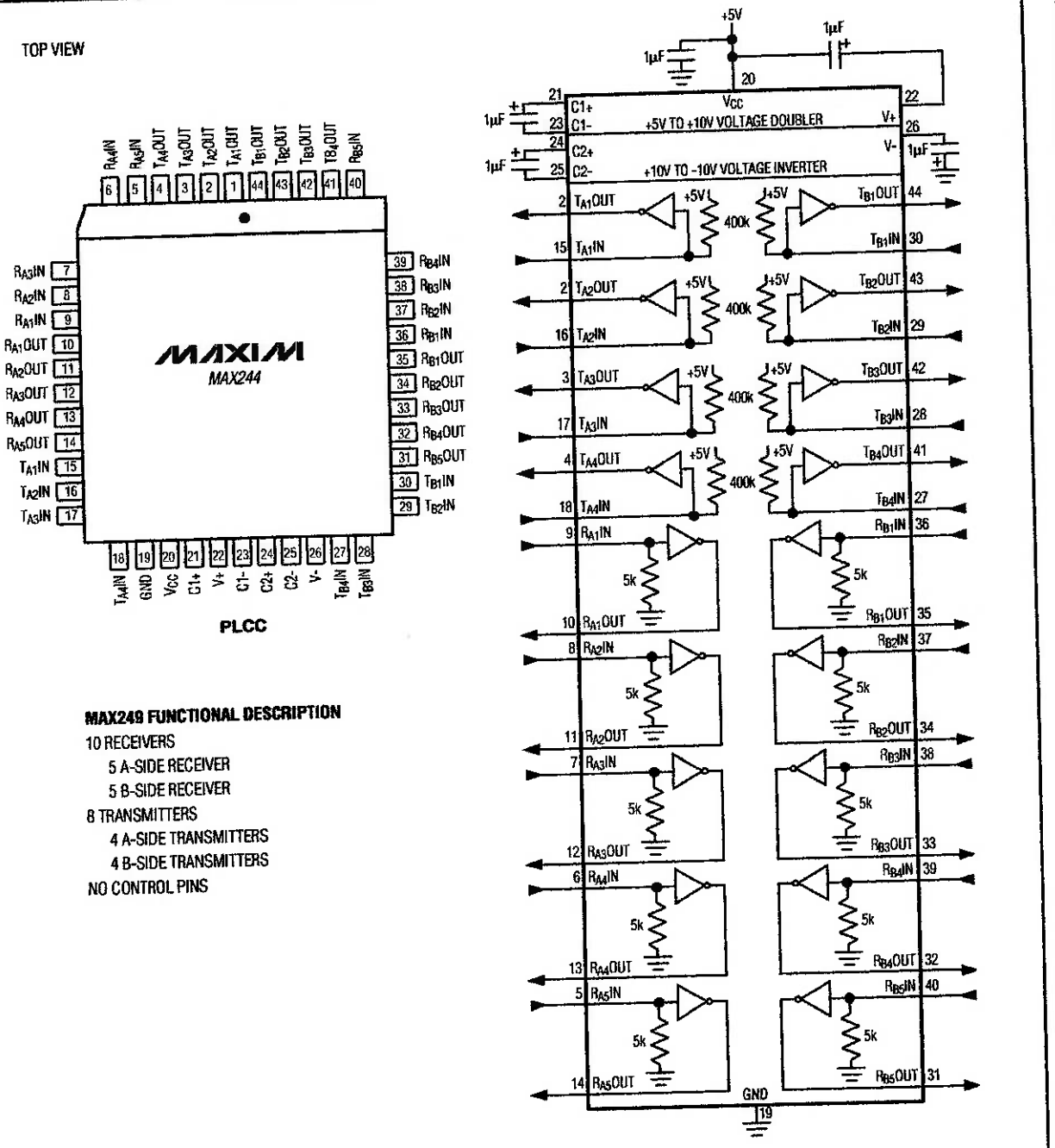


Figure 18. MAX240 Pin Configuration and Typical Operating Circuit



RECEIVER INPUT	R1 OUTPUT	R2 OUTPUT
≤ -3 V	HIGH	HIGH
OPEN	HIGH	LOW
≥ +3V	LOW	LOW

Figure 19. MAX243 Pin Configuration and Typical Operating Circuit



MAX249 FUNCTIONAL DESCRIPTION

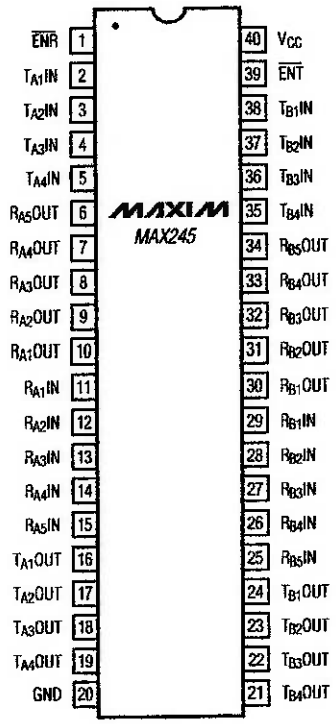
10 RECEIVERS
- 5 A-SIDE RECEIVER
- 5 B-SIDE RECEIVER

8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

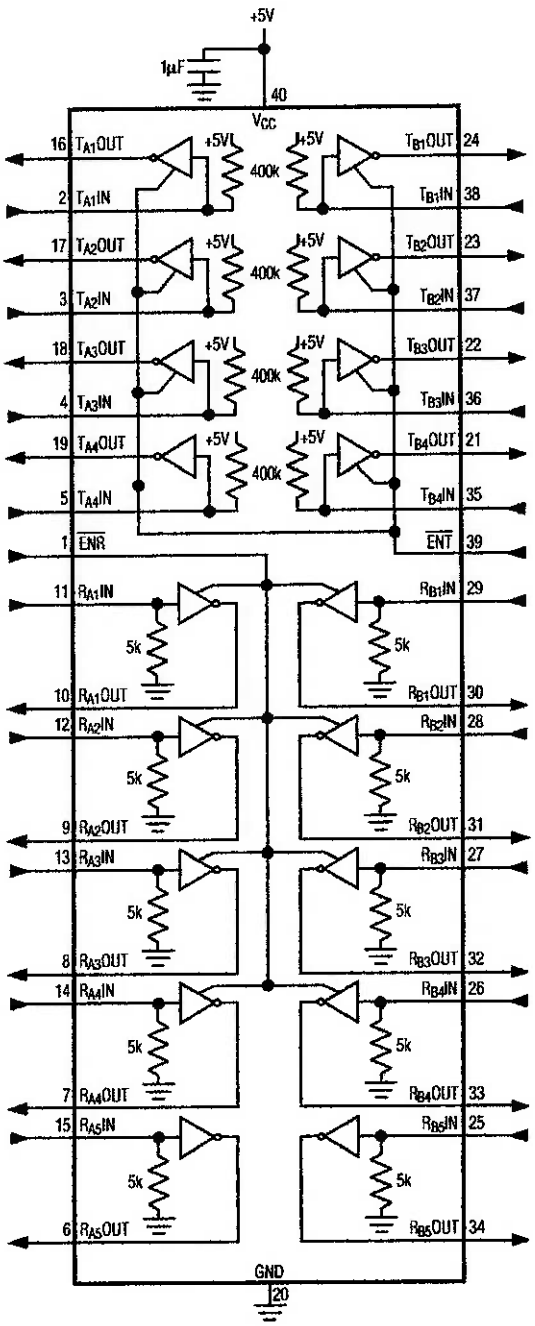
NO CONTROL PINS

Figure 20. MAX244 Pin Configuration and Typical Operating Circuit

TOP VIEW



DIP



MAX245 FUNCTIONAL DESCRIPTION

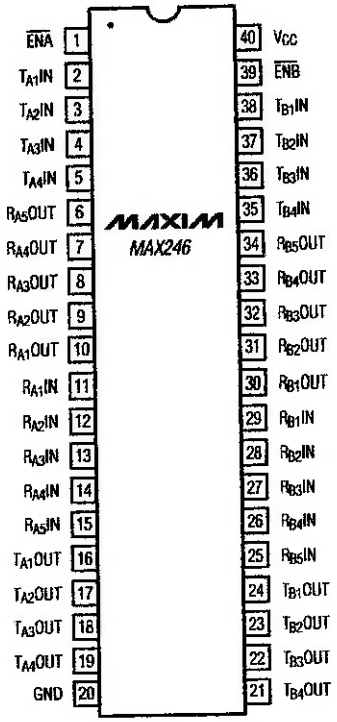
10 RECEIVERS
- 5 A-SIDE RECEIVERS (R_{A5} ALWAYS ACTIVE)
- 5 B-SIDE RECEIVERS (R_{B5} ALWAYS ACTIVE)

8 TRANSMITTTERS
- 4 A-SIDE TRANSMITTERS

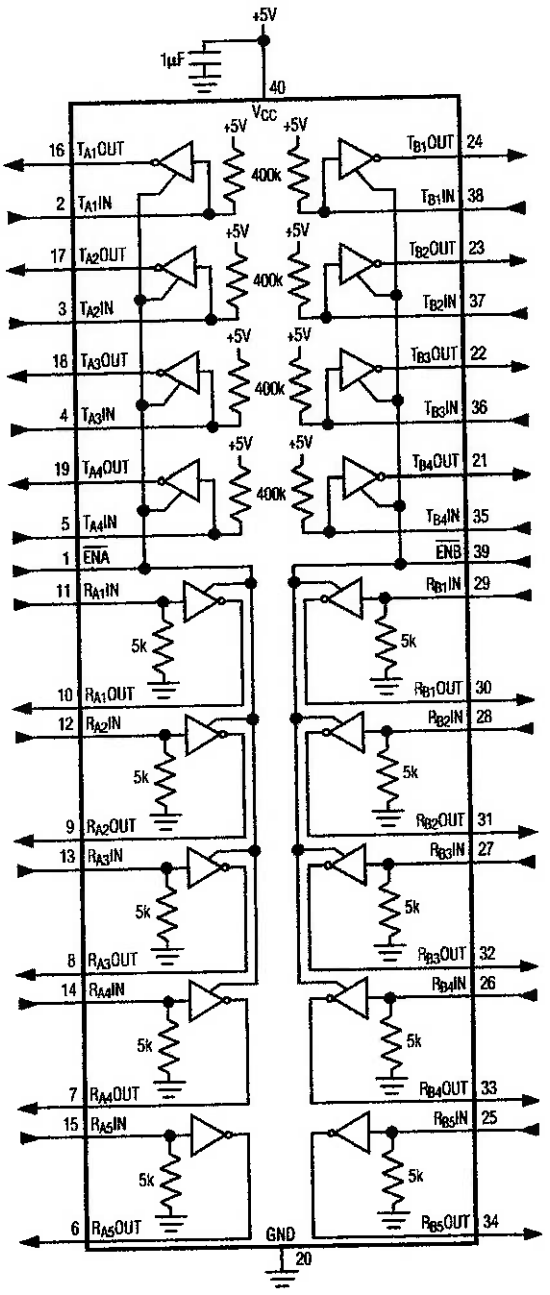
2 CONTROL PINS
- 1 RECEIVER ENABLE ($\overline{ENR}$)
- 1 TRANSMITTER ENABLE ($\overline{ENT}$)

Figure 21. MAX245 Pin Configuration and Typical Operating Circuit

TOP VIEW



DIP



MAX246 FUNCTIONAL DESCRIPTION

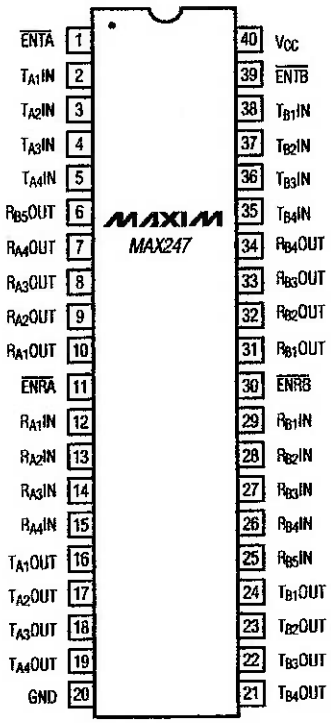
10 RECEIVERS
- 5 A-SIDE RECEIVERS (R_{A5} ALWAYS ACTIVE)
- 5 B-SIDE RECEIVERS (R_{B5} ALWAYS ACTIVE)

8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

2 CONTROL PINS
- ENABLE A-SIDE ($\overline{ENA}$)
- ENABLE B-SIDE ($\overline{ENB}$)

Figure 22. MAX246 Pin Configuration and Typical Operating Circuit

TOP VIEW



DIP

MAX247 FUNCTIONAL DESCRIPTION

9 RECEIVERS
- 4 A-SIDE RECEIVERS
- 5 B-SIDE RECEIVERS (R_{B5} ALWAYS ACTIVE)

8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

4 CONTROL PINS
- ENABLE RECEIVER A-SIDE ($\overline{ENRA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENRB}$)
- ENABLE RECEIVER A-SIDE ($\overline{ENTA}$)
- ENABLE RECEIVERr B-SIDE ($\overline{ENTB}$)

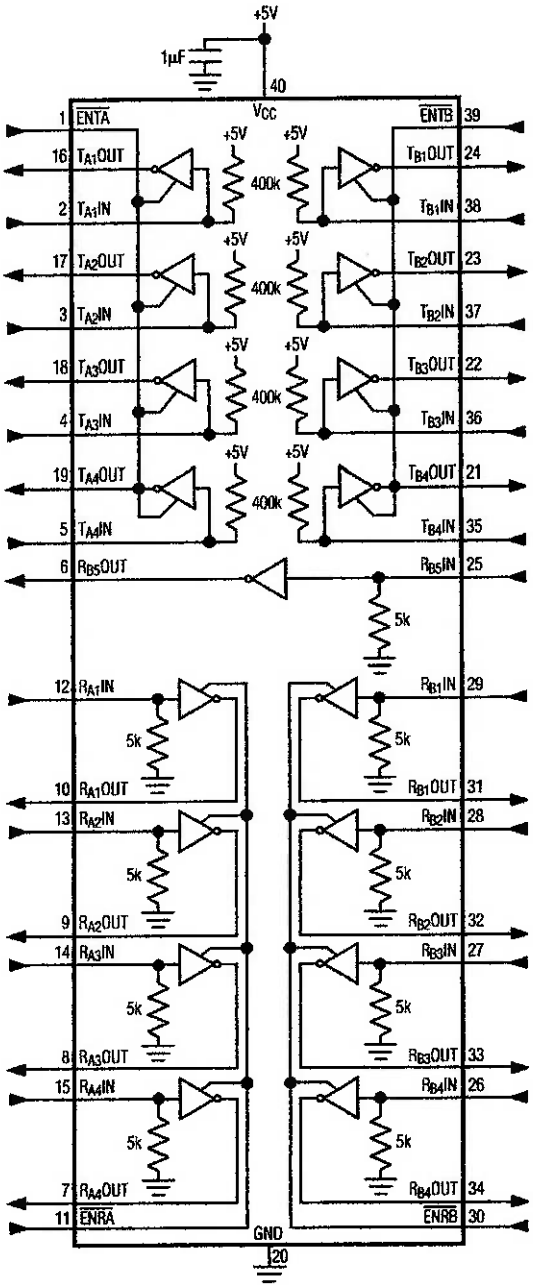
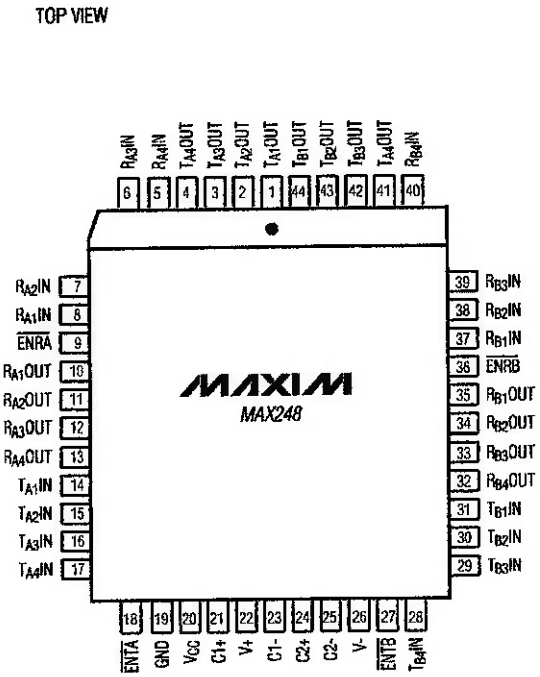
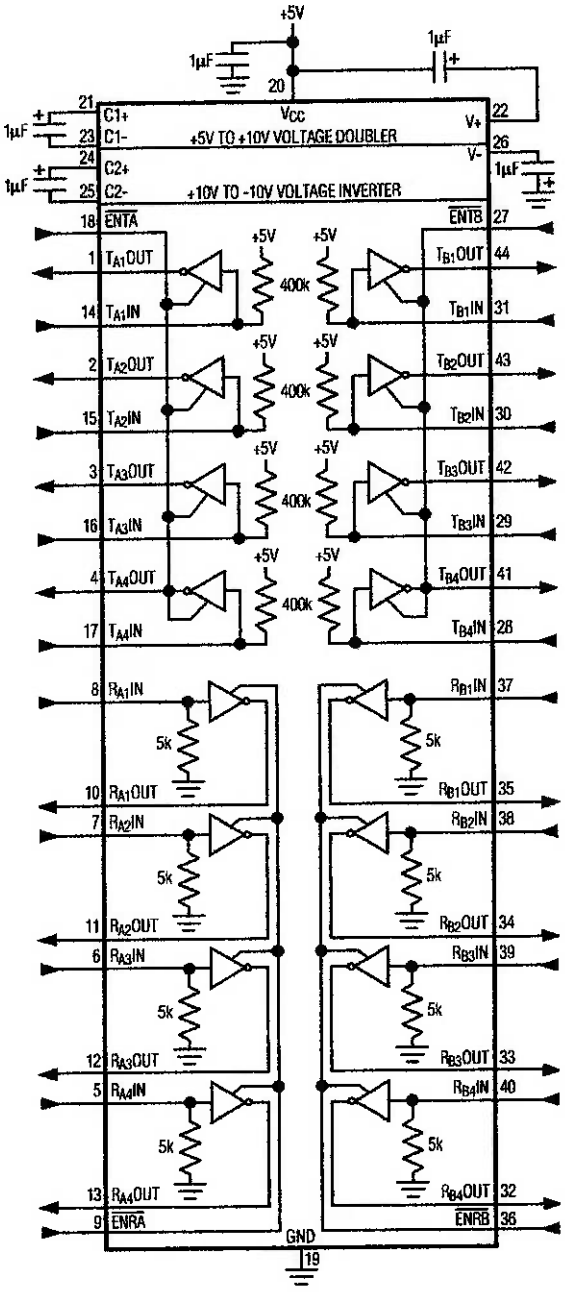


Figure 23. MAX247 Pin Configuration and Typical Operating Circuit





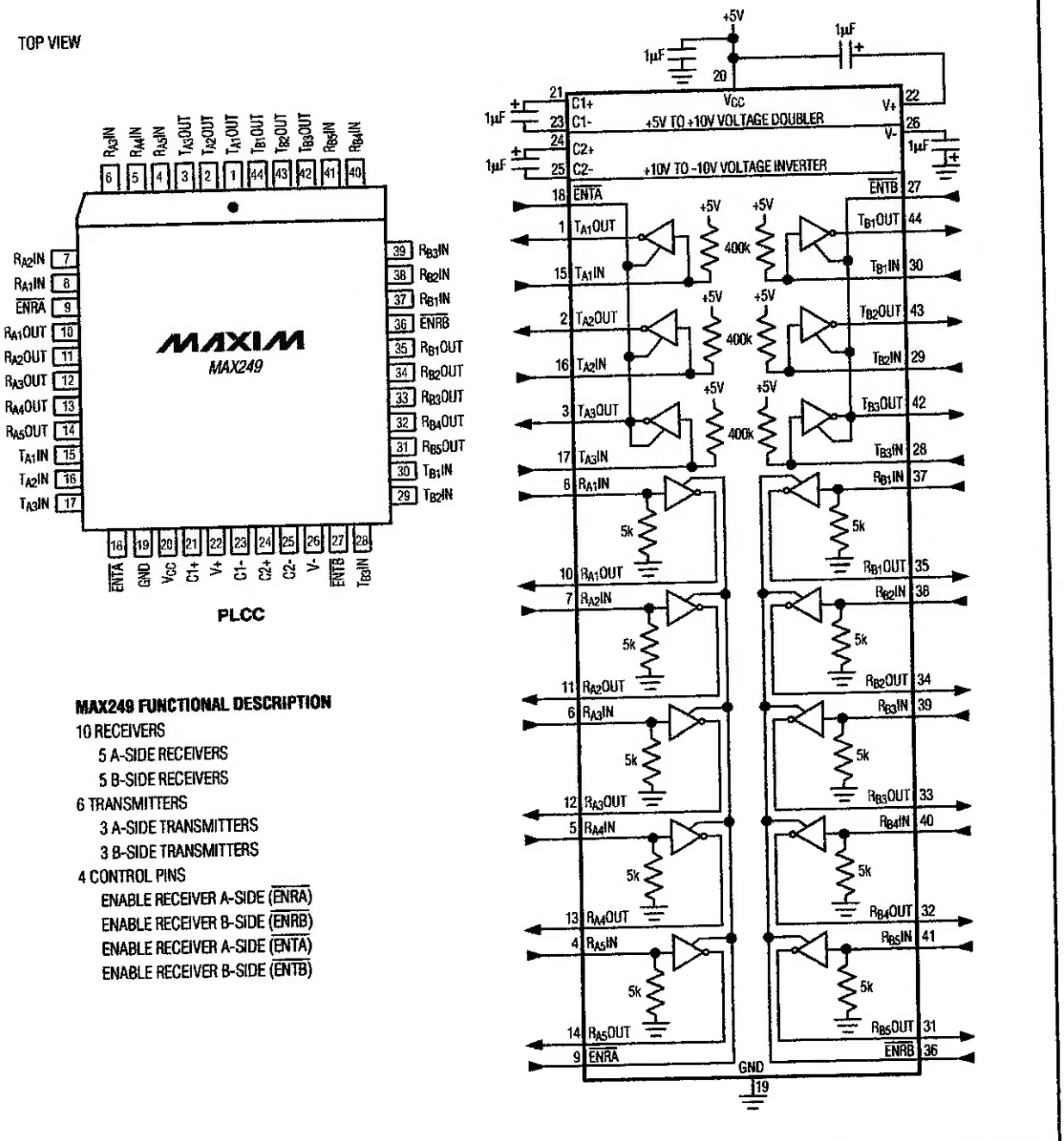
MAX248 FUNCTIONAL DESCRIPTION

8 RECEIVERS
- 4 A-SIDE RECEIVERS
- 4 B-SIDE RECEIVERS

8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

4 CONTROL PINS
- ENABLE RECEIVER A-SIDE ($\overline{ENRA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENRB}$)
- ENABLE RECEIVER A-SIDE ($\overline{ENTA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENTB}$)

Figure 24. MAX248 Pin Configuration and Typical Operating Circuit



MAX249 FUNCTIONAL DESCRIPTION

10 RECEIVERS
- 5 A-SIDE RECEIVERS
- 5 B-SIDE RECEIVERS

6 TRANSMITTERS
- 3 A-SIDE TRANSMITTERS
- 3 B-SIDE TRANSMITTERS

4 CONTROL PINS
- ENABLE RECEIVER A-SIDE ($\overline{ENRA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENRB}$)
- ENABLE RECEIVER A-SIDE ($\overline{ENTA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENTB}$)

Figure 25. MAX249 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

Ordering Information (continued)

PART	TEMP. RANGE	PIN-PACKAGE
MAX222CPN	0°C to +70°C	18 Plastic DIP
MAX222CWN	0°C to +70°C	18 Wide SO
MAX222C/D	0°C to +70°C	Dice*
MAX222EPN	-40°C to +85°C	18 Plastic DIP
MAX222EWN	-40°C to +85°C	18 Wide SO
MAX222EJN	-40°C to +85°C	18 CERDIP
MAX222MJN	-55°C to +125°C	18 CERDIP
MAX223CAI	0°C to +70°C	28 SSOP
MAX223CWI	0°C to +70°C	28 Wide SO
MAX223C/D	0°C to +70°C	Dice*
MAX223EAI	-40°C to +85°C	28 SSOP
MAX223EWI	-40°C to +85°C	28 Wide SO
MAX225CWI	0°C to +70°C	28 Wide SO
MAX225EWI	-40°C to +85°C	28 Wide SO
MAX230CPP	0°C to +70°C	20 Plastic DIP
MAX230CWP	0°C to +70°C	20 Wide SO
MAX230C/D	0°C to +70°C	Dice*
MAX230EPP	-40°C to +85°C	20 Plastic DIP
MAX230EWP	-40°C to +85°C	20 Wide SO
MAX230EJP	-40°C to +85°C	20 CERDIP
MAX230MJP	-55°C to +125°C	20 CERDIP
MAX231CPD	0°C to +70°C	14 Plastic DIP
MAX231CWE	0°C to +70°C	16 Wide SO
MAX231CJD	0°C to +70°C	14 CERDIP
MAX231C/D	0°C to +70°C	Dice*
MAX231EPD	-40°C to +85°C	14 Plastic DIP
MAX231EWE	-40°C to +85°C	16 Wide SO
MAX231EJD	-40°C to +85°C	14 CERDIP
MAX231MJD	-55°C to +125°C	14 CERDIP
MAX232CPE	0°C to +70°C	16 Plastic DIP
MAX232CSE	0°C to +70°C	16 Narrow SO
MAX232CWE	0°C to +70°C	16 Wide SO
MAX232C/D	0°C to +70°C	Dice*
MAX232EPE	-40°C to +85°C	16 Plastic DIP
MAX232ESE	-40°C to +85°C	16 Narrow SO
MAX232EWE	-40°C to +85°C	16 Wide SO
MAX232EJE	-40°C to +85°C	16 CERDIP
MAX232MJE	-55°C to +125°C	16 CERDIP
MAX232MLP	-55°C to +125°C	20 LCC
MAX232ACPE	0°C to +70°C	16 Plastic DIP
MAX232ACSE	0°C to +70°C	16 Narrow SO
MAX232ACWE	0°C to +70°C	16 Wide SO
MAX232AC/D	0°C to +70°C	Dice*
MAX232AEPE	-40°C to +85°C	16 Plastic DIP
MAX232AESE	-40°C to +85°C	16 Narrow SO
MAX232AEWE	-40°C to +85°C	16 Wide SO
MAX232AEJE	-40°C to +85°C	16 CERDIP
MAX232AMJE	-55°C to +125°C	16 CERDIP
MAX232AMLP	-55°C to +125°C	20 LCC
MAX233CPP	0°C to +70°C	20 Plastic DIP
MAX233EPP	-40°C to +85°C	20 Plastic DIP
MAX233ACPP	0°C to +70°C	20 Plastic DIP
MAX233ACWP	0°C to +70°C	20 Wide SO
MAX233AEPP	-40°C to +85°C	20 Plastic DIP
MAX233AEWP	-40°C to +85°C	20 Wide SO
MAX234CPE	0°C to +70°C	16 Plastic DIP
MAX234CWE	0°C to +70°C	16 Wide SO
MAX234C/D	0°C to +70°C	Dice*
MAX234EPE	-40°C to +85°C	16 Plastic DIP
MAX234EWE	-40°C to +85°C	16 Wide SO
MAX234EJE	-40°C to +85°C	16 CERDIP
MAX234MJE	-55°C to +125°C	16 CERDIP
MAX235CPG	0°C to +70°C	24 Wide Plastic DIP
MAX235EPG	-40°C to +85°C	24 Wide Plastic DIP
MAX235EDG	-40°C to +85°C	24 Ceramic SB
MAX235MDG	-55°C to +125°C	24 Ceramic SB
MAX236CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX236CWG	0°C to +70°C	24 Wide SO
MAX236C/D	0°C to +70°C	Dice*
MAX236ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX236EWG	-40°C to +85°C	24 Wide SO
MAX236ERG	-40°C to +85°C	24 Narrow CERDIP
MAX236MRG	-55°C to +125°C	24 Narrow CERDIP
MAX237CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX237CWG	0°C to +70°C	24 Wide SO
MAX237C/D	0°C to +70°C	Dice*
MAX237ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX237EWG	-40°C to +85°C	24 Wide SO
MAX237ERG	-40°C to +85°C	24 Narrow CERDIP
MAX237MRG	-55°C to +125°C	24 Narrow CERDIP
MAX238CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX238CWG	0°C to +70°C	24 Wide SO
MAX238C/D	0°C to +70°C	Dice*
MAX238ENG	-40°C to +85°C	24 Narrow Plastic DIP

** Contact factory for dice specifications.*

Ordering Information (continued)

PART	TEMP. RANGE	PIN-PACKAGE
MAX238EWG	-40°C to +85°C	24 Wide SO
MAX238ERG	-40°C to +85°C	24 Narrow CERDIP
MAX238MRG	-55°C to +125°C	24 Narrow CERDIP
MAX239CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX239CWG	0°C to +70°C	24 Wide SO
MAX239C/D	0°C to +70°C	Dice*
MAX239ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX239EWG	-40°C to +85°C	24 Wide SO
MAX239ERG	-40°C to +85°C	24 Narrow CERDIP
MAX239MRG	-55°C to +125°C	24 Narrow CERDIP
MAX240CMH	0°C to +70°C	44 Plastic FP
MAX240C/D	0°C to +70°C	Dice*
MAX241CAI	0°C to +70°C	28 SSOP
MAX241CWI	0°C to +70°C	28 Wide SO
MAX241C/D	0°C to +70°C	Dice*
MAX241EAI	-40°C to +85°C	28 SSOP
MAX241EWI	-40°C to +85°C	28 Wide SO
MAX242CAP	0°C to +70°C	20 SSOP
MAX242CPN	0°C to +70°C	18 Plastic DIP
MAX242CWN	0°C to +70°C	18 Wide SO
MAX242C/D	0°C to +70°C	Dice*
MAX242EPN	-40°C to +85°C	18 Plastic DIP
MAX242EWN	-40°C to +85°C	18 Wide SO
MAX242EJN	-40°C to +85°C	18 CERDIP
MAX242MJN	-55°C to +125°C	18 CERDIP
MAX243CPE	0°C to +70°C	16 Plastic DIP
MAX243CSE	0°C to +70°C	16 Narrow SO
MAX243CWE	0°C to +70°C	16 Wide SO
MAX243C/D	0°C to +70°C	Dice*
MAX243EPE	-40°C to +85°C	16 Plastic DIP
MAX243ESE	-40°C to +85°C	16 Narrow SO
MAX243EWE	-40°C to +85°C	16 Wide SO
MAX243EJE	-40°C to +85°C	16 CERDIP
MAX243MJE	-55°C to +125°C	16 CERDIP
MAX244CQH	0°C to +70°C	44 PLCC
MAX244C/D	0°C to +70°C	Dice*
MAX244EQH	-40°C to +85°C	44 PLCC
MAX245CPL	0°C to +70°C	40 Plastic DIP
MAX245C/D	0°C to +70°C	Dice*
MAX245EPL	-40°C to +85°C	40 Plastic DIP
MAX246CPL	0°C to +70°C	40 Plastic DIP
MAX246C/D	0°C to +70°C	Dice*
MAX246EPL	-40°C to +85°C	40 Plastic DIP
MAX247CPL	0°C to +70°C	40 Plastic DIP
MAX247C/D	0°C to +70°C	Dice*
MAX247EPL	-40°C to +85°C	40 Plastic DIP
MAX248CQH	0°C to +70°C	44 PLCC
MAX248C/D	0°C to +70°C	Dice*
MAX248EQH	-40°C to +85°C	44 PLCC
MAX249CQH	0°C to +70°C	44 PLCC
MAX249EQH	-40°C to +85°C	44 PLCC

** Contact factory for dice specifications.*

ANEXO III –

Especificações do transceiver ZHX1810

ZHX1810

Slim Series SIR Transceiver

Product Specification

PS009309-0901

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Document Disclaimer

PS009309-0901

Table of Contents

List of Figures

List of Tables

Description

The ZILOG ZHX1810 is a low-profile version of ZiLOG's popular ZHX1010 1-meter transceiver. The transceiver is mechanically enhanced for ultra compact, power-conscious portable products, such as mobile phones, portable printers, handheld computers, and personal data assistants (PDAs). Designed to operate using the IrDA-Data mode, the transceiver combines an infrared emitting diode (IRED) emitter, a PIN photodiode detector, a digital AC coupled LED driver, and a receiver/decoder in a single package.

The ZILOG ZHX1810 provides an efficient implementation of the SIR standard in a small-outline footprint format. Application circuit space is also minimized, as only three components are required.

ZHX1810 also features an independently controlled shutdown that minimizes current draw to a maximum of 1 µA.

Features

- Compliant to IrDA Data Specification SIR
- Wide power supply voltage range, 2.4 to 5.5 V
- Minimum link distance, 1 M
- Low-power, listening current, 90 µA (typical) at 3.0 V
- Slim form factor (9.1 mm long x 3.8 mm wide x 2.73 mm high)
- Only three external components required
- Extended operating temperature range (–30 °C to +85 °C)
- Meets IEC 825-1 Class 1 Eye Safety Specifications

Block Diagram

Figure 1 is the block diagram for the Slim SIR transceiver.

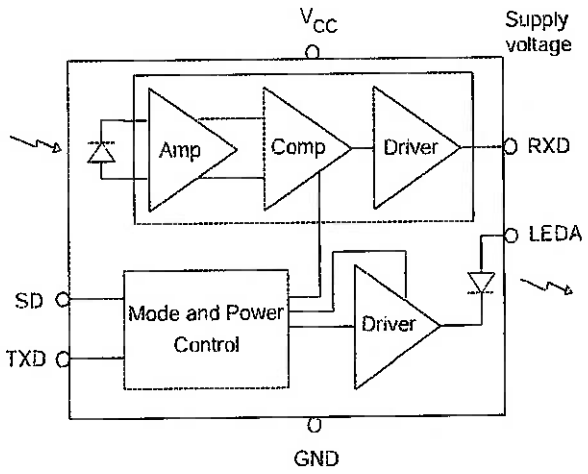


Figure 1. Slim SIR Transceiver Block Diagram

Pin Descriptions

The ZHX1810 transceiver uses the pins listed in Table 1. The pins are described in this section.

Table 1. Pin Out for the ZHX1810 Transceiver

Pin	Name	Function	I/O
1	LEDA	IRED anode	—
2	TXD	Transmitter input	I
3	RXD	Receiver output	O
4	SD	Enables shutdown mode	I
5	V_{CC}	Supply voltage	—
6	GND	Ground	—
—	TAB	Shield ground	—

LEDA LED Driver Anode

(Power)

This output is connected to the LED anode. Current to the LED is sourced through an external resistor.

TXD Transmit Data

(Input, active high)

This CMOS input is used to transmit serial data. This input has an internal pull-down resistor that is disabled (open-circuited) during shutdown.

RXD/Receive Data

(Output, active low)

This output indicates received serial data. It is a tri-state, slew rate controlled CMOS output (tri-stated during shutdown) driver capable of driving a standard CMOS load. No external resistor is required.

SD Shutdown

(Input, active high)

This input is used to place the integrated circuit into shutdown mode. Module shut-down current is influenced by the choice of capacitor used from V_{CC} to ground.

V_{CC} Positive Supply

(Power)

Connect to positive power supply (2.4–5.5 V). Filter with a 0.33-µF ceramic bypass capacitor and terminating resistor as close as possible to the V_{CC} pin.

GND Ground

(Power)

Connect to ground of the power supply. A solid ground plane is recommended for proper operation.

TAB

(Shield)

The Shield tab must be soldered to the ground plane.

Recommended Application Circuits

Figure 2 shows application block diagrams for the ZHX1810 transceiver.

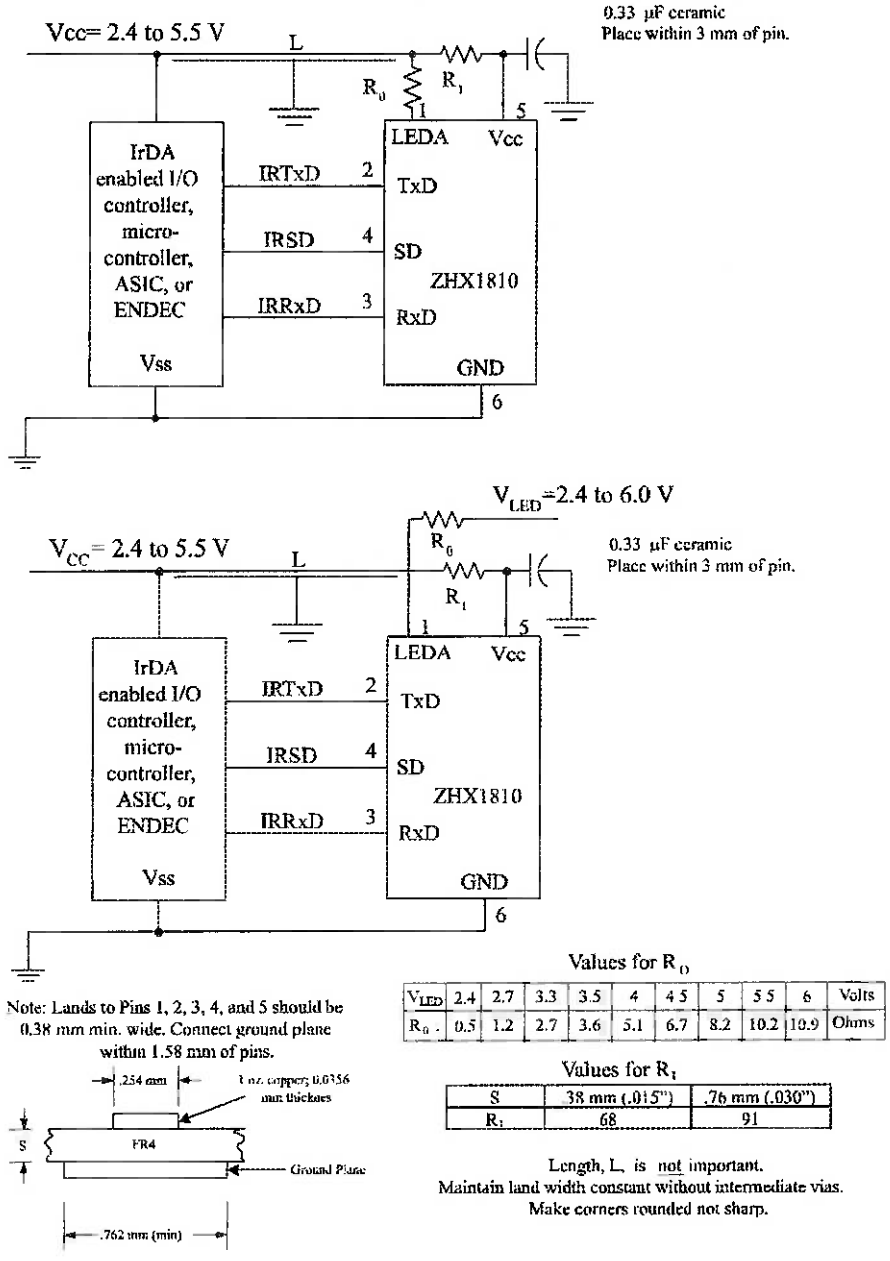


Figure 2. Application Block Diagrams

Electrical and Timing Specifications

Table 2 through Table 4 present the electrical and timing specifications for the ZHX1810 transceiver.

Table 2. Absolute Maximum Ratings

Parameter	Symbol	Minimum	Maximum	Unit	Comment
Supply Voltage	V_{CC}	–0.3	6.0	V	V_{CC}, GND
Input Voltage	V_{IN}	GND–0.3	V_{CC}+0.3	V	TxD, SD
Output (External) Voltage	V_{OUT}	GND–0.3	V_{CC}+0.3	V	RxD
LED Current	I_{AV}		500	mA	20% duty cycle, Ta=25 °C, t_{ON}≤90 µS
Storage Temperature	T_{ST}	–40	100	°C	
Solder Temperature	T_{SOL}		240	°C	

Table 3. Recommended Operating Conditions

Parameter	Symbol	Minimum	Maximum	Unit
Supply Voltage	V_{CC}	2.4	5.5	V
LED Voltage	V_{LED}	2.4	6.0	V
Ambient Operating Temperature	T_{OP}	–30	85	°C

Table 4. Electrical Characteristics

Parameter	Symbol	Condition	Min	Typical	Max	Unit	Remarks
High-Level Input Voltage	V_{IH}		0.6 V_{CC}		V_{CC}+0.5	V	TXD, SD
Low-Level Input Voltage	V_{IL}		–0.5		0.2 V_{CC}	V	TXD, SD
High-Level Output Voltage	V_{OH}		2.2			V	RxD
Low-Level Output Voltage	V_{OL}				0.4	V	RxD
Transmitter Current	I_{LED}			260		mA	
Listening Current	I_{CC}			90	150	µA	
Receive Current	I_{CC}			90	150	µA	

Unless otherwise noted: V_{CC}=3.3 V, GND= 0 V, T_A= 25 °C

Table 4. Electrical Characteristics (Continued)

Parameter	Symbol	Condition	Min	Typical	Max	Unit	Remarks
Standby Current	I_{STB}				1	µA	SD=V_{cc}, TxD=0 V
Optical Rise/Fall Time	t_{Rr}, t_{Rf}			100		nS	
RxD Pulse Width	t_{PWA}	SIR=115.2 Kbps	1.1	1.6	3.9	µS	
Power Shutdown Time	T_{SD}				1	µS	
Startup Time	T_{STU}				200	µS	
Latency	T_{RRT}				50	µS	
Trans. Radiant Intensity	I_E	I_{LED}=260 mA	40		100	mW/sr	θh, θv≤(±15°)
Min. Threshold Irradiance	E_{emin}	V_{cc}=3.3 V		2	3	µW/cm^2	θh, θv≤(±15°)
Angle of Half Intensity	θ			20		°	Hor. and Vert.
Light Pulse Rise, Fall Time	t_{or}, t_{of}			40		nS	
Optical Pulse Width	t_{OPW}			20		µS	TxD="H"
Optical Overshoot	t_{OPO}				3	%	
Peak Wavelength	λ_P			870		nm	
Unless otherwise noted: V_{cc}=3.3 V, GND= 0 V, T_A= 25 °C							

Figure 3 through Figure 6 show various electrical characteristics.

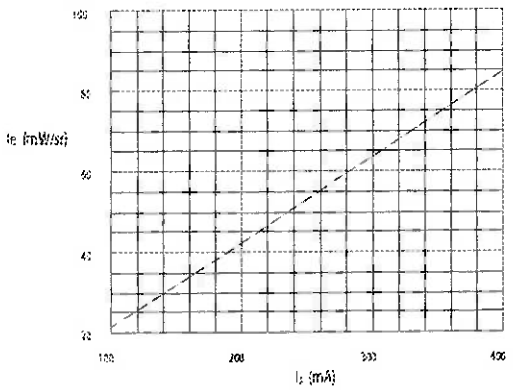


Figure 3. I_F-Ie Characteristics (0°)

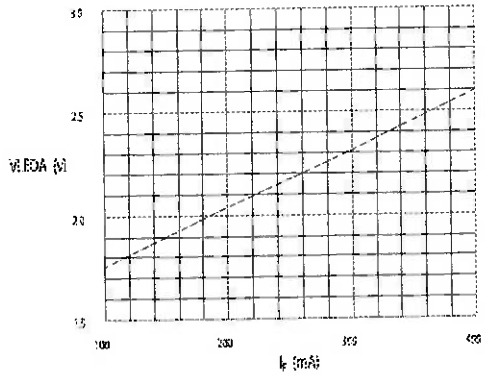


Figure 4. I_F-LEDA Characteristics (0°)

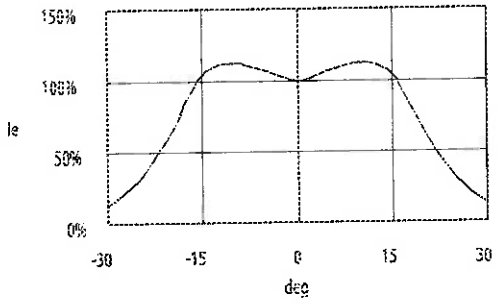


Figure 5. Directive Characteristics (Emitting)

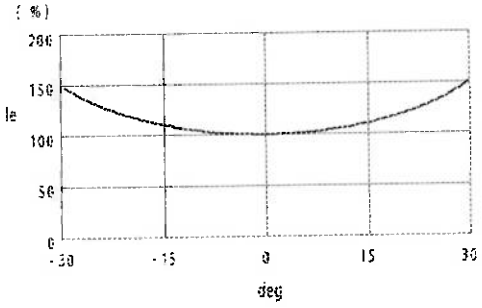
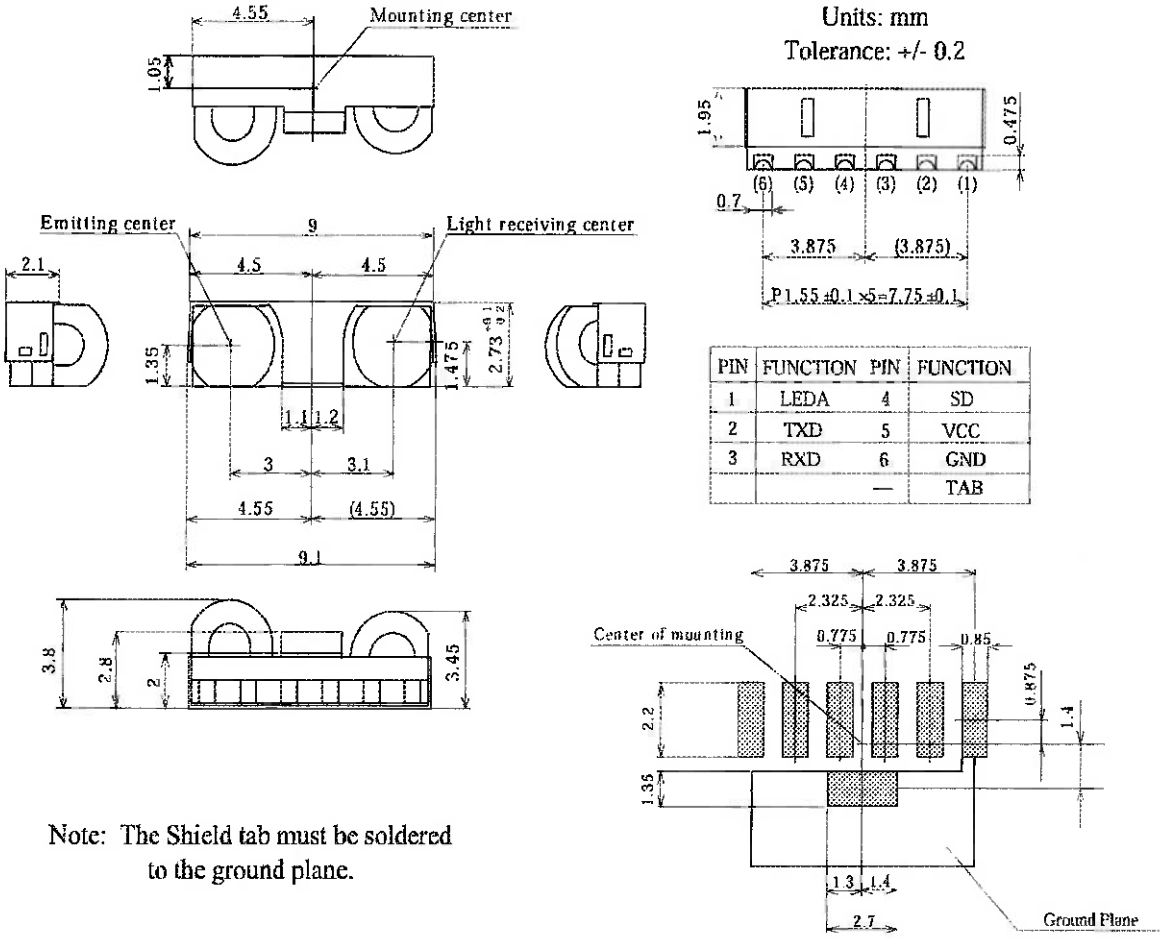


Figure 6. Directive Characteristics (Receiving)

Mechanical Drawings

Figure 7 shows the mechanical drawing for the ZHX1810 transceiver.



PIN	FUNCTION	PIN	FUNCTION
1	LEDA	4	SD
2	TXD	5	VCC
3	RXD	6	GND
		—	TAB

Note: The Shield tab must be soldered to the ground plane.

Devices will be marked ZHX1810.

Recommended Reflow Solder Pad Layout

Figure 7. ZHX1810 Mechanical Drawing

Soldering and Cleaning Recommendations

Follow these recommendations to maintain the performance of the ZHX1810 transceiver.

Reflow Soldering

- Reflow soldering paste is recommended:

  Melting temperature: 178 °C ~ 192 °C

  Composition: Sn 63%, Pb 37%
- The recommended thickness of the metal mask is between 0.2 mm and 0.25 mm for screen printing.
- Number of soldering times: 2 times *maximum*
- The temperature profile at the top surface of ZHX1810, shown in Figure 8, is recommended.

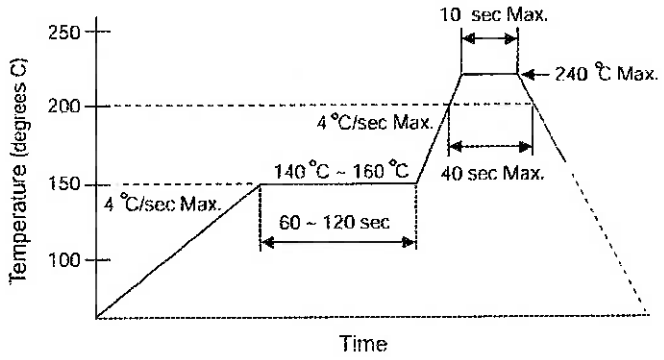


Figure 8. Temperature Profile at the Top Surface of ZHX1810

Manual Soldering

- Use 63/37 or silver solder.
- Use a soldering iron of 25 W or smaller. Adjust the temperature of the soldering iron below 300 °C.
- Finish soldering within 3 seconds.
- Handle only after ZHX1810 has cooled off.

Cleaning

Perform cleaning after soldering under the following conditions:

- Cleaning agent: Alcohol
- Temperature and time: 30 seconds below 50 °C or 3 minutes below 30 °C
- Ultrasonic cleaning: Below 20 W

Packing, Storage, and Baking Recommendations

Follow these recommendations to maintain the performance of the ZHX1810 transceiver.

Storage

To avoid moisture absorption, ZHX1810 reels must remain in the original, unopened moisture-proof packing. Parts must be soldered within 48 hours after unpacking. Reels that have been unpacked, but will not be soldered within 48 hours, must be stored in a desiccator.

Baking

Parts that have been stored over 12 months or unpacked over 48 hours must be baked under the following guidelines.

Reels

60 °C for 48 hours or more

Loose Parts

- 100 °C for 4 hours or more

  or

- 125 °C for 2 hours or more

  or

- 150 °C for 1 hour or more

Moisture-Proof Packing

In order to avoid moisture absorption during transportation and storage, ZHX1810 reels are packed in aluminum envelopes (see Figure 9) that contain a desiccant with a humidity indicator. The indicator changes color from blue to pink as moisture is absorbed.

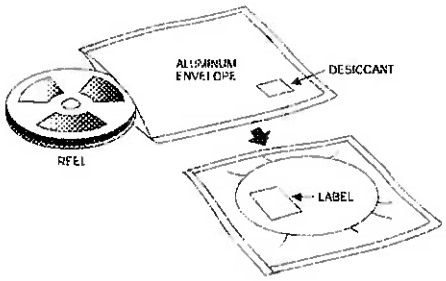


Figure 9. ZHX1810 Packaging

Taping Specifications

Figure 10 shows the reel dimensions for the ZHX1810. Figure 11 shows the tape dimensions and configuration for the ZHX1810.

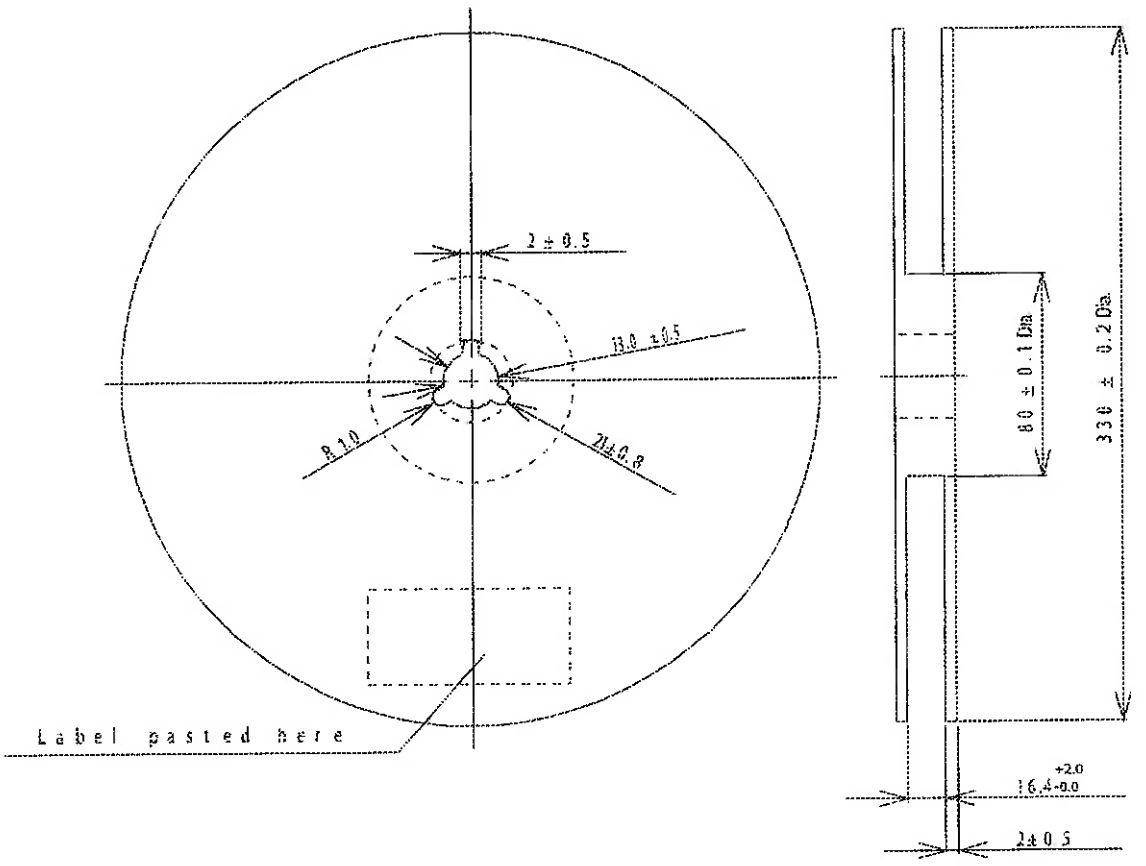
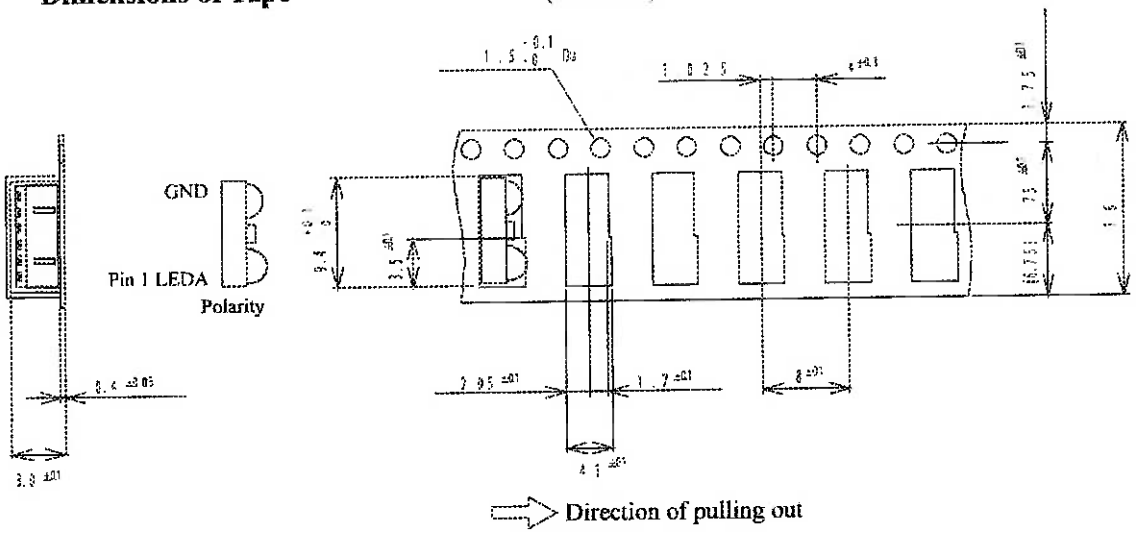
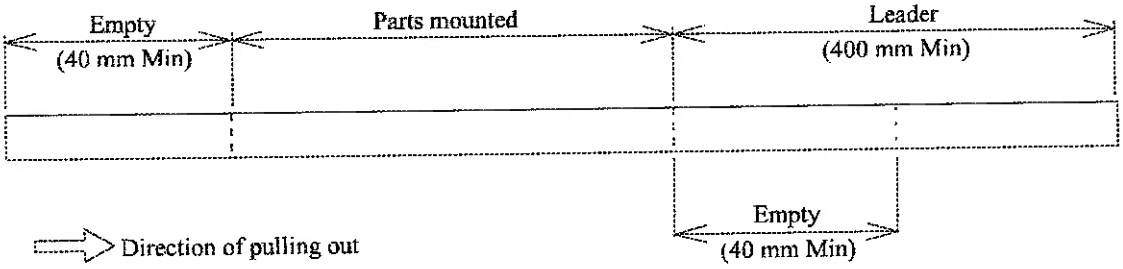


Figure 10. ZHX1810 Reel Dimensions (Unit: mm)

Dimensions of Tape (Unit: mm)



Configuration of Tape



Quantity: 2,000 pcs/reel

Figure 11. ZHX1810 Tape Dimensions and Configuration (Unit: mm)

Ordering Information

To order ZHX1810, use ZiLOG part number ZHX1810MV115THTR.

Note: In order to ensure the lowest possible lead times, ZiLOG uses two different fab sources for the transceiver IC. Both of these ICs have been extensively tested and qualified to meet the ZHX1810 transceiver specifications.

Customer Feedback Form

If you experience any problems while operating the ZHX1810 transceiver, or if you note any inaccuracies while reading this product specification, please copy and complete this form, then mail or fax it to ZiLOG (see "Return Information," below). We also welcome your suggestions!

Customer Information

Name	Country
Company	Phone
Address	Fax
City/State/Zip	email

Product Information

Serial # or Board Fab #/Rev #
Software Version
Document Number
Host Computer Description/Type

Return Information

ZiLOG
System Test/Customer Support
910 E. Hamilton Avenue, Suite 110, MS 4–3
Campbell, CA 95008
Fax: (408) 558-8536
Email: tools@zilog.com

Problem Description or Suggestion

Provide a complete description of the problem or your suggestion. If you are reporting a specific problem, include all steps leading up to the occurrence of the problem. Attach additional pages as necessary.

ANEXO IV –

Listagem do programa do microcontrolador

TF.C

```
/************************************************
*                                               *
*          Trabalho de Formatura - 2002         *
*         Interface IrDA Microprocessada        *
*               Programa Principal              *
*                                               *
* Erick Dario Leon Bueno de Camargo             *
* Orientador: Prof. Dr. Jun Okamoto Jr.         *
*                                               *
************************************************/

/*********************************************************/
/******************** Bibliotecas Usadas *****************/
/*********************************************************/

#include <8051io.h>                             /* Definicoes de I/O do 8051 */
#include <8051reg.h>                            /*  Definicoes de registradores do
8051 */
#include <8051int.h>                            /* Macro de interrupcoes do 8051 */
#include <8051bit.h>                            /* Manipulacao de bits */

#include "tf.h"                                 /* Definicoes do programa */

/*********************************************************/
/****************** Interrupcao do Timer 0 ***************/
/*********************************************************/

INTERRUPT(_TF0_) Timer0isr()
{
        base++;                                 /* ajuste de velocidade de
recepcao */
        if(base >= IRVEL)
        {
                base = 0;
                cont++;                         /* habilita IE0 */
                setbit(IE.0);

                au++;
                if(au>=2)
                {
                        au = 0;
                        setbit(P3.3);
                }
                else clrbit(P3.3);


        }                                       /*   se
        if(cont >= (8 + disp_ir + disp_ir))
 fim do byte... */
        {                                       /* volta para
                cont = 0;
 inicio do byte */                              /* desabilita T0 */
                clrbit(IE.1);
                if(!disp_ir)
                {
                        if(byte_rec == 0)
                        {
                                if(iRXwp!=7)
                                {
                                        if (iRXwp!=5)
                                        {
                                                if (iRXwp!=8)
                                                {
```

```
                                                        iRXwp = 0;
                                                        iRXrp = 0;
                                                }
                                        }
                                }
                        }
                }
                iRXBuf[iRXwp] = byte_rec;                       /* guarda no buffer
*/
                iRXwp++;                                                /* incrementa
ponteiro de escrita */
                if (iRXwp >= IRXSIZE)                           /* se no fim do
buffer */
                        iRXwp = 0;                                      /*
volta para inicio */
        }
};

/*********************************************************/
/****************** Interrupcao Externa 0 ****************/
/*********************************************************/

INTERRUPT(_IE0_) Ext_i0()
{
        if (cont == 0)                                          /* se inicio do byte... */
        {
                setbit(IE.1);                                           /* habilita T0 */
                byte_rec = 0xFF;                                        /* byte_rec <-
FF */
        }
        if (cont == (0 + disp_ir)) byte_rec = byte_rec & 0xFE;      /* grava '0' em
byte_rec.cont */
        else if (cont == (1 + disp_ir)) byte_rec = byte_rec & 0xFD;
        else if (cont == (2 + disp_ir)) byte_rec = byte_rec & 0xFB;
        else if (cont == (3 + disp_ir)) byte_rec = byte_rec & 0xF7;
        else if (cont == (4 + disp_ir)) byte_rec = byte_rec & 0xEF;
        else if (cont == (5 + disp_ir)) byte_rec = byte_rec & 0xDF;
        else if (cont == (6 + disp_ir)) byte_rec = byte_rec & 0xBF;
        else if (cont == (7 + disp_ir)) byte_rec = byte_rec & 0x7F;
        clrbit(IE.0);                                           /* desabilita IE0 */
};

/*********************************************************/
/****************** Interrupcao da Serial ****************/
/*********************************************************/

INTERRUPT(_SER_) serial_int()
{
        clrbit(IE.4);                                   /* desabilita interrupcao serial */
        if(SCON&1)                                              /* Se RI = 1 */
        {
                sRXBuf[sRXwp] = SBUF;                           /* guarda no buffer */
                sRXwp++;                                                /*         incrementa
ponteiro de escrita */
                if (sRXwp >= RXSIZE)                            /* se chegou no final do
buffer.. */
                        sRXwp = 0;                                      /* volta para
inicio */
                asm {
                        CLR SCON.0                                      /* RI <- 0 */
                };
        }
        setbit(IE.4);                                   /* habilita interrupcao serial */
}
```

```
/*************************************************************/
/************** Inicializacao das Interrupcoes ***************/
/*************************************************************/

ini_ints()
{
        SCON=0x50;                              // 8bit UART variable baudrate, TI=RI=0,
SM2=0
        TMOD=0x22;                      // Timer1 mode 2, Timer0 mode 2
        TCON=0x51;                      // Habilita Timer1 , Timer0 e interrupcao externa 0

        PCON=PCON|128;                          // Seta SMOD=PCON.7 para K=2 na formula de
baud rate
        TH1=243;                                // programa timer1
        TL1=243;                                // programa timer1

        TH0=243;                                // programa timer0
        TL0=243;                                // programa timer0

        IE=0x91;                                // Habilita interrupcoes: Serial int, EI0 e
T0
}

/*************************************************************/
/************** Inicializacao das Variaveis ******************/
/*************************************************************/

ini_vars()
{
        for (iRXrp = 0; iRXrp < IRXSIZE; iRXrp++) iRXBuf[iRXrp] = 0;

        sRXwp = 0;
        sRXrp = 0;
        cont = 0;
        base = 0;

        setbit(P3.3);                   /* porta indicacao de recebimento Ir */
        setbit(P3.7);                   /* porta para envio Ir */
        setbit(P1.0);                   /* LED 0 */
        setbit(P1.1);                   /* LED 1 */
        setbit(P1.2);                   /* LED 2 */
        setbit(P1.3);                   /* LED 3 */
        setbit(P1.4);                   /* LED 4 */
        setbit(P1.5);                   /* LED 5 */
        setbit(P1.6);                   /* LED 6 */
        setbit(P1.7);                   /* LED 7 */

        au =0;                          /* aux recebimento Ir */
}

/*************************************************************/
/************* Envia caracter ASCII pela serial **************/
/*************************************************************/

envia_char (unsigned char a)
{
        unsigned char i;
        asm {
                CLR SCON.1
        };
        SBUF = a;
        asm {
                JNB SCON.1,$
                CLR SCON.1
```

```
        };
        for(i=1;i<100;i++);
}
envia_char_ (char a)
{
        unsigned char i;
        asm {
                CLR SCON.1
        };
        SBUF = a;
        asm {
                JNB SCON.1,$
                CLR SCON.1
        };
        for(i=1;i<100;i++);
}

/*************************************************************/
/************* Envia caracter HEXA pela serial *************/
/*************************************************************/

envia_hex (unsigned char a)
{
        unsigned char u;
        d = a;
        asm {
                MOV A,d
                SWAP A
                MOV d,A
        };
        d = d & 0x0F;
        u = a & 0x0F;
        d = d + '0';
        if(d > '9') d = d - '0' - 10 + 'A';
        u = u + '0';
        if(u > '9') u = u - '0' - 10 + 'A';
        envia_char(' ');
        envia_char(d);
        envia_char(u);
}

/*************************************************************/
/******************** Programa Principal *********************/
/*************************************************************/

main()
{
        int aux2, i;
        unsigned char b, c, b_on;
        char aux;

        aux = 0;
        aux2 = 0;
        disp_ir = 0;
        disp_rs = 3;
        b_on = 0;

        ini_vars();                                    // inicializa variaveis
        ini_ints();                                    // inicializa interrupcoes


        for(;;)                                        // LOOP infinito
        {
                if(disp_rs == 1)
                {
```

```
                if (sRXwp != sRXrp)                    // se receber pela serial...
                {
                        b = sRXBuf[sRXrp];
                        sRXrp++;
                        if (sRXrp>=RXSIZE) sRXrp = 0;
                        envia_hex(b);                  //   escreve   o   q
receber em HEXA
                        if (b == 'b')                  // se receber 'b'...
                        {
                                envia_char('\n');
                                envia_char('\r');
                                envia_char(' ');
                                envia_char('I');
                                envia_char('r');
                                envia_char('B');
                                envia_char('u');
                                envia_char('f');
                                envia_char(':');
                                envia_char('\n');
                                envia_char('\r');
                                envia_hex(iRXrp);
                                envia_char(':');
                                envia_hex(iRXwp);
                                envia_char(':');
                                for(b = 0; b <= IRXSIZE; b++)
                                {
                                        envia_hex(iRXBuf[b]);
                                        envia_char(' ');
                                }
                                envia_char('\n');
                                envia_char('\r');
                                iRXrp = iRXwp;
                        }
                        else if (b == 'n')             // se receber 'n'...
                        {
                                envia_char('\n');
                                envia_char('\r');
                        }
                        else if (b == 'l')             // se receber 'l'...
                        {
                                clrbit(P1.2);
                        }
                        else if (b == 'h')             // se receber 'h'...
                        {
                                setbit(P1.2);
                        }
                        else if (b == 'm')             // se receber 'm'...
                        {
                                b_on = b_on + 1;
                                if(b_on > 1) b_on = 0;
                        }
                }
        } // fim do disp_rs

        if(!disp_ir)                                   //  se  disp_ir  =  Controle
remoto
        {
                setbit(P1.7);
                setbit(P1.6);
                setbit(P1.5);
                setbit(P1.4);
                setbit(P1.3);

                if(iRXwp != 0)                         //  se  receber  por
Ir...
```

```
{
        if(iRXBuf[0] == 0x00)
        {
                if(iRXBuf[1] == 0x20)
                {
                        clrbit(IE.0);
                        if(iRXBuf[7] == 0x40)
                        {
                                if(iRXBuf[9] == 0x30)
// cima
                                {
                                        aux = aux + 10;
                                        if (aux > 100) aux = 100;
                                        iRXwp = 0;
                                        clrbit(P1.7);
                                }
                        }
                        if(iRXBuf[7] == 0x18)
                        {
                                if(iRXBuf[8] == 0x04)
// baixo
                                {
                                        aux = aux - 10;
                                        if (aux < -100) aux = -100;
                                        iRXwp = 0;
                                        clrbit(P1.6);
                                }
                        }
                        if(iRXBuf[7] == 0x40)
// direita
                        {
                                if(iRXBuf[9] == 0x0C)
                                {
                                        aux2 = 1;
                                        iRXwp = 0;
                                        clrbit(P1.5);
                                }
                        }
                        if(iRXBuf[7] == 0x10)
//esquerda
                        {
                                if(iRXBuf[8] == 0x0C)
                                {
                                        aux2 = 2;
                                        iRXwp = 0;
                                        clrbit(P1.4);
                                }
                        }
                        if(iRXBuf[7] == 0x00)
// parar
                        {
                                if(iRXBuf[8] == 0x00)
                                {
                                        aux = 0;
                                        iRXwp = 0;
                                        clrbit(P1.3);
                                }
                        }
                        setbit(IE.0);
                }
        }
}
envia_char_(aux);
if(aux2 == 0) envia_char(0);
if(aux2 == 2)
```

```
                {
                        envia_char_(11);
                        aux2 = 0;
                }
                if(aux2 == 1)
                {
                        envia_char_(-11);
                        aux2 = 0;
                }
        } // fim do !disp_ir

        if(disp_ir)                                     // se disp_ir = PALM
        {
                setbit(P1.7);
                setbit(P1.6);
                setbit(P1.5);
                setbit(P1.4);
                setbit(P1.3);

                if (iRXwp != iRXrp)                     // se receber pela IR...
                {
                        b = iRXBuf[iRXrp];
                        iRXrp++;
                        if (iRXrp>=IRXSIZE) iRXrp = 0;
                        if(b_on)                                        //    se    for
mostrar...
                        {
                                envia_hex(b);                           // escreve o q
receber em HEXA
                        }
                        if (b == 0x61)                                  // cima = 'a'
                        {
                                aux = aux + 10;
                                if (aux > 100) aux = 100;
                                clrbit(P1.7);
                        }
                        if (b == 0x62)                                  // baixo = 'b'
                        {
                                aux = aux - 10;
                                if (aux < -100) aux = -100;
                                clrbit(P1.6);
                        }
                        if (b == 0x63)                                  // dir = 'c'
                        {
                                aux2 = 1;
                                clrbit(P1.5);
                        }
                        if (b == 0x64)                                  // esq = 'd'
                        {
                                aux2 = 2;
                                clrbit(P1.4);
                        }
                        if (b == 0x65)                                  // parar = 'e'
                        {
                                aux = 0;
                                clrbit(P1.3);
                        }
                }
                envia_char_(aux);
                if(aux2 == 0) envia_char(0);
                if(aux2 == 2)
                {
                        envia_char_(11);
                        aux2 = 0;
                }
```

```
                if(aux2 == 1)
                {
                        envia_char_(-11);
                        aux2 = 0;
                }
        } // fim do disp_ir

        if(!(P3&0x10))                          // se B1 -> muda disp_ir
        {
                disp_ir = disp_ir + 1;
                if(disp_ir == 2) disp_ir = 0;
                asm {
                        JNB P3.4,$
                }
        }
        if(!(P3&0x20))                          // se B2 -> muda disp_rs
        {
                disp_rs = disp_rs + 1;
                if(disp_rs >= 2)
                {
                        disp_rs = 0;
                        envia_char('\r');
                        envia_char('\r');
                        envia_char('\r');
                        envia_char('s');
                        envia_char('e');
                        envia_char('t');
                        envia_char(' ');
                        envia_char('m');
                        envia_char('o');
                        envia_char('d');
                        envia_char('e');
                        envia_char(' ');
                        envia_char('3');
                        envia_char('\r');
                }
                if(disp_rs == 1)
                {
                        sRXwp = 0;
                        sRXrp = 0;
                }
                asm {
                        JNB P3.5,$
                }
        }
        setbit(P1.0);
        if(disp_rs) clrbit(P1.0);
        setbit(P1.1);
        if(disp_ir) clrbit(P1.1);

        for(i=1;i<1000;i++);
    } // fim loop infinito
}
```

TF.H

```
/************************************************
*                                               *
*          Trabalho de Formatura - 2002         *
*         Interface IrDA Microprocessada        *
*             Definicoes do Programa            *
*                                               *
* Erick Dario Leon Bueno de Camargo             *
* Orientador: Prof. Dr. Jun Okamoto Jr.         *
*                                               *
************************************************/

/*********************************************************/
/*********************** Constantes **********************/
/*********************************************************/

/***********************/
/* Tamanho dos Buffers */
/***********************/
#define RXSIZE  8                               /* Tamanho do buffer RX serial */
#define IRXSIZE 32                              /* Tamanho do buffer RX Ir */
#define IRVEL           8                       /* velocidade de recepcao Ir */

/*********************************************************/
/******************** Variaveis Globais ******************/
/*********************************************************/

/**********/
/* Serial */
/**********/
register unsigned char sRXBuf[RXSIZE];          /* Buffer de Recepcao serial */
register unsigned char sRXwp, sRXrp;            /* Ponteiros de escrita e leitura do
RXBuf serial */
register unsigned char disp_rs;

/********/
/* IrDA */
/********/
register unsigned char cont;                    /* contador de bits */
register unsigned char base;                    /* contador de bits */
register unsigned char byte_rec;                /* byte recebido por Ir */
register unsigned char iRXBuf[IRXSIZE];         /* Buffer de Recepcao Ir */
register unsigned char iRXwp, iRXrp;            /* Ponteiros de escrita e leitura do
RXBuf Ir */
register unsigned char disp_ir;

/**********/
/* Outros */
/**********/
register unsigned char au;
register unsigned char d;
register unsigned char irvel;                   /* velocidade de recepcao Ir */
```

ANEXO V –

Listagem do programa para Palm

Controle.c

```
#include <Window.h>
#include "TF.h"
#include "TFRes.h"

// ********************************************************************
// ********************************************************************
// FUNCTION: ControleFormHandleEvent
//
// DESCRIPTION:
//
// This routine is the event handler for the "ControleForm" of this
// application.
//
// PARAMETERS:
//
// eventP
//     a pointer to an EventType structure
//
// RETURNED:
//     true if the event was handled and should not be passed to
//     FrmHandleEvent

Boolean ControleFormHandleEvent(EventType * eventP)
{
    Boolean handled = false;
    FormType * frmP;

    EvtResetAutoOffTimer();

    switch (eventP->eType)
        {
        case menuEvent:
            return MainFormDoCommand(eventP->data.menu.itemID);

        case frmOpenEvent:
            frmP = FrmGetActiveForm();
            FrmDrawForm(frmP);
            handled = true;
            break;

        case frmUpdateEvent:
            // To do any custom drawing here, first call
            // FrmDrawForm(), then do your drawing, and
            // then set handled to true.
            break;

               case ctlSelectEvent: // Buttom commands
               return ControleFormDoButtonCommand(eventP->data.ctlSelect.controlID);

        default:
            break;
        }

    return handled;
}


// ********************************************************************
// ********************************************************************
// FUNCTION: ControleFormDoButtomCommand
//
// DESCRIPTION: This routine performs the specified action required by a button press.
```

```
//
// PARAMETERS: command - the ID of the button that was pressed.
//
// RETURNED:    true if the event has been handled and should not be passed to a higher
level handler.
//

Boolean ControleFormDoButtonCommand(UInt16 command)
{
        Err error;
        Boolean handled = false;


        switch (command)
        {
                case ControlePrevButton:
                        FrmGotoForm(MainForm);
                        handled = true;
                        break;

                case ControleCimaButton:
                        SrmSend(portId, "1", 1, &error);
                        handled = true;
                        break;

                case ControleBaixoButton:
                        SrmSend(portId, "5", 1, &error);
                        handled = true;
                        break;

                case ControleEsqButton:
                        SrmSend(portId, "a", 1, &error);
                        handled = true;
                        break;

                case ControleDirButton:
                        SrmSend(portId, "e", 1, &error);
                        handled = true;
                        break;

                case ControlePararButton:
                        SrmSend(portId, "a", 1, &error);
                        handled = true;
                        break;

                default:
                   break;
        }

   return (handled);
}
```

Extras.c

```
// ********************************************************************
// ********************************************************************
// Extras.c
//

// ********************************************************************
// ********************************************************************
// FUNCTION: DrawNumbers
//
// DESCRIPTION: Draws 3x6 numbers.
//
void DrawNumbers (Int16 x, Int16 y, int numa)
{
        RectangleType rect;
        int num, a;

        if (numa > 99999) a = 100000;
        else if (numa > 9999) a = 10000;
        else if (numa > 999) a = 1000;
        else if (numa > 99) a = 100;
        else if (numa > 9) a = 10;
        else a = 1;
        num = (numa/a) - (int)(10*(int)(numa/(10*a)));
        while (a != 0)
        {
                rect.topLeft.x = x-1;
                rect.topLeft.y = y-1;
                rect.extent.x = 5;
                rect.extent.y = 7;
                WinEraseRectangle (&rect, 0);
                num = (numa/a) - (int)(10*(int)(numa/(10*a)));
                a = a/10;
                if (num == 0)
                {
                        WinDrawPixel(x+1, y);
                        WinDrawPixel(x, y+1);
                        WinDrawPixel(x+2, y+1);
                        WinDrawPixel(x, y+2);
                        WinDrawPixel(x+2, y+2);
                        WinDrawPixel(x, y+3);
                        WinDrawPixel(x+2, y+3);
                        WinDrawPixel(x+1, y+4);
                        WinDrawPixel(x, y);
                        WinDrawPixel(x+2, y);
                        WinDrawPixel(x, y+4);
                        WinDrawPixel(x+2, y+4);
                }
                if (num == 1)
                {
                        WinDrawPixel(x+1, y);
                        WinDrawPixel(x, y+1);
                        WinDrawPixel(x+1, y+1);
                        WinDrawPixel(x+1, y+2);
                        WinDrawPixel(x+1, y+3);
                        WinDrawPixel(x, y+4);
                        WinDrawPixel(x+1, y+4);
                        WinDrawPixel(x+2, y+4);
                }
                if (num ==2)
                {
                        WinDrawPixel(x, y);
                        WinDrawPixel(x+1, y);
                        WinDrawPixel(x+2, y);
```

```
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+4);
}
if (num == 3)
{
        WinDrawPixel(x, y);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x, y+4);
}
if (num == 4)
{
        WinDrawPixel(x, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
}
if (num == 5)
{
        WinDrawPixel(x, y);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+4);
}
if (num == 6)
{
        WinDrawPixel(x, y);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+4);
        WinDrawPixel(x, y+3);
}
if (num == 7)
{
```

```
                WinDrawPixel(x, y);
                WinDrawPixel(x+1, y);
                WinDrawPixel(x+2, y);
                WinDrawPixel(x, y+1);
                WinDrawPixel(x+2, y+1);
                WinDrawPixel(x+2, y+2);
                WinDrawPixel(x+2, y+3);
                WinDrawPixel(x+2, y+4);
            }
            if (num == 8)
            {
                WinDrawPixel(x+1, y);
                WinDrawPixel(x, y+1);
                WinDrawPixel(x+2, y+1);
                WinDrawPixel(x, y+2);
                WinDrawPixel(x+2, y+2);
                WinDrawPixel(x, y+3);
                WinDrawPixel(x+2, y+3);
                WinDrawPixel(x+1, y+4);
                WinDrawPixel(x+1, y+2);
                WinDrawPixel(x, y);
                WinDrawPixel(x+2, y);
                WinDrawPixel(x, y+4);
                WinDrawPixel(x+2, y+4);
            }
            if (num == 9)
            {
                WinDrawPixel(x+1, y);
                WinDrawPixel(x, y+1);
                WinDrawPixel(x+2, y+1);
                WinDrawPixel(x, y+2);
                WinDrawPixel(x+2, y+2);
                WinDrawPixel(x+2, y+3);
                WinDrawPixel(x+1, y+4);
                WinDrawPixel(x+1, y+2);
                WinDrawPixel(x, y);
                WinDrawPixel(x+2, y);
                WinDrawPixel(x, y+4);
                WinDrawPixel(x+2, y+4);
            }
            x += 4;
        }
}

// ***************************************************************************
// ***************************************************************************
// FUNCTION: DrawChars
//
// DESCRIPTION: Draws strings.
//
// aBcCdeHilLmMnoOpPrRstvVWxy :/-*0123456789
//
void DrawChars (Int16 x, Int16 y, char *str)
{
        int i = 0;
        RectangleType rect;

        while(str[i]!='\0')
        {
            switch(str[i])
            {
                case 'a':
                    rect.topLeft.x = x-1;
                    rect.topLeft.y = y-1;
                    rect.extent.x = 5;
```

```
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

    case 'A':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+3);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

    case 'b':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        x += 4;
        break;

    case 'B':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+3);
        x += 4;
```

```
        break;

case 'c':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'C':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'e':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'D':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+1);
```

```
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        x += 4;
        break;

case 'e':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'E':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'H':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'i':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
```

```
        rect.extent.x = 3;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        x += 2;
        break;

case 'I':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 3;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+3);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'J':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        x += 4;
        break;

case 'k':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'l':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
```

```
        rect.extent.x = 3;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        x += 2;
        break;

case 'L':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'm':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 7;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        WinDrawPixel(x+3, y+1);
        WinDrawPixel(x+4, y+2);
        WinDrawPixel(x+4, y+3);
        WinDrawPixel(x+4, y+4);
        x += 6;
        break;

case 'M':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
```

```
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

    case 'n':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

    case 'o':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        x += 4;
        break;

    case 'O':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        x += 4;
        break;

    case 'P':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y);
```

```
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+2, y+1);
        x += 4;
        break;

case 'p':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+3);
        WinDrawPixel(x+2, y+2);
        x += 4;
        break;

case 'r':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 4;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+1);
        x += 3;
        break;

case 'R':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 's':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-2;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+3);
        WinDrawPixel(x+1, y+4);
```

```
        WinDrawPixel(x+2, y+1);
        x += 4;
        break;

case 'S':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+3);
        x += 4;
        break;

case 't':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+3);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'v':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        x += 4;
        break;

case 'V':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+1);
```

```
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        x += 4;
        break;

case 'W':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 7;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+3, y+4);
        WinDrawPixel(x+4, y);
        WinDrawPixel(x+4, y+1);
        WinDrawPixel(x+4, y+2);
        WinDrawPixel(x+4, y+3);
        x += 6;
        break;

case 'x':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+3);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case 'y':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        x += 4;
        break;

case ':':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 3;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+3);
```

```
        x += 2;
        break;

case ' ':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 3;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        x += 2;
        break;

case '#':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+3);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case '0':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case '1':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+1, y+1);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+1, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;
```

```
case '2':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case '3':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x, y+4);
        x += 4;
        break;

case '4':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case '5':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
```

```
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case '6':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+1, y+2);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x, y+4);
        WinDrawPixel(x+1, y+4);
        WinDrawPixel(x+2, y+4);
        WinDrawPixel(x, y+3);
        x += 4;
        break;

case '7':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x, y);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x+2, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+2, y+4);
        x += 4;
        break;

case '8':
        rect.topLeft.x = x-1;
        rect.topLeft.y = y-1;
        rect.extent.x = 5;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        WinDrawPixel(x+1, y);
        WinDrawPixel(x, y+1);
        WinDrawPixel(x+2, y+1);
        WinDrawPixel(x, y+2);
        WinDrawPixel(x+2, y+2);
        WinDrawPixel(x, y+3);
        WinDrawPixel(x+2, y+3);
        WinDrawPixel(x+1, y+4);
```

```
                WinDrawPixel(x+1, y+2);
                WinDrawPixel(x, y);
                WinDrawPixel(x+2, y);
                WinDrawPixel(x, y+4);
                WinDrawPixel(x+2, y+4);
                x += 4;
                break;

        case '9':
                rect.topLeft.x = x-1;
                rect.topLeft.y = y-1;
                rect.extent.x = 5;
                rect.extent.y = 7;
                WinEraseRectangle (&rect, 0);
                WinDrawPixel(x+1, y);
                WinDrawPixel(x, y+1);
                WinDrawPixel(x+2, y+1);
                WinDrawPixel(x, y+2);
                WinDrawPixel(x+2, y+2);
                WinDrawPixel(x+2, y+3);
                WinDrawPixel(x+1, y+4);
                WinDrawPixel(x+1, y+2);
                WinDrawPixel(x, y);
                WinDrawPixel(x+2, y);
                WinDrawPixel(x, y+4);
                WinDrawPixel(x+2, y+4);
                x += 4;
                break;

        case '/':
                rect.topLeft.x = x-1;
                rect.topLeft.y = y-1;
                rect.extent.x = 5;
                rect.extent.y = 7;
                WinEraseRectangle (&rect, 0);
                WinDrawPixel(x, y+3);
                WinDrawPixel(x, y+4);
                WinDrawPixel(x+1, y+2);
                WinDrawPixel(x+2, y+1);
                WinDrawPixel(x+2, y);
                x += 4;
                break;

        case '-':
                rect.topLeft.x = x-1;
                rect.topLeft.y = y-1;
                rect.extent.x = 4;
                rect.extent.y = 7;
                WinEraseRectangle (&rect, 0);
                WinDrawPixel(x, y+2);
                WinDrawPixel(x+1, y+2);
                x += 3;
                break;

        }
        i++;
    }
}
```

Irda.c

```
// ********************************************************************
// ********************************************************************
// Irda.c
//

#include <PalmOS.h>
#include <SerialMgr.h>

// MathLib headers
#include <MathLib.h>

#include "TF.h"
#include "TFRec.h"

// ********************************************************************
// Global variables
// ********************************************************************

UInt16 portId;
void *customSerQP;
Int32 baudvalue;

// ********************************************************************
// Internal Constants
// ********************************************************************

#define myCustomSerQueueSize 1024

// ********************************************************************
// ********************************************************************
// FUNCTION: SerialStart
//
// DESCRIPTION:  Open serial port.
//
// CONFIGURATION: - 9600 baud rate
//
// RETURNED:
//     err - errNone if nothing went wrong

Err SerialStart(void)
{
        Err   err = errNone;

        err = SrmOpen(serPortIrPort /* port */, 9600 /* baud */, &portId);
        if (err) {
        if (err == serErrAlreadyOpen) {
                FrmAlert(SerialInUseAlert);
                SrmClose(portId);
        } else
                FrmAlert(CantOpenSerialAlert);
        return err;
    }
    SrmControl(portId, srmCtlIrDAEnable, NULL, 0);
        err = SerialConfig();

        SetReceiveBuffer();
        SrmReceiveFlush(portId, 0);

        baudvalue = 9600;

    return err;
}
```

```
// ********************************************************************
// ********************************************************************
// FUNCTION: SetReceiveBuffer
//
// DESCRIPTION: Set buffer with a receive queue of 'myCustomSerQueueSize' bytes

Err SetReceiveBuffer(void)
{
        Err err;
        customSerQP = MemPtrNew(myCustomSerQueueSize);
        if (customSerQP) {
                err = SrmSetReceiveBuffer(portId, customSerQP, myCustomSerQueueSize);
        } else {
                FrmAlert(NotEnoughSpaceAlert);
                return memErrNotEnoughSpace;
        }
        return errNone;
}

// ********************************************************************
// ********************************************************************
// FUNCTION: SerialConfig
//
// DESCRIPTION:  Configure serial port.
//
// CONFIGURATION: - 1 stop bit
//                - 8 data bits
//                - Flow Control disabled
//
// RETURNED:
//     err - errNone if nothing went wrong

Err SerialConfig(void)
{
        Err err = errNone;
        UInt16 paramSize;
        UInt32 flags = srmSettingsFlagBitsPerChar8 |
         srmSettingsFlagStopBits1| srmSettingsFlagRTSInactive;

        paramSize = sizeof(flags);
        err = SrmControl(portId, srmCtlSetFlags, &flags, &paramSize);

        return err;
}

// ********************************************************************
// ********************************************************************
// FUNCTION: SerialStop
//
// DESCRIPTION: Restore the defaut buffer and close serial port.

void SerialStop(void)
{
        SetDefautReceiveBuffer();
        SrmClose(portId);
}

// ********************************************************************
// ********************************************************************
// FUNCTION: SetDefautReceiveBuffer
//
// DESCRIPTION: Restore the defaut buffer.

void SetDefautReceiveBuffer(void)
```

```
{
        SrmSetReceiveBuffer(portId, NULL, 0);
        if (customSerQP)
        {
                MemPtrFree(customSerQP);
                customSerQP = NULL;
        }
}

// ********************************************************************
// ********************************************************************
// FUNCTION: SerialSend
//
// DESCRIPTION: Manda dados pela porta Ir.

void SerialSend(void *sendbuf, UInt32 taman)
{
        Err error;
        UInt32 tama;
        tama = SrmSend(portId, sendbuf, taman, &error);
        if(tama != taman) FrmAlert(CantOpenSerialAlert);

}

// ********************************************************************
// ********************************************************************
// FUNCTION: GetFieldText
//

Char *GetFieldText(UInt16 fieldID)
{
        MemHandle               txtH;                                   // Handle para
o Texto
        Char                    *txtP;                                  //    Ponteiro
para o Texto
        FormPtr                 frm = FrmGetActiveForm();       // Ponteiro para o Form
        FieldType               *fldP;                                  //    Ponteiro
para o Campo de Edição

        // Encontra o campo informado
        fldP = FrmGetObjectPtr(frm, FrmGetObjectIndex(frm, fieldID));

        // Pega Handle do texto do Campo de edição
        txtH = FldGetTextHandle(fldP);

        // Se não houver texto no campo, retornamos NULL
        if (!txtH)
                return NULL;

        // Bloqueia a área de memória do Palm onde está o texto
        txtP = MemHandleLock(txtH);

        // Se não obtermos um Ponteiro válido para o texto, retornamos NULL
        if (!txtP)
                return NULL;

        // Desbloqueia a memória
        MemHandleUnlock(txtH);

        return txtP;
}

// ********************************************************************
// ********************************************************************
// FUNCTION: SetFieldText
```

```
//

void SetFieldText(UInt16 fieldID, Char *strP)
{
        MemHandle               txtH, oldtxtH;                                  //
Novo e Antigo Handle
        Char                    *txtPtr;                                        //
Ponteiro para o Texto
        FormPtr                 frm = FrmGetActiveForm();               //  Ponteiro  para  a
Form
        FieldType               *fldP;                                          //
Ponteiro para o Campo de Edição

        // Se o ponteiro para o novo texto não for valido, sai
        if (!strP)
                return;

        // Encontra o campo informado
        fldP = FrmGetObjectPtr(frm, FrmGetObjectIndex(frm, fieldID));

        // Salva o Handle anterior deste campo para podermos
        // libera-lo depois de colocar no novo Handle
        oldtxtH = FldGetTextHandle(fldP);

        // Aloca um Handle para a String adequado ao tamanho
        // da String que sera armazenada nele
        txtH = MemHandleNew(StrLen(strP) + 1);
        if (!txtH)
                return; // Se não consequir alocar memória, sai

        // Bloqucia o Handle para podermos alterar seu conteúdo
        txtPtr = MemHandleLock(txtH);
        if (!txtPtr)
                return; // Se não conseguir bloquear memória, sai

        // Copia a String para o Handle Bloqueado
        StrCopy(txtPtr, strP);

        // Desbloqueia o Handle
        MemHandleUnlock(txtH);

        // Coloca o novo texto
        FldSetTextHandle(fldP, txtH);

        // Redesenha o Campo
        FldDrawField(fldP);

        // Libera o Handle anterior
        if (oldtxtH)
                MemHandleFree(oldtxtH);
}

// ********************************************************************
// ********************************************************************
// FUNCTION: Envia
//
// DESCRIPTION: Manda dados pela porta Ir.

void Envia()
{
        char *texto;

        texto = GetFieldText(MainEnvField);
        SerialSend(texto, StrLen(texto));
//      SerialSend(GetFieldText(MainEnvField));
```

```
}

// ********************************************************************
// ********************************************************************
// FUNCTION: RecebeTexto
//
// DESCRIPTION: Recebe dados pela porta Ir.

void RecebeTexto()
{
        char textoRec[100];
        int BitsRcvd = 0;

        do
        {
                textoRec[BitsRcvd] = RecebeChar();
                if ((textoRec[BitsRcvd] == 3) || (textoRec[BitsRcvd] == 4))
                {
                        textoRec[BitsRcvd] = '\0';
                        SrmReceiveFlush(portId, 1);
                }
                BitsRcvd += 1;
        } while (textoRec[BitsRcvd - 1] != '\0');
        if (BitsRcvd > 1)
        {
                SetFieldText(MainRecField, textoRec);
        }
}

// ********************************************************************
// ********************************************************************
// FUNCTION: RecebeChar
//
// DESCRIPTION: Recebe caracter pela porta Ir.

char RecebeChar()
{
        UInt32 byteRcvd, waitTime;
        char rcvData[1];
        Err error;

        waitTime = SysTicksPerSecond() / 5;

        byteRcvd = SrmReceive(portId, rcvData, 1, waitTime, &error);
        if ((error == serErrTimeOut) && (byteRcvd == 0))
        {
                SrmClearErr(portId);
                SrmReceiveFlush(portId, 1);
                rcvData[0] = 3;
        }
        if ((error) && (error != serErrTimeOut))
        {
                SrmClearErr(portId);
                SrmReceiveFlush(portId, 1);
                rcvData[0] = 4;
        }
        return rcvData[0];
}

// ********************************************************************
// ********************************************************************
// FUNCTION: BaudRate
//
// DESCRIPTION: Muda Baud Rate
```

```
void BaudRate(int i)
{
        UInt16 dataLen = sizeof(Int32);
        RectangleType rect;

        if (i == 1)
        {
                if (baudvalue ==  1200) baudvalue = 1300;
                else if (baudvalue ==  1300) baudvalue = 1400;
                else if (baudvalue ==  1400) baudvalue = 1500;
                else if (baudvalue ==  1500) baudvalue = 1600;
                else if (baudvalue ==  1600) baudvalue = 1700;
                else if (baudvalue ==  1700) baudvalue = 1800;
                else if (baudvalue ==  1800) baudvalue = 1900;
                else if (baudvalue ==  1900) baudvalue = 2000;
                else if (baudvalue ==  2000) baudvalue = 2100;
                else if (baudvalue ==  2100) baudvalue = 2200;
                else if (baudvalue ==  2200) baudvalue = 2300;
                else if (baudvalue ==  2300) baudvalue = 2400;
                else if (baudvalue ==  2400) baudvalue = 4800;
                else if (baudvalue ==  4800) baudvalue = 9600;
                else if (baudvalue ==  9600) baudvalue = 19200;
                else if (baudvalue == 19200) baudvalue = 38400;
                else if (baudvalue == 38400) baudvalue = 57600;
                else if (baudvalue == 57600) baudvalue = 115200;
        }

        if (i == 0)
        {
                if (baudvalue ==  1300) baudvalue = 1200;
                else if (baudvalue ==  1400) baudvalue = 1300;
                else if (baudvalue ==  1500) baudvalue = 1400;
                else if (baudvalue ==  1600) baudvalue = 1500;
                else if (baudvalue ==  1700) baudvalue = 1600;
                else if (baudvalue ==  1800) baudvalue = 1700;
                else if (baudvalue ==  1900) baudvalue = 1800;
                else if (baudvalue ==  2000) baudvalue = 1900;
                else if (baudvalue ==  2100) baudvalue = 2000;
                else if (baudvalue ==  2200) baudvalue = 2100;
                else if (baudvalue ==  2300) baudvalue = 2200;
                else if (baudvalue ==  2400) baudvalue = 2300;
                else if (baudvalue ==  4800) baudvalue = 2400;
                else if (baudvalue ==  9600) baudvalue = 4800;
                else if (baudvalue == 19200) baudvalue = 9600;
                else if (baudvalue == 38400) baudvalue = 19200;
                else if (baudvalue == 57600) baudvalue = 38400;
                else if (baudvalue == 115200) baudvalue = 57600;
        }

        rect.topLeft.x = 24;
        rect.topLeft.y = 147;
        rect.extent.x = 18;
        rect.extent.y = 7;
        WinEraseRectangle (&rect, 0);
        DrawNumbers (25,148,(baudvalue/100));

    SrmControl(portId, srmCtlSetBaudRate, &baudvalue, &dataLen);
}
```

Tf.h

```
// TF.h
//
// header file for TF
//

#ifndef TF_H_
#define TF_H_

// ***********************************************************************
// Prototypes
// ***********************************************************************

void * GetObjectPtr(UInt16 objectID);
Boolean ControleFormHandleEvent(EventType * eventP);
Boolean ControleFormDoButtonCommand(UInt16 command);
Boolean MainFormDoCommand(UInt16 command);
void DrawChars (Int16 x, Int16 y, char *str);
void DrawVel(int i);
void DrawDir();
Err SerialStart(void);
Err SerialConfig(void);
Err SetReceiveBuffer(void);
void SerialStop(void);
void SetDefautReceiveBuffer(void);
void SerialSend(void *sendbuf, UInt32 tamal);
Char *GetFieldText(UInt16 fieldID);
void Envia();
void RecebeTexto();
char RecebeChar();
void SetFieldText(UInt16 fieldID, Char *strP);
void BaudRate(int i);
void DrawNumbers (Int16 x, Int16 y, int numa);

// ***********************************************************************
// Internal Structures
// ***********************************************************************

typedef struct TFPreferenceType
{
        UInt8 replaceme;
} TFPreferenceType;

// ***********************************************************************
// Global variables
// ***********************************************************************

extern TFPreferenceType g_prefs;
extern UInt16 portId;
extern void *customSerQP;
extern Int32 baudvalue;

// ***********************************************************************
// Internal Constants
// ***********************************************************************

#define appFileCreator              'ELTF'
#define appName                             "TF"
#define appVersionNum               0x01
#define appPrefID                       0x00
#define appPrefVersionNum           0x01

#endif // TF_H_
```

TfMain.c

```
// TFMain.c
//

#include <PalmOS.h>

// MathLib headers
#include <MathLib.h>
#include <math.h>

#include "TF.h"
#include "TFRsc.h"

// ********************************************************************
// Entry Points
// ********************************************************************

// ********************************************************************
// Global variables
// ********************************************************************
Boolean Asc2;


// g_prefs
// cache for application preferences during program execution
TFPreferenceType g_prefs;

// ********************************************************************
// Internal Constants
// ********************************************************************

// Define the minimum OS version we support
#define ourMinVersion    sysMakeROMVersion(3,3,0,sysROMStageDevelopment,0)
#define kPalmOS10Version sysMakeROMVersion(1,0,0,sysROMStageRelease,0)

// ********************************************************************
// Internal Functions
// ********************************************************************

// ********************************************************************
// ********************************************************************
// FUNCTION: GetObjectPtr
//
// DESCRIPTION:
//
// This routine returns a pointer to an object in the current form.
//
// PARAMETERS:
//
// formId
//     id of the form to display
//
// RETURNED:
//     address of object as a void pointer

void * GetObjectPtr(UInt16 objectID)
{
    FormType * frmP;

    frmP = FrmGetActiveForm();
    return FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, objectID));
}

// ********************************************************************
```

```
// ********************************************************************
// FUNCTION: MainFormInit
//
// DESCRIPTION: This routine initializes the MainForm form.
//
// PARAMETERS:
//
// frm
//     pointer to the MainForm form.

static void MainFormInit(FormType * /*frmP*/)
{
}

// ********************************************************************
// ********************************************************************
// FUNCTION: MainFormDoButtomCommand
//
// DESCRIPTION: This routine performs the specified action required by a button press.
//
// PARAMETERS: command - the ID of the button that was pressed.
//
// RETURNED:     true if the event has been handled and should not be passed to a higher
level handler.
//

static Boolean MainFormDoButtonCommand(UInt16 command)
{
        Boolean handled = false;

        switch (command)
        {
                case MainNextButton:
                        FrmGotoForm(ControleForm);
                        handled = true;
                        break;

                case MainMaisButton:
                        BaudRate(1);
                        handled = true;
                        break;

                case MainMenosButton:
                        BaudRate(0);
                        handled = true;
                        break;

                case MainEnviarButton:
                        Envia();
                        handled = true;
                        break;

                default:
                   break;
        }

   return (handled);
}

// ********************************************************************
// ********************************************************************
// FUNCTION: MainFormDoCommand
//
// DESCRIPTION: This routine performs the menu command specified.
//
```

```
// PARAMETERS:
//
// command
//     menu item id

Boolean MainFormDoCommand(UInt16 command)
{
    Boolean handled = false;
    FormType * frmP;

    switch (command)
    {
        case OptionsAboutTF:

            // Clear the menu status from the display
            MenuEraseStatus(0);

            // Display the About Box.
            frmP = FrmInitForm (AboutForm);
            FrmDoDialog (frmP);
            FrmDeleteForm (frmP);

            handled = true;
            break;
   }

   return handled;
}

// ********************************************************************
// ********************************************************************
// FUNCTION: MainFormHandleEvent
//
// DESCRIPTION:
//
// This routine is the event handler for the "MainForm" of this
// application.
//
// PARAMETERS:
//
// eventP
//     a pointer to an EventType structure
//
// RETURNED:
//     true if the event was handled and should not be passed to
//     FrmHandleEvent

static Boolean MainFormHandleEvent(EventType * eventP)
{
    Boolean handled = false;
    FormType * frmP;

        EvtResetAutoOffTimer();

    switch (eventP->eType)
    {
        case menuEvent:
            return MainFormDoCommand(eventP->data.menu.itemID);

        case frmOpenEvent:
            frmP = FrmGetActiveForm();
            MainFormInit(frmP);
            FrmDrawForm(frmP);
            WinDrawLine(80,20,80,120);
            WinDrawLine(5,125,155,125);
```

```
            DrawChars(20,20,"Enviando:");
            DrawChars(100,20,"Recebendo:");
            Asc2 = true;
                        DrawNumbers (25,148,(baudvalue/100));
            handled = true;
            break;

        case frmUpdateEvent:
            // To do any custom drawing here, first call
            // FrmDrawForm(), then do your drawing, and
            // then set handled to true.
            break;

        case nilEvent:
                RecebeTexto();
                        handled = true;
                break;

                case ctlSelectEvent: // Buttom commands
                return MainFormDoButtonCommand(eventP->data.ctlSelect.controlID);

        default:
            break;
        }

    return handled;
}

// ********************************************************************
// ********************************************************************
// FUNCTION: AppHandleEvent
//
// DESCRIPTION:
//
// This routine loads form resources and set the event handler for
// the form loaded.
//
// PARAMETERS:
//
// event
//     a pointer to an EventType structure
//
// RETURNED:
//     true if the event was handled and should not be passed
//     to a higher level handler.

static Boolean AppHandleEvent(EventType * eventP)
{
    UInt16 formId;
    FormType * frmP;

    if (eventP->eType == frmLoadEvent)
    {
        // Load the form resource.
        formId = eventP->data.frmLoad.formID;
        frmP = FrmInitForm(formId);
        FrmSetActiveForm(frmP);

        // Set the event handler for the form.  The handler of the
        // currently active form is called by FrmHandleEvent each
        // time is receives an event.
        switch (formId)
        {
        case MainForm:
```

```
            FrmSetEventHandler(frmP, MainFormHandleEvent);
            break;

        case ControleForm:
            FrmSetEventHandler(frmP, ControleFormHandleEvent);
            break;

        default:
            break;

        }
        return true;
    }

    return false;
}

// ********************************************************************
// ********************************************************************
// FUNCTION: AppEventLoop
//
// DESCRIPTION: This routine is the event loop for the application.

static void AppEventLoop(void)
{
    UInt16 error;
    EventType event;

    do {
        // change timeout if you need periodic nilEvents
        EvtGetEvent(&event, 0);

        if (! SysHandleEvent(&event))
        {
            if (! MenuHandleEvent(0, &event, &error))
            {
                if (! AppHandleEvent(&event))
                {
                    FrmDispatchEvent(&event);
                }
            }
        }
    } while (event.eType != appStopEvent);
}

// ********************************************************************
// ********************************************************************
// FUNCTION: AppStart
//
// DESCRIPTION:  Get the current application's preferences.
//
// RETURNED:
//     errNone - if nothing went wrong

static Err AppStart(void)
{
    UInt16 prefsSize;
    Err err = errNone;

    // Read the saved preferences / saved-state information.
    prefsSize = sizeof(TFPreferenceType);
    if (PrefGetAppPreferences(
        appFileCreator, appPrefID, &g_prefs, &prefsSize, true) !=
        noPreferenceFound)
    {
```

```
    }

    err = SerialStart();
    return err;
}

// ********************************************************************
// ********************************************************************
// FUNCTION: AppStop
//
// DESCRIPTION: Save the current state of the application.

static void AppStop(void)
{
    // Write the saved preferences / saved-state information.  This
    // data will be saved during a HotSync backup.
    PrefSetAppPreferences(
        appFileCreator, appPrefID, appPrefVersionNum,
        &g_prefs, sizeof(TFPreferenceType), true);

    SerialStop();

    // Close all the open forms.
    FrmCloseAllForms();
}

// all code from here to end of file should use no global variables
#pragma warn_a5_access on

// ********************************************************************
// ********************************************************************
// FUNCTION: RomVersionCompatible
//
// DESCRIPTION:
//
// This routine checks that a ROM version is meet your minimum
// requirement.
//
// PARAMETERS:
//
// requiredVersion
//     minimum rom version required
//     (see sysFtrNumROMVersion in SystemMgr.h for format)
//
// launchFlags
//     flags that indicate if the application UI is initialized
//     These flags are one of the parameters to your app's PilotMain
//
// RETURNED:
//     error code or zero if ROM version is compatible

static Err RomVersionCompatible(UInt32 requiredVersion, UInt16 launchFlags)
{
    UInt32 romVersion;

    // See if we're on in minimum required version of the ROM or later.
    FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
    if (romVersion < requiredVersion)
    {
        if ((launchFlags &
            (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp)) ==
            (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp))
        {
            FrmAlert (RomIncompatibleAlert);
```

```
            // Palm OS 1.0 will continuously relaunch this app unless
            // we switch to another safe one.
            if (romVersion <= kPalmOS10Version)
            {
                AppLaunchWithCommand(
                    sysFileCDefaultApp,
                    sysAppLaunchCmdNormalLaunch, NULL);
            }
        }

        return sysErrRomIncompatible;
    }

    return errNone;
}

// ********************************************************************
// ********************************************************************
// FUNCTION: TFPalmMain
//
// DESCRIPTION: This is the main entry point for the application.
//
// PARAMETERS:
//
// cmd
//     word value specifying the launch code.
//
// cmdPB
//     pointer to a structure that is associated with the launch code
//
// launchFlags
//     word value providing extra information about the launch
//
// RETURNED:
//     Result of launch, errNone if all went OK

static UInt32 TFPalmMain(
    UInt16 cmd,
    MemPtr /*cmdPBP*/,
    UInt16 launchFlags)
{
    Err error;

    error = RomVersionCompatible (ourMinVersion, launchFlags);
    if (error) return (error);

    switch (cmd)
    {
    case sysAppLaunchCmdNormalLaunch:
        error = AppStart();
        if (error)
            return error;

        // Start application by opening the main form
        // and then entering the main event loop
        FrmGotoForm(MainForm);
        AppEventLoop();

        AppStop();
        break;

    default:
        break;
    }
```

```
    return errNone;
}

// ********************************************************************
// ********************************************************************
// FUNCTION: PilotMain
//
// DESCRIPTION: This is the main entry point for the application.
//
// PARAMETERS:
//
// cmd
//     word value specifying the launch code.
//
// cmdPB
//     pointer to a structure that is associated with the launch code
//
// launchFlags
//     word value providing extra information about the launch.
//
// RETURNED:
//     Result of launch, errNone if all went OK

UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
    return TFPalmMain(cmd, cmdPBP, launchFlags);
}

// turn a5 warning off to prevent it being set off by C++
// static initializer code generation
#pragma warn_a5_access reset
```