Universidade São Francisco
Engenharia de Computação

ANDERSON INVENCIONE LEITE SILVA

HARDWARE MICRO-CONTROLADO PARA AUTOMAÇÃO RESIDENCIAL COM CONTROLE REMOTO VIA WEB

Itatiba
2010

ANDERSON INVENCIONE LEITE SILVA

HARDWARE MICRO-CONTROLADO PARA AUTOMAÇÃO RESIDENCIAL COM CONTROLE REMOTO VIA WEB

Monografia apresentada ao curso de Engenharia de Computação da Universidade São Francisco, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Claudio Kiyoshi Umezu

Itatiba
2010

SILVA, ANDERSON INVENCIONE LEITE. **Hardware micro-controlado para automação residencial com controle remoto via WEB**. 2010. Monografia (Graduação em Engenharia de Computação) – Universidade São Francisco, Itatiba.

RESUMO

Domótica é a fusão da palavra em latim Domus, que significa casa, com robótica, também conhecida por Automação Residencial. A Domótica integra computadores, circuitos eletrônicos e interfaces de comunicação com o objetivo de controlar, monitorar e automatizar os recursos disponíveis nas residências, como por exemplo: iluminação, eletrodomésticos, segurança, climatização, etc. Com a Domótica, processos manuais que são executados diariamente nas casas, podem ser automatizados, gerando mais conforto e praticidade no dia a dia. Residências inteligentes e sistemas domóticos têm atraído crescente interesse, uma vez que possibilitam a atuação, supervisionada ou não, de dispositivos eletrônicos em uma residência, exercendo tarefas complexas e interagindo com usuários e com o meio físico. O objetivo deste trabalho é construir um hardware micro-controlado, baseado no PIC16F877A da Microchip, para controlar ações de uma casa remotamente via WEB. Foram utilizados módulos de comunicação serial, acionamento de dispositivos através de PWM (modulação por largura de pulso), além de entradas analógicas e saídas digitais.

Palavras-chave: PIC, HARDWARE, DOMÓTICA, SERIAL.

ABSTRACT

Domotica, in portuguese, is the combination between the Latin word Domus, which means home, and robotic, known as Home automation. The Domotica integrates computers, electronics circuits and communication's interfaces in order to control, monitor and automate home's available resources like: lighting, appliances, security, air condition, etc. With Domotica, manual process, which is executed daily in homes, can be automated, creating more comfort and convenience to life. Smart homes and home automation systems attract increasing interest, once they provide the performance, supervised or not, of electronics devices in a residence, performing complex tasks and interacting with users and the physical environment. The objective of this work is to build a micro-controlled hardware, based in PIC16F877A, from Microchip, to control home actions remotely via WEB. Communication serial modules, activation devices through PWM (Pulse Width Modulation), beyond analogical input and digital output, be used.

Key words: PIC, HARDWARE, DOMOTICA, SERIAL.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ANSI	*American National Standards Institute*
ANSI	*American National Standards Institute*
A/D	Analógico / Digital
CCP	*Capture*/*Compare*/PWM
CPU	*Central Processing Unit*
DCE	Equipamento de Comunicação de Dados
DTE	Equipamento de Dados Terminal
EEPROM	*Electrically Erasable Programmable Read Only Memory*
EIA	*Electronic Industries Association*
I/O	*Input* / *Output*
I2C	*Inter Intergrated Circuit*
ICSP	*InCircuit Serial Programming*
IDE	*Integrated Development Environment*
LED	*Light Emitting Diode*
LSB	*Least significant Bit*
MSB	*Most significant Bit*
PC	*Personal Computer*
PDIP	*Plastic Dual Inline Package*
PIC	*Programmable Interface Controller*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
SBCI	Sensor Baseado em Circuito Integrado
SPI	*Serial Peripheral Interface*
SRAM	*Static Random Access Memory*
TTL	*Transistor-Transistor Logic*
USART	*Universal Synchronous Asynchronous Receiver Transmiter*

1 INTRODUÇÃO

O ser humano é muito frágil. Nos primórdios da civilização humana o homem era nômade e sofria por ter que vencer inúmeras batalhas diárias para se manter vivo. Essas batalhas iam desde a procura do alimento até a busca por refúgio seguro longe de animais e do frio, geralmente em cavernas. A maneira que a raça humana encontrou para sobressair, evoluir e "dominar" o planeta foi a utilização de seu maior e mais poderoso dom, a inteligência. No decorrer da evolução humana, o homem aprendeu técnicas de cultivo de seu próprio alimento. Com isso, a batalha para encontrá-lo acabou e o homem deixou de ser nômade, mas as técnicas de cultivo eram extremamente arcaicas. Cultivar o alimento era um trabalho muito pesado, pois era totalmente manual. O homem, então, percebeu que poderia usar animais para auxiliá-lo em serviços pesados. A domesticação de animais foi um passo muito importante na evolução humana [1], pois a partir deste ponto, animais passaram a ser aliados dos homens. Com isso, a produtividade aumentou e o trabalho ficou mais leve. Desde então, a raça humana não parou de evoluir e de desenvolver técnicas para uma vida mais confortável.

A partir do século XVIII, com a Revolução Industrial, o homem passou a viver uma revolução diária de conhecimento, informação e principalmente tecnologia. Os computadores, a microeletrônica e a telecomunicação juntas, mudam a cada dia o estilo de vida da sociedade global e abrem portas para um futuro ilimitado, com possibilidades além da imaginação. Um bom exemplo disso é a Domótica, fusão da palavra em latim *Domus* que significa casa com robótica, também conhecida por Automação Residencial. A Domótica integra computadores, circuitos eletrônicos e alguma interface de comunicação com o objetivo de controlar, monitorar e automatizar os recursos habitacionais, como por exemplo, iluminação, eletrodomésticos, segurança, climatização e etc. Com a Domótica, processos manuais que são executados diariamente nas casas podem ser automatizados, gerando mais conforto e praticidade no dia a dia. Residências inteligentes e sistemas domóticos têm atraído crescente interesse, uma vez que possibilitam a atuação supervisionada e não supervisionada de dispositivos eletrônicos em uma residência, exercendo tarefas complexas e interagindo com usuários e com o meio físico.

Quem nunca sonhou em ter as facilidades que Os Jetsons (desenho animado criado pela dupla Hanna-Barbera em 1962, mostrando as aventuras de uma família comum em um mundo futurista[3]) possuiam? Apesar de o tema parecer futurista, as primeiras aplicações

práticas da Domótica surgiram nos anos 80, com os edifícios inteligentes que eram capazes de controlar a iluminação, condições climáticas, a segurança e interligar estes três elementos [4].

Segundo Gordon Earl Moore (1965), o poder computacional dobra a cada 24 meses. Essa afirmação se mostra verdadeira até hoje. Afinal desde 1971 com o lançamento do primeiro processador Intel de 4 *bits* (4004) até hoje, os sistemas microprocessados não param de evoluir. Devido a essa rápida evolução, sistemas microprocessados são encontrados em praticamente todos os aparelhos que utilizamos no dia-a-dia e essa popularização torna esses sistemas cada vez mais baratos, afinal quanto maior a produção menor é o seu custo.

A Domótica nada mais é que um sistema microprocessado e como tal está em constante evolução ao ponto em que o limite da automatização é praticamente a imaginação, o maior desafio para que esta tecnologia se popularize e seja utilizada em massa se deve ao seu alto custo. A fim de minimizar este problema, foi proposto neste trabalho o desenvolvimento de um *hardware* simples e o mais barato possível, com as principais funcionalidades dos modelos encontrados no mercado.

2 CONCEITOS

2.1 Microcontroladores PIC

PIC[1] é um circuito altamente integrado produzido pela *Microchip Technology Inc*., que pertence a categoria dos microcontroladores programáveis de arquitetura *Harvard* de conjunto reduzido de instruções (RISC[2])[5]. Externamente o PIC é visto como um circuito integrado normal, mas internamente é um microcomputador completo, ou seja, é constituído de memória RAM[3], memória não-volátil EEPROM[4], memória de programa, controladores de Entrada/Saída digital e analógica (opcional), em torno de uma CPU[5] com um conjunto reduzido de instruções, dentro de um único *chip* [6].

O PIC pode ser programado para executar diversas tarefas, como controlar um dispositivo eletro-mecânico, realizar medições, exibir informações em um *display*, ou simplesmente piscar luzes. A simplicidade, disponibilidade e o baixo custo são os principais atrativos do PIC [5].

2.1.1 PIC16F877A

O microcontrolador PIC16F877A, utilizado neste projeto, é baseado na arquitetura Harvard, o que permite uma alta velocidade de operação. Isso se deve ao fato de que os microcontroladores baseados nesta arquitetura possuem um espaço de memória separado para os dados e outro para as instruções. Isto permite que a leitura das instruções e dos dados ocorra em paralelo. Também, enquanto uma instrução está sendo executada outra já pode estar sendo transferida para a CPU [9] . Esta alta velocidade é importante para a execução de várias tarefas simultaneamente. Além de sua alta velocidade, este dispositivo possui diversas características que o tornam extremamente versátil. A Figura 1 mostra de maneira resumida as principais características do dispositivo.

---

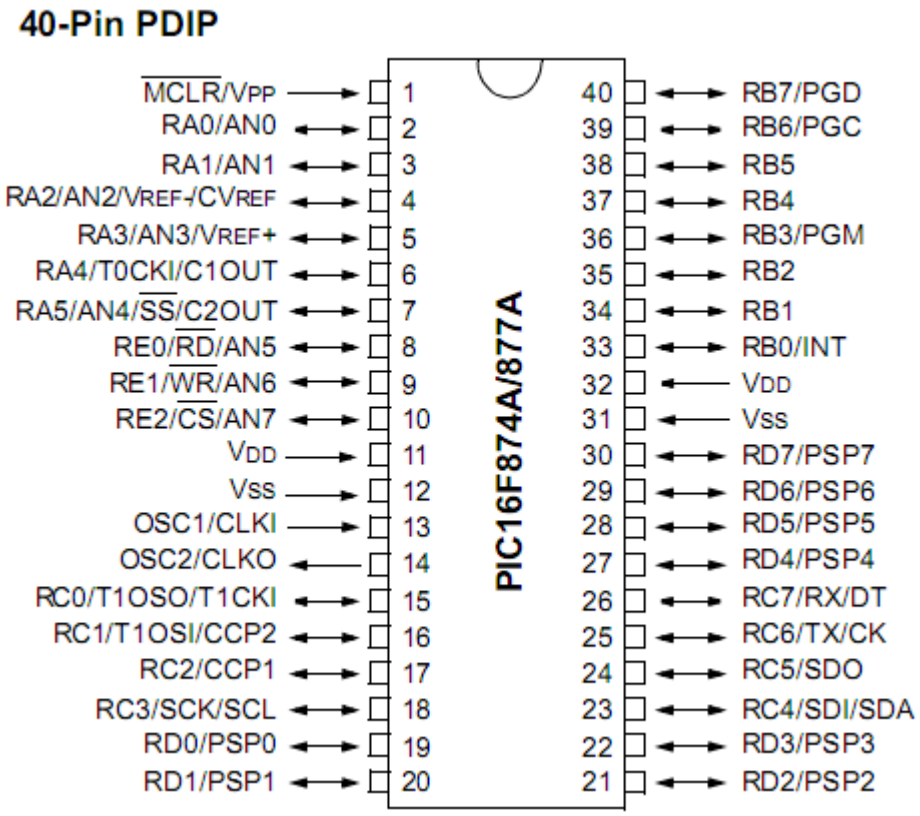
[1] *Programmable Interface Controller*
[2] *Reduced Instruction Set Computer*
[3] *Random Access Memory*
[4] *Electrically-Erasable Programmable Read-Only Memory*
[5] *Central Processing Unit*

Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
Bytes	# Single Word Instructions						SPI	Master I^2C			
14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

Fonte: *Datasheet* PIC16F877A

FIGURA 1 - RESUMO CARACTERÍSTICAS PIC 16F877A

A Figura 2 mostra os 40 pinos do dispositivo com sua respectiva função. A Tabela 1 descreve de maneira sucinta a função de cada um dos pinos do dispositivo.



Fonte: *Datasheet* PIC16F877A

FIGURA 2 - PINOS PIC 16F877A

TABELA 1 - FUNÇÃO DOS PINOS PIC 16F877A

Nome do Pino	Número do Pino	Tipo do pino I/O/P[6]	Descrição
OSC1/CLKIN	13	I	Entrada para cristal externo (fonte de clock)
OSC2/CLKOU	14	O	Saída para cristal externo
MCLR/VPP	1	I/P	*Master Clear* (*reset*) externo. Lógica baixa para *reset*
RA0/AN0	2	I/O	I/O digital bidirecional ou entrada analógica 0
RA1/AN1	3	I/O	I/O digital bidirecional ou entrada analógica 1
RA2/AN2/VREF	4	I/O	I/O digital bidirecional ou entrada analógica 2 ou tensão de referência analógica negativa
RA3/AN3/VREF+	5	I/O	I/O digital bidirecional ou entrada analógica 3 ou tensão de referência analógica positiva
RA4/T0CKI	6	I/O	I/O digital bidirecional ou entrada de *clock* para contador TMR0
RA5/SS/AN4	7	I/O	I/O digital bidirecional ou entrada analógica 4 ou *slave select* para a porta de comunicação serial síncrona
RB0/INT	33	I/O	I/O digital bidirecional ou entrada para interrupção externa
RB1	34	I/O	I/O digital bidirecional
RB2	35	I/O	I/O digital bidirecional
RB3/PGM	36	I/O	I/O digital bidirecional ou entrada para programação em baixa tensão
RB4	37	I/O	I/O digital bidirecional
RB5	38	I/O	I/O digital bidirecional
RB6/PGC	39	I/O	I/O digital bidirecional ou *clock* da programação serial (ICSP[7])
RB7/PGD	40	I/O	I/O digital bidirecional ou dados da programação serial (ICSP)
C0/T1OSO/	15	I/O	I/O digital bidirecional ou saída para cristal externo para TMR1 ou entrada de T1CK1 *clock* para contador TMR1
RC1/T1OSI/	16	I/O	I/O digital bidirecional ou entrada para cristal externo para TMR1 ou I/O para CCP2 *Capture*,*Compare*,PWM 2
RC2/CCP1	17	I/O	I/O digital bidirecional ou I/O para *Capture*,*Compare*,PWM 1
RC3/SCK/SCL	18	I/O	I/O digital bidirecional ou entrada de *clock* serial síncrono ou saída para os modos SPI[8] e I^2C[9]
RC4/SDI/SDA	23	I/O	I/O digital bidirecional ou entrada de dados

[6] I = *Input*, O = *Output*, I/O = *Input*/*Output*, P = *Power*
[7] *In-Circuit Serial Programming*
[8] *Serial Peripheral Interface*
[9] *Inter-Intergrated Circuit*

Nome do Pino	Número do Pino	Tipo do pino I/O/P[6]	Descrição
			SPI ou I/O de dados I^2C
RC5/SDO	24	I/O	I/O digital bidirecional ou saída de dados SPI
RC6/TX//CK	25	I/O	I/O digital bidirecional ou Transmissão para comunicação USART assíncrona ou via de *clock* para comunicação USART síncrona
RC7/RX/DT	26	I/O	I/O digital bidirecional ou Recepção para comunicação USART assíncrona ou via de dados para comunicação USART síncrona
RD0/PSP0	19	I/O	I/O digital bidirecional ou Porta paralela escrava
RD1/PSP1	20	I/O	I/O digital bidirecional ou Porta paralela escrava
RD2/PSP2	21	I/O	I/O digital bidirecional ou Porta paralela escrava
RD3/PSP3	22	I/O	I/O digital bidirecional ou Porta paralela escrava
RD4/PSP4	27	I/O	I/O digital bidirecional ou Porta paralela escrava
RD5/PSP5	28	I/O	I/O digital bidirecional ou Porta paralela escrava
RD6/PSP6	29	I/O	I/O digital bidirecional ou Porta paralela escrava
RD7/PSP7	30	I/O	I/O digital bidirecional ou Porta paralela escrava
RE0/RD/AN5	8	I/O	I/O digital bidirecional ou Controle de Leitura para a Porta paralela escrava ou entrada analógica 5
RE1/WR/AN6	9	I/O	I/O digital bidirecional ou Controle de Escrita para a Porta paralela escrava ou entrada analógica 6
RE2/CS/AN7	10	I/O	I/O digital bidirecional ou *Select Control* para a Porta paralela escrava ou entrada analógica 7
VSS	12 31	P	Referência TERRA (0V)
VDD	11 32	P	Alimentação Positiva (+5V)

Fonte: *Datasheet* PIC16F877A

2.2 **Conversor A/D**

A interação do homem com a natureza se dá por meio de várias grandezas como temperatura, luz, som, pressão e etc. De maneira geral, a intensidade destas grandezas variam de maneira analógica. Uma variação analógica pode ser definida como uma variação contínua em relação ao tempo. Isso significa que, para ir de um valor a outro de intensidade, a grandeza

terá de passar por todos os pontos intermediários de intensidade entre os valores inicial e final [8].

É possível converter uma variação contínua de tensão ou de corrente elétrica em um outro formato. Em que a variação não é contínua, mas discreta, composta normalmente de dois níveis (0 e 1) conhecido por variação digital [8]. O PIC16F877A possui internamente o módulo conversor analógico para digital ou simplesmente conversor A/D, segue a relação de características do módulo:

- Possui 10 bits de resolução, dando um total de 1024 pontos;
- Até oito canais de conversão;
- Quatro tipos de referência: V_{DD} (interna), V_{SS} (interna), V_{REF+} (externa) e V_{REF-} (externa);
- Freqüência de conversão baseada no *clock* da máquina ou em RC dedicado, possibilitando o funcionamento em modo *SLEEP;*
- Três ajustes de freqüência (divisores) para o *clock*;
- Dois tipos de justificação para o resultado da conversão: direita e esquerda;
- Uma interrupção para término da conversão.

Existem diversas maneiras de implementar um conversor A/D. Entretanto foi descrito o funcionamento do sistema de conversão utilizado pelo PIC16F877A, que é o conversor de aproximação sucessiva.

Nesse tipo de conversor, a conversão é realizada do *bit* mais significativo (MSB[10]) para o menos significativo (LSB[11]). Uma vez que o MSB representa a metade da tensão de referencia, conhecer o estado deste *bit* (0 ou 1) já significa saber se a tensão de entrada é maior ou menor que a metade da referência. Conhecido o *bit* mais significativo, passa-se ao próximo *bit*, que representa a metade da metade da tensão de referencia, ou seja, ¼ da tensão de referencia. A conversão segue assim até o LSB.[10]

O módulo conversor A/D interno do PIC utilizado possui quatro registradores associados, que são eles:

[10] *Most significant bit*
[11] *Least significant bit*

- ADCON0 – Registrador de controle 0;
- ADCON1 – Registrador de controle 1;
- ADRESH – Recebe a parte alta da amostra convertida;
- ADRESL – Recebe a parte baixa da amostra convertida.

A função de cada *bit* na do registrador ADCON0 será mostrada a seguir na Tabela 2.

TABELA 2 – DESCRIÇÃO DOS *BITS* DO REGISTRADOR ADCON0

Registrador ADCON0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCS1	**ADCS0**	**CHS2**	**CHS1**	**CHS0**	**GO/$\overline{\text{DONE}}$**	**-**	**ADON**

Fonte: *Datasheet* PIC16F877A

Bit 7 - 6: **ADSC1:ADSC0** *Bits* de seleção de *clock* do conversor A/D. A Tabela 3 mostra as opções de configurações do *clock* do conversor.

TABELA 3 - CONFIGURAÇÃO DE VALOR DO *CLOCK* PARA O CONVERSOR A/D

ADCON1 <ADSCS2>	**ADCON0 <ADCS1:ADCS0>**	**Clock do conversor**
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derivado de um oscilador interno RC)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derivado de um oscilador interno RC)

Fonte: *Datasheet* PIC16F877A

Bit 5 – 3: **CHS2:CHS0** *Bit* de seleção do canal analógico.

000 = Canal 0 (RA0/AN0)
001 = Canal 1 (RA1/AN1)
010 = Canal 2 (RA2/AN2)
011 = Canal 3 (RA3/AN3)
100 = Canal 4 (RA4/AN4)
101 = Canal 5 (RE0/AN5)
110 = Canal 6 (RE1/AN6)
111 = Canal 7 (RE2/AN7)

Bit 2: **GO/$\overline{\textbf{DONE}}$**: Inicia a conversão A/D.

1 = Conversão A/D em progresso (setando esse *bit*, tem inicio a conversão).

0 = Zerado automaticamente pelo hardware quando a conversão terminar.

Bit 1: Não implementado. Lido como 0.

Bit 0: **ADON** Liga/Desliga o conversor A/D.

1 = Conversor A/D está em operação.

0 = Conversor A/D desligado.

A função de cada *bit* na do registrador ADCON1 será mostrada a seguir na Tabela 4.

TABELA 4 - DESCRIÇÃO DOS *BITS* DO REGISTRADOR ADCON1

Registrador ADCON1							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADFM	**ADCS2**	**-**	**-**	**PCFG3**	**PCFG2**	**PCFG1**	**PCFG0**

Fonte: *Datasheet* PIC16F877A

Bit 7: **ADFM** *bit* de seleção do formato do resultado da conversão.

1 = Justificado à direita. Os seis *bits* mais significativos de ADRESH são lidos como 0.

0 = Justificado à esquerda. Os seis *bits* menos significativos de ADRESL são lidos como 0.

Bit 6: **ADSC2** *bit* de seleção de *clock* do conversor A/D. Ver Tabela 3.

Bit 5 – 4: Não implementados. Lidos como 0.

Bit 3 – 0: **PCFG3:PCFG0**: Configura o modo de funcionamento do conversor A/D, como mostra a Tabela 5.

TABELA 5 - CONFIGURAÇÃO DE FUNCIONAMENTO DO CONVERSOR A/D

PCFG3:PCFG0	RE2/AN7	RE1/AN6	RE0/AN5	RA4/AN4	RA3/AN3	RA2/AN2	RA1/AN1	RA0/AN0	VREF+	VREF-	Canais analógicos
0000	A	A	A	A	A	A	A	A	VDD	VSS	8
0001	A	A	A	A	VREF-	A	A	A	AN3	VSS	7
0010	D	D	D	A	A	A	A	A	VDD	VSS	5
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4
0100	D	D	D	D	A	D	A	A	VDD	VSS	3
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2
011x	D	D	D	D	D	D	D	D	-	-	0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6
1001	D	D	A	A	A	A	A	A	VDD	VSS	6
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1

A = Entrada analógica; D = Pino I/O; X = Irrelevante, pode ser 1 ou 0.

Fonte: *Datasheet* PIC16F877A

2.3 **Comunicação Serial**

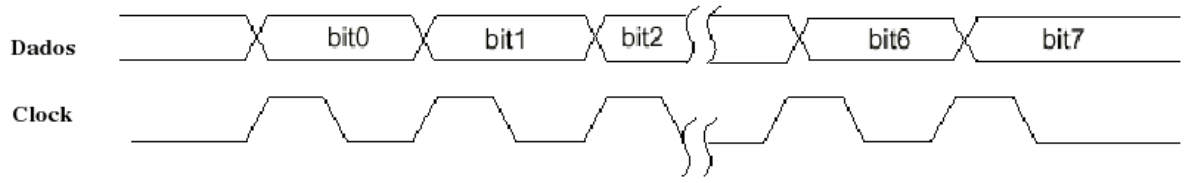
A maioria das mensagens digitais possuem mais que alguns *bits*. Por não ser prático nem econômico transferir todos os *bits* de uma mensagem simultaneamente, a mensagem é quebrada em partes menores e transmitida seqüencialmente [7]. A transmissão serial envia ao canal de comunicação um "trem de pulsos", ou seja, a mensagem original é quebrada em *bits* e estes são transmitidos um a um pelo canal de comunicação. Os *bits*, ao chegarem ao destino, são rearranjados para compor a mensagem original.

A transmissão serial foi escolhida neste projeto como a interface de comunicação entre a placa micro-controlada e o microcomputador, pois é um método de comunicação muito difundido e relativamente simples de se implementar, tendo em vista que o PIC utilizado possui um módulo interno dedicado a comunicação serial o USART[12].

[12] *Universal Synchronous Asynchronous Receiver Transmiter*

2.3.1 Transmissão Síncrona X Assíncrona

Geralmente, dados serializados não são enviados de maneira uniforme através de um canal. Ao invés disso, pacotes com informações regulares são enviados seguidos de pausa. Os pacotes de dados binários são enviados dessa maneira, com comprimentos de pausa variável entre pacotes, até que a mensagem tenha sido totalmente transmitida. O circuito receptor dos dados deve saber o momento apropriado para ler os *bits* individuais deste canal, saber exatamente quando um pacote começa e quanto tempo decorre entre *bits*. Quando essa temporização for conhecida, o receptor é dito sincronizado com o transmissor. Falhas na manutenção do sincronismo durante a transmissão irão causar a corrupção ou perda dos dados. Em sistemas síncronos são utilizados dois canais, como mostra a Figura 3, um para os dados e outro para informações de tempo. O canal de temporização transmite pulsos de *clock* para o receptor. Através da recepção de um pulso de *clock*, o receptor lê o canal de dado. O canal de dados não será lido novamente até que o próximo pulso de *clock* chegue. Como o transmissor é responsável pelos pulsos de dados e de temporização, o receptor irá ler o canal de dados apenas quando comandado pelo transmissor, garantindo a sincronização [7].

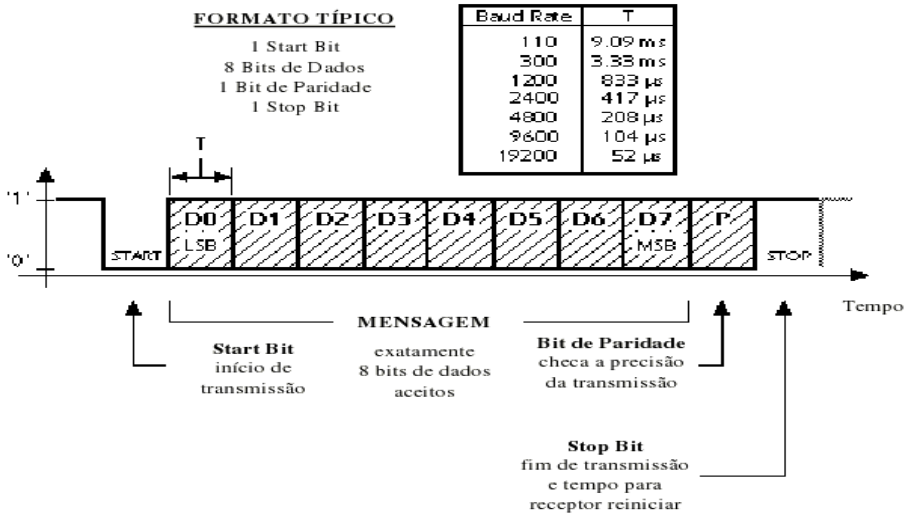


Fonte: Zanco (2006)

FIGURA 3 - TRANSMISSÃO SÍNCRONA

Em sistemas assíncronos, a informação trafega por um único canal. O transmissor e o receptor devem ser configurados antecipadamente para que a comunicação se estabeleça corretamente. Um oscilador preciso no receptor irá gerar um sinal de *clock* interno que é igual (ou muito próximo) ao do transmissor. Para o protocolo serial mais comum, os dados são enviados em pequenos pacotes de 10 ou 11 *bits*, dos quais 8 constituem a mensagem. Quando o canal está em repouso, o sinal correspondente no canal tem nível lógico ‘1’. Um pacote de dados sempre começa com nível lógico ‘0’ (*start bit*) para sinalizar ao receptor que uma transmissão foi iniciada. O *start bit* inicializa um temporizador interno no receptor avisando que a transmissão começou e que serão necessários pulsos de *clock*. Seguido do *start bit* 8 *bits*

de dados de mensagem são enviados na taxa de transmissão especificada. O pacote é concluído com os *bits* de paridade e de parada (*stop bit*). O processo pode ser visto na Figura 4.



Fonte: Zanco (2006)

FIGURA 4 - TRANSMISSÃO ASSÍNCRONA

O comprimento do pacote de dados é pequeno em sistemas assíncronos para minimizar o risco do oscilador do transmissor e do receptor variar. Quando osciladores a cristal são utilizados, a sincronização pode ser garantida sobre os 11 *bits* de período [7].

O *bit* de paridade é utilizado para detecção de erros de transmissão. A paridade pode ser par, ímpar, marca (*Mark*) ou espaço [8].

- **Paridade par:** a soma dos *bits* de dados iguais a 1 mais o *bit* de paridade tem que ser par;
- **Paridade ímpar:** a soma dos *bits* de dados iguais a 1 mais o *bit* de paridade tem que ser ímpar;
- **Marca:** o *bit* de paridade é sempre 1;
- **Espaço:** o *bit* de paridade é sempre 0.

O *bit* de paridade caiu no desuso. Hoje existem maneiras mais eficientes de detecção de erros. Por este motivo o seu uso é opcional devendo ser configurado a inclusão ou não deste recurso assim como o tamanho do *stop bit*. Ambos os *bits* devem ser configurados com os mesmos valores tanto para o transmissor quanto no receptor.

2.3.2 Módulo USART

Alguns microcontroladores possuem internamente um módulo de transmissão e recepção de dados serial a USART. Embora a USART possa transmitir e receber dados tanto de forma assíncrona quanto de maneira síncrona, neste projeto foi utilizado somente o modo de operação assíncrono. Isso se deve ao fato de que não existe nenhum sincronismo entre a placa microcontrolada e o microcomputador.

A USART é um protocolo de comunicação serial que foi desenvolvido para permitir a comunicação entre *mainframes* e computadores terminais remotos. A USART tornou-se tão popular que é amplamente utilizada até hoje [8].

Embora que apenas uma única linha de transmissão seja suficiente para implementar uma comunicação serial assíncrona, é muito mais comum a utilização de um canal de transmissão e outro para recepção. Esses canais são denominados de TX e RX, respectivamente, transmissão e recepção de dados. A utilização de canais diferentes para transmissão e recepção permite que os dados sejam transmitidos e recebidos simultaneamente, ou seja, comunicação *full-duplex* [8].

O PIC utilizado neste projeto é o PIC16F877A, este possui o módulo de comunicação USART interno e a partir deste ponto será detalhada a configuração de cada *bit* de configuração do módulo.

O TXSTA é o registrador de controle do módulo transmissor da USART. A função de cada *bit* na transmissão assíncrona será mostrada a seguir na Tabela 6.

TABELA 6 - DESCRIÇÃO DE BITS DO REGISTRADOR TXSTA

Registrador TXSTA							
Bit 7	*Bit* 6	*Bit* 5	*Bit* 4	*Bit* 3	*Bit* 2	*Bit* 1	*Bit* 0
CSRC	**TX9**	**TXEN**	**SYNC**	**-**	**BRGH**	**TRMT**	**TX9D**

Fonte: *Datasheet* PIC16F877A

Bit 7: **CSRC:** *Bit* de seleção da origem do *clock* utilizado na transmissão. (Na trasnmissão assíncrona esse *bit* não tem função.

Bit 6: **TX9:** Habilita a transmissão do 9º *bit*

1 = Transmissão terá 9 *bits*

0 = Transmissão terá 8 *bits*

Bit 5: **TXEN:** Habilita a transmissão

1 = Transmissão habilitada

0 = Transmissão desabilitada

Bit 4: **SYNC:** Seleciona o modo de transmissão

1 = Transmissão síncrona

0 = Transmissão assíncrona

Bit 3: **Não implementado:** lido como 0

Bit 2: **BRGH:** Seleciona modo de transmissão

Na transmissão assíncrona:

1 = Alta velocidade

0 = Baixa velocidade

Bit 1: **TRMT:** Status do registrador de deslocamento utilizado na transmissão (TSR)

1 = TSR vazio

0 = TSR cheio

Bit 0: **TX9D:** Nono *bit* de dados. Pode ser utilizado como *bit* de paridade.

O RCSTA é o registrador de controle do módulo receptor da USART. A função de cada *bit* na transmissão assíncrona será mostrada a seguir na Tabela 7.

TABELA 7 - DESCRIÇÃO DE BITS DO REGISTRADOR RCSTA

Registrador RCSTA							
Bit 7	*Bit* 6	*Bit* 5	*Bit* 4	*Bit* 3	*Bit* 2	*Bit* 1	*Bit* 0
SPEN	**RX9**	**SREN**	**CREN**	**ADDEN**	**FERR**	**OERR**	**RX9D**

Fonte: *Datasheet* PIC16F877A

Bit 7: **SPEN:** Habilita a porta serial (configura RC7/RX/DT e RC6/TX/CK como RX e TX, respectivamente, fazendo-se os *bits* TRISC<7:6> = 1)

1 = Habilita a porta serial

0 = Desabilita a porta serial

Bit 6: **RX9:** Habilita a recepção do 9º *bit*

1 = Habilita a recepção de 9 *bits*

0 = Habilita a recepção de 8 *bits*

Bit 5: **SREN:** Habilita a recepção única (Na transmissão assíncrona esse *bit* não tem função)

Bit 4: **CREN:** Habilita a recepção contínua

Na transmissão assíncrona:

1 = Habilita recepção contínua

0 = Desabilita recepção contínua

Bit 3: **ADDEN:** Habilita a detecção de endereço

Na transmissão assíncrona modo 9-*bit* (RX9 = 1):

1 = Habilita detecção de endereço (só carrega o dado recebido no registrador RCREG se o nono *bit* for igual a 1, ou seja, byte recebido é um endereço)

0 = Desabilita detecção de endereço (todos os bytes são recebidos e o nono *bit* pode ser utilizado como paridade)

Na transmissão assíncrona modo 8-*bit* (RX9 = 0):

Não utilizado neste modo

Bit 2: **FERR:** Indica erro de *Frame*

1 = Erro de Frame, ou seja, o stop *bit* foi lido como 0

0 = Não ocorreu erro de *Frame*

Bit 1: **OERR:** *Bit* de erro de transbordo

1 = Houve transbordo (pode ser resetado apagando o *bit* CREN)

0 = Não houve transbordo

Bit 0: **RX9D:** Nono *bit* de dados.

Baud rate é a quantidade de *bits* que pode ser transmitida por segundo (bps) [8]. O PIC16F877A possui um circuito gerador de *baud rate* (BRG), cujo valor para a transmissão assíncrona é calculado a partir das fórmulas da Tabela 8, que foram retiradas do *datasheet* deste PIC. O valor do *baud rate* depende da freqüência do oscilador (Fosc), do *bit* BRGH e do valor armazenado no registrador SPBRG.

TABELA 8 - FÓRMULAS PARA CALCULAR O *BAUD RATE*

BRGH = 0 (baixa velocidade)	**BRGH = 1 (alta velocidade)**
Baud Rate = Fosc/[64x(SPBRG+1)]	Baud Rate = Fosc/[16x(SPBRG+1)]

Fonte: *Datasheet* PIC16F877A

Baseado nestas fórmulas, foi calculado o valor que escrito no registrador SPBRG para uma transmissão assíncrona, com *baud rate* de 9600 bps com Fosc de 4MHz.

Isolando SPBRG na fórmula obtém-se:

SPBRG = [Fosc/(Y x Baud Rate)] -1

Y = 64 para BRGH = 0

Y = 16 para BRGH = 1

Assim:

Se BRGH = 0	Se BRGH = 1
SPBRG = [4 x 10^6 / (64 x 9600)] -1	SPBRG = [4 x 10^6 / (16 x 9600)] -1
SPBRG = 5,51	SPBRG = 25,04

Ambos os resultados não foram inteiros por este motivo será aproveitado para os cálculos apenas a parte inteira dos números. Ou seja, para:

BRGH = 0 → SPBRG = 5

BRGH = 1 → SPBRG = 25

Se BRGH = 0	Se BRGH = 1
Baud Rate = 4 x 10^6 / [64x(5+1)]	*Baud Rate* = 4 x 10^6 / [16x(25+1)]
Baud Rate = 10416,66bps	SPBRG = 9615,38bps

Esta aproximação fará com que o *baud rate* não seja exatamente 9600bps. Essa diferença é chamada de Erro de *Baud Rate* [8]. O erro será calculado em porcentagem.

Para BRGH = 0:

Erro de *Baud Rate* = [(10416,66 – 9600) / 9600] x 100

Erro de *Baud Rate* = 8,5 %

Para BRGH = 1:

Erro de *Baud Rate* = [(9615,38 – 9600) / 9600] x 100

Erro de *Baud Rate* = 0,16 %

Os cálculos mostram que o Erro de *Baud Rate* é muito menor quando BRGH = 1. Por este motivo, neste projeto a comunicação foi feita de modo assíncrono com taxa de transmissão de 9600bps no modo alta velocidade, ou seja, BRGH = 1.

2.3.3 RS232

O RS-232 é um padrão que cria uma interface comum para comunicação de dados entre equipamentos. Foi criada no início dos anos 60, por um comitê conhecido atualmente como EIA[13] [7].

O padrão RS-232 define características mecânicas, elétricas, e funcionais para a comunicação serial de dados entre DTE[14] e um DCE[15] [8].

Este projeto não foca o padrão RS-232, por este motivo só será relatado o que é necessário para a comunicação entre o microcontrolador e o PC.

A maioria dos computadores atuais possui pelo menos uma porta de comunicação serial, mais conhecida por porta COM. Um microcontrolador pode se comunicar, ou seja, trocar informações com o PC utilizando a porta COM do microcomputador. A comunicação é feita pelo protocolo USART quando definimos os parâmetros da comunicação. São eles:

- *Baud Rate*, ou taxa de transferência (bps);
- Número de *bits* de dados;
- *Bit* de paridade;
- Tamanho do stop *bit*;
- Controle de fluxo

A Tabela 9 a seguir mostra os parâmetros mínimos e máximos de uma comunicação serial assíncrona que podem ser configurados tanto para o PIC16F877A quanto para a porta COM do PC.

TABELA 9 - PARAMETROS DA COMUNICAÇÃO SERIAL PIC - PC

	PIC16F877A	**Porta COM do PC**
Taxa de transferência	110 a 1.250.000 bps	110 a 921.600 bps
Número de *bits*	8 ou 9	4 a 8
***Bit* de Paridade**	Não suporta	Par, ímpar, marca ou espaço
Stop *Bit*	1	1; 1,5 ou 2
Controle de fluxo	Não suporta	Hardware, XON/XOFF ou nenhum

Fonte: Zanco (2006)

[13] *Electronic Industries Association*
[14] Equipamento de Dados Terminal
[15] Equipamento de Comunicação de Dados

2.4 **PWM**

O termo PWM *Pulse Width Modulation*, modulação por largura de pulso em português, trata-se de uma onda com freqüência constante (período fixo) e largura de pulso (*duty cycle*) variável.[10]

A Figura 5 apresenta algumas ondas tipo PWM, cada uma com uma largura de pulso diferente. Os gráficos mostram que o período (T) é sempre o mesmo, apenas a largura do pulso é variável, a tensão média de um sinal PWM é diretamente proporcional a largura do pulso.

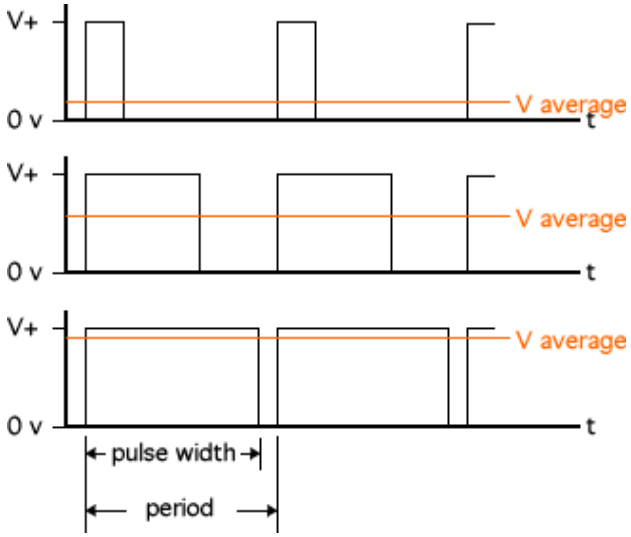


FIGURA 5 - EXEMPLO SINAL PWM

O PIC16F877A possui dois canais de PWM, cada um com resolução máxima de dez bits. Isso significa que o *duty cycle* pode ser regulado de 0 a 100% com uma resolução máxima de 1024 pontos, dependendo apenas da configuração adotada.

O período (T) do PWM é controlado pelo *Timer* 2, através do registrador PR2. Sempre que TMR2 = PR2, o *timer* é zerado. Neste momento um novo período do PWM é iniciado. Desta forma é possível definir o período e a freqüência do PWM pelas seguintes fórmulas:

$$T = [(PR2) + 1] \times 4 \times T_{OSC} \times (\textit{Prescale} \text{ do TMR2})$$

$$PWM_{Freq} = 1/T$$

No PIC não é possível definir um valor para o *duty* cycle, entretanto pode-se determinar o tempo que o pulso fica em nível alto. O tempo do pulso pode ser calculado pela fórmula seguinte:

$$Tp = (CCPR1L:CCP1CON<5:4>) \times T_{OSC} \times (\textit{Prescale} \text{ do TMR2})$$

A largura do pulso é ajustada em dois registradores: CCPRxL, que armazena os 8 *bits* mais significativos, e CCPxCON, que armazena os dois *bits* menos significativos. Assim, temos os 10 *bits* que controlam o *duty cycle* do PWM.

Para calcular efetivamente o *duty cycle* é necessário dividir o tempo do pulso em nível alto pelo período total do PWM.

2.5 **MAX 232**

MAX232 é um circuito integrado que permite um microcontrolador, que na maioria dos casos utiliza a lógica TTL[16] na representação dos níveis lógicos 0 e 1, interfacear com o padrão RS-232 [8]. O que esse circuito integrado faz na verdade é converter os níveis de tensão correspondentes à lógica TTL no padrão RS-232 e vice-versa. A Tabela 10 apresenta os níveis de tensão correspondentes aos níveis lógicos 0 e 1 que trafegam no padrão RS-232 e os níveis lógicos correspondentes fornecidos pelo PIC16F877A para um VDD = 5V.

TABELA 10 - CORRESPONDÊNCIA DE TENSÃO ENTRE TTL E RS-232

Nível Lógico	PIC16F877A	Padrão RS-232
0	≤ 0,6 V	+5 V a +15 V
1	≥ 5 - 0,6 V	-5 V a -15 V

Fonte: Zanco (2006)

2.6 **LM35**

O LM35 é um de sensor de temperatura de precisão baseado na tecnologia SBCI[17], fabricado pela *National Semicondutor*. Este dispositivo fornece na sua saída, uma tensão linearmente proporcional à temperatura em graus Celsius, à qual está submetido, 10mV de

[16] *Transistor-Transistor Logic*
[17] Sensor Baseado em Circuito Integrado

tensão de saída (Vout) para cada grau de temperatura. [11]. O LM35 não requer calibração externa e pode operar na faixa de temperatura de -55 a 150 ºC com uma precisão de até ¼ ºC. Além de todas as características apresentadas, este sensor possui um baixo valor, por todos estes motivos, o sensor de temperatura utilizado no projeto foi o LM35.

A Figura 6 mostra a disposição dos pinos do LM35 no encapsulamento TO-92.

Fonte: *Datasheet* LM35

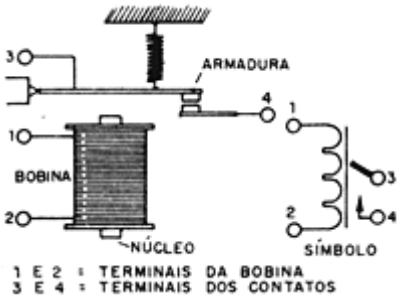
FIGURA 6 - PINOS LM35

2.7 **Saídas de potência**

Todo pino de I/O do PIC16F877A suporta no máximo uma corrente de 25 mA, para não danificar o microcontrolador não pode ser ligado diretamente aos seus terminais cargas que consumam o valor de corrente maior do que o suportado pelo dispositivo. Este valor é suficiente para acender um LED, por exemplo. Quando há necessidade de controlar / acionar cargas mais robustas, ou seja, que exigem uma corrente maior para o seu funcionamento, é necessário a utilização de um circuito de acionamento. Este circuito serve como uma ponte que liga a carga ao PIC. Basicamente o PIC controla envia o sinal de controle, baixa corrente, para o circuito de acionamento e este aciona a carga liberando tensão e corrente para a mesma.

2.7.1 Saída a Relé

Os relés são dispositivos comutadores eletromecânicos. A estrutura simplificada de um relé é mostrada na Figura 7.



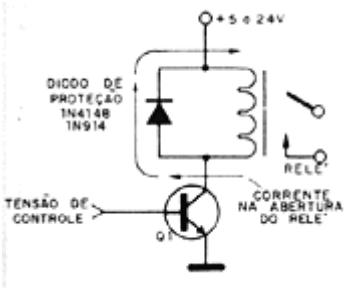
Fonte: *site* Metaltex

FIGURA 7 - ESTRUTURA SIMPLIFICADA DE UM RELÉ

Nas proximidades de um eletroímã é instalada uma armadura móvel que tem por finalidade abrir ou fechar um jogo de contatos. Quando a bobina é percorrida por uma corrente elétrica, é criado um campo magnético que atua sobre a armadura, atraindo-a. Nesta atração ocorre um movimento que ativa os contatos, os quais podem ser abertos, fechados ou comutados, dependendo de sua posição.[12] Isso significa que, através de uma corrente de controle aplicada à bobina de um relé, é possível abrir, fechar ou comutar os contatos de uma determinada forma, controlando assim as correntes que circulam por circuitos externos. Quando a corrente deixa de circular pela bobina do relé, o campo magnético criado desaparece, e com isso a armadura volta a sua posição inicial pela ação da mola.

O acionamento a relé, também conhecido por contato seco, é utilizado para o acionamento de cargas que não precisem de um acionamento rápido e/ou com grandes freqüências de acionamento, isso se deve ao fato de que o relé é um dispositivo eletromecânico por isso é lento e com acionamentos sucessivos tem sua vida útil diminuída consideravelmente. Este tipo de acionamento é muito utilizado na automação residencial para acionar lâmpadas e eletrodomésticos.

O esquema elétrico da Figura 8 mostra o circuito necessário para a utilização de uma saída a relé.



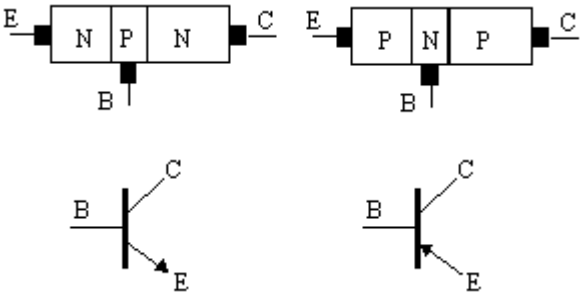
Fonte: *site* Metaltex

FIGURA 8 - CIRCUITO DE ACIONAMENTO A RELÉ

Nota-se que há um diodo em paralelo com a bobina do relé isto se mostra necessário, pois no momento em que um relé é desenergizado, há uma indução de uma tensão inversa que pode atingir valores muito altos. Se o componente que faz o acionamento do relé não estiver dimensionado para suportar esta tensão, se não houver uma proteção adequada, sua queima será inevitável.[12] O que ocorre neste caso é que o diodo está polarizado inversamente em relação a tensão que dispara o relé. Assim, quando ocorre a indução de uma alta tensão nos extremos da bobina no momento da interrupção da corrente, o diodo polarizado no sentido direto passa a ter uma baixa resistência absorvendo assim a energia que, de outra forma, poderia afetar o componente de disparo. O componente de disparo é um transistor bipolar de pouco ganho, componente extremamente barato, o que torna o acionamento a relé mais barato que se comparado ao acionamento a transistor.

2.7.2 Saída a Transistor

O princípio de funcionamento do transistor é poder controlar a corrente. Ele é montado numa estrutura de cristais semicondutores (P e N), de modo a formar duas camadas de cristais do mesmo tipo, intercaladas por uma camada de cristal do tipo oposto, que controla a passagem de corrente entre as outras duas. As extremidades são chamadas de emissor e coletor, e a camada central é chamada de base.[13]



Fonte: http://www.arvm.org/exames/trasistor.htm

FIGURA 9 - ESTRUTURA INTERNA E SÍMBOLOS DE TRANSISTOR BIPOLAR

O comportamento básico dos transistores em circuitos eletrônicos é fazer o controle da passagem de corrente entre o emissor e o coletor através da base. Quanto maior a corrente da base, maior será a corrente de emissor – coletor.

Acionamentos a transistor geralmente são utilizados em automação residencial para controlar a rotação de motores e a intensidade de iluminação, isto porque essas funções são controladas por PWM, ou seja, o sinal que chega a carga é uma onda quadrada com largura de pulso variável com uma freqüência que um relé não conseguiria chavear. Logo conclui-se que o acionamento a transistor é mais eficiente do que o a relé, afinal transistores são mais rápidos, seguros e confiáveis do que relé, entretanto para cargas de alta potência os transistores que devem ser utilizados se tornam muito caros.

3 MATERIAIS UTILIZADOS

3.1 *Hardware*

A Tabela 11 contém a lista de componentes usados para a elaboração do projeto.

TABELA 11 - LISTA DE COMPONENTES

Quantidade	Componente
1	PIC16F877A
1	Cristal de 4MHz
1	MAX232
1	Conector DB9 fêmea
1	Cabo DB9 null-modem macho-fêmea
6	Capacitor eletrolítico 1uF x 50V
2	Capacitor cerâmico 22pF
2	Transistor bipolar BC337
1	LM35
1	Rele 5V-10A
1	LED vermelho 3mm
1	LED azul 3mm
1	LED branco 1W
2	Resistor 330Ω ¼W
1	Resistor 4K7Ω ¼W
1	Resistor 10KΩ ¼W
1	Resistor 1K5Ω ¼W
1	Resistor 6,8Ω 1W
1	Placa de fenolite

3.2 *Software*

Segue uma lista contendo todos os *softwares* utilizados como base ou suporte para o desenvolvimento do projeto.

- **Proteus**: desenvolvido pela *Labcenter Electronics* é uma suíte que agrega um ambiente de simulação de circuitos eletrônicos (**ISIS**) e um programa para desenho de circuito impresso (**Ares Professional**);

- **MPLAB*:*** *d*esenvolvido pela *Microchip* é uma IDE [18] de desenvolvimento para a linha de produtos comercializados pela Microchip;
- **mikroC for PIC*:*** mikroElektronika é uma IDE de desenvolvimento e compilador da linguagem ANSI C para microcontroladores PIC;
- **Serial Monitor*:*** desenvolvido pela *HHD Software* é um analisador de comunicação serial, *sniffer*.

[18] *Integrated Development Environment*

4 PROJETO

4.1 Descrição do *Hardware*

O *hardware* é baseado no microcontrolador PIC 16F877A, responsável por processar as informações provenientes da interface de comunicação e repassar a ações necessárias aos dispositivos dentro da casa.

A Figura 10 mostra o diagrama em blocos do *hardware* elaborado neste projeto.

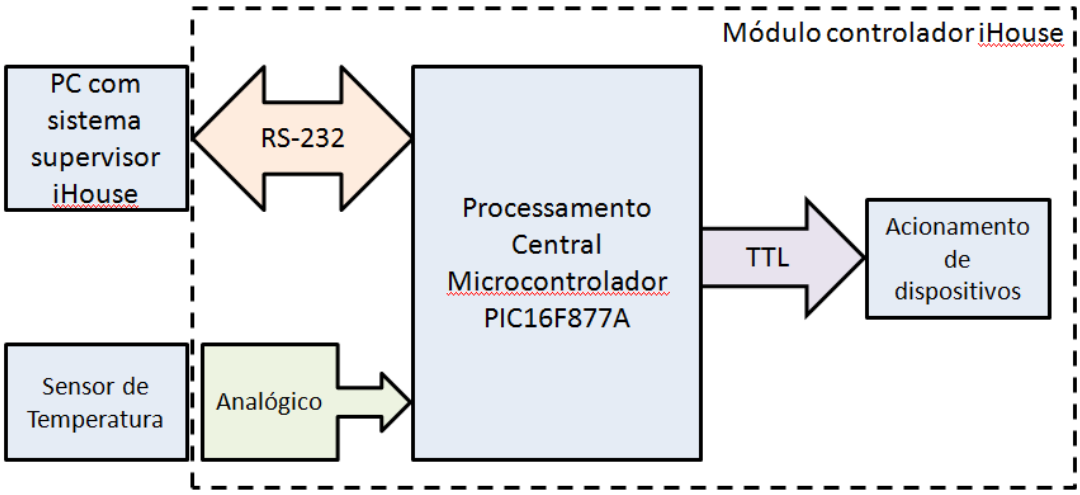


FIGURA 10 - DIAGRAMA EM BLOCOS DA PLACA

A Interface de comunicação é do tipo serial RS232. A mesma é ligada diretamente a porta serial do PC, através de um cabo *null-modem* (Figura 11 - Esquema de conexão entre PC e placa) e ao barramento de comunicação do PIC tendo como intermediário o MAX232, que é um circuito integrado responsável pela adequação de sinais para uma comunicação efetiva entre PC-PIC. O protocolo de comunicação utilizado foi desenvolvido pelo acadêmico Guilherme da Silva Mello e será descrito com maiores detalhes posteriormente.

Para a elaboração deste projeto a comunicação foi configurada como mostra a Tabela 12.

TABELA 12 - CONFIGURAÇÃO DE COMUNICAÇÃO SERIAL UTILIZADA

Taxa de transferência	9600 bps
Número de *bits*	8
***Bit* de Paridade**	-
Stop *Bit*	1
Controle de fluxo	-

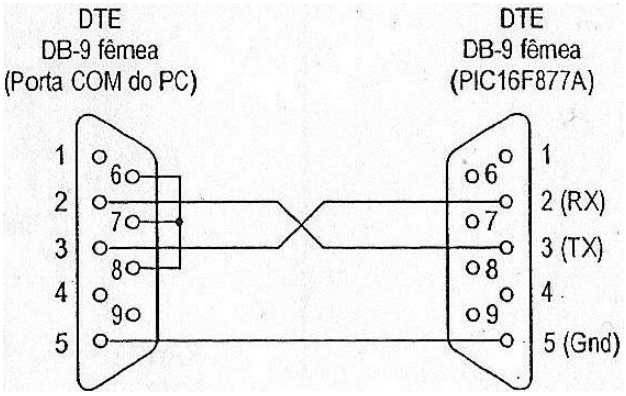
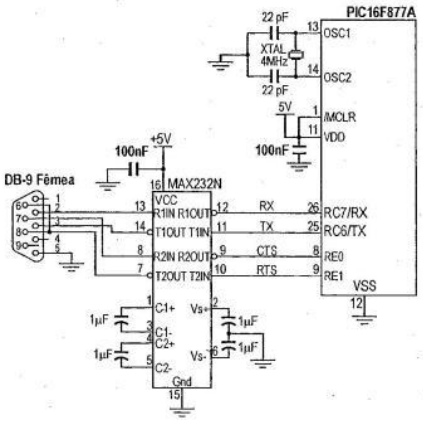


FIGURA 11 - ESQUEMA DE CONEXÃO ENTRE PC E PLACA

Fonte: Zanco (2006)

O interfaceamento entre o MAX232 e o PIC é feito de maneira muito simples, necessitando apenas de quatro capacitores de 1µF conectados a alguns de seus pinos. A Figura 12 mostra o esquema de ligação entre o PIC e o microcontrolador.



Fonte: Zanco (2006)

FIGURA 12 - INTERFACEAMENTO ENTRE PIC E O MAX232

Os dispositivos controlados neste projeto são:

- Uma lâmpada incandescente 60 W 127 V acionada através de uma saída a relé;
- Um LED de potência, com a possibilidade de controlar sua luminosidade através de PWM, acionado através de uma saída a transistor;
- Dois LEDS (azul e vermelho) que simulam o acionamento de um sistema de ar-condicionado. O LED azul será acesso quando a temperatura medida for maior que a configurada pelo usuário e o vermelho será acesso na situação contrária, ou seja, quando a temperatura medida for menor que a configurada pelo usuário.

  O sensor LM35 foi utilizado para medir efetivamente a temperatura, ele alimenta o módulo conversor A/D do PIC16F877A que realiza a conversão e processamento dos dados e com isso aciona ou não o sistema de ar-condicionado.

A Figura 13, a seguir, mostra o diagrama elétrico da placa, ou seja, a representação gráfica das conexões físicas entre os componentes.

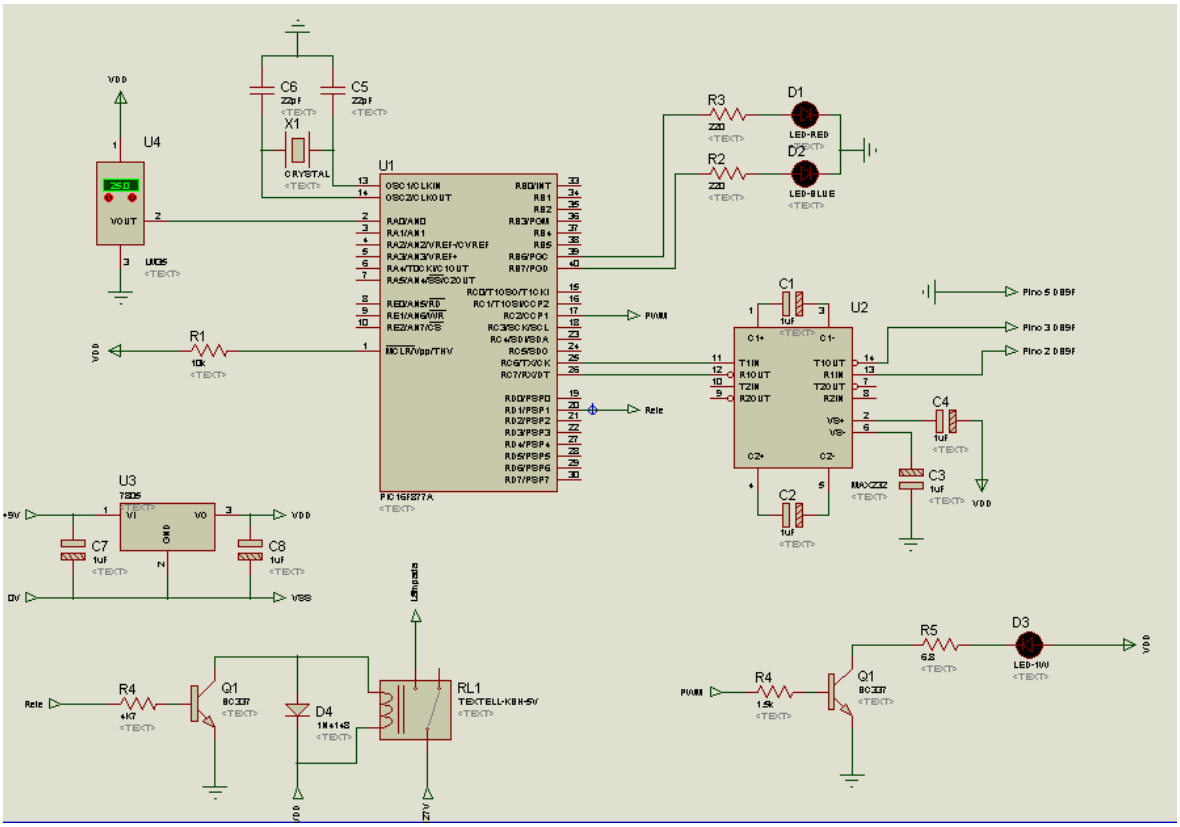


FIGURA 13 - DIAGRAMA ELÉTRICO DA PLACA

4.2 **Confecção da PCB**

Os circuitos impressos foram criados em substituição às antigas pontes onde se fixavam os componentes eletrônicos, em montagem conhecida no jargão de eletrônica como montagem "aranha", devido à aparência final que o circuito tomava, principalmente onde existiam válvulas eletrônicas e seus múltiplos pinos terminais do soquete de fixação.

O circuito impresso consiste de uma placa de fenolite, fibra de vidro, fibra de poliéster, filme de poliéster, filmes específicos à base de diversos polímeros, etc, que possuem a superfície coberta numa ou nas duas faces por fina pelicula de cobre, prata, ou ligas à base de ouro, níquel entre outras, nas quais são desenhadas pistas condutoras que representam o circuito onde serão fixados os componentes eletrônicos.

4.2.1 ARES FOTOLITE

ARES Fotolite é um *software* integrante da suíte Proteus desenvolvido pela empresa *Labcenter Electronics Ltd*, com este *software* foi elaborado o desenho da PCB deste projeto.

O *software* possui uma vasta biblioteca de componentes, uma interface relativamente simples e é uma ferramenta poderosa, com ela é possível rotear placas com até 16 camadas de trilhas. Para o projeto foi utilizado apenas uma camada.

A diagramação do circuito é bem simples e intuitiva, basta procurar os componentes desejados na biblioteca posicioná-los e efetuar as ligações elétricas, após isso clica-se no botão *auto-router* e o *software* automaticamente roteia o circuito da melhor maneira possível.

A Figura 14 mostra o desenho final do circuito que será impresso na placa. O *ARES Fotolite* permite que seja criado um modelo 3D da placa, essa função é útil para que se possa ter uma idéia da disposição física de todos os componentes desta maneira é possível evitar problemas de componentes sobrepostos. A Figura 15 mostra o modelo 3D gerado pelo *software*.

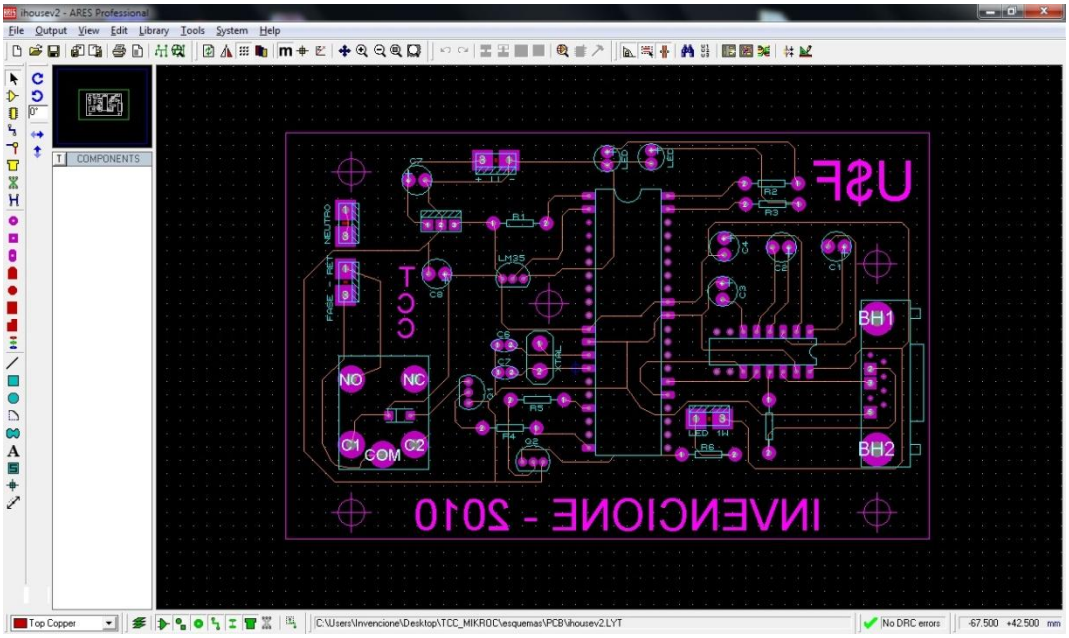


FIGURA 14 - DESENHO DA PLACA

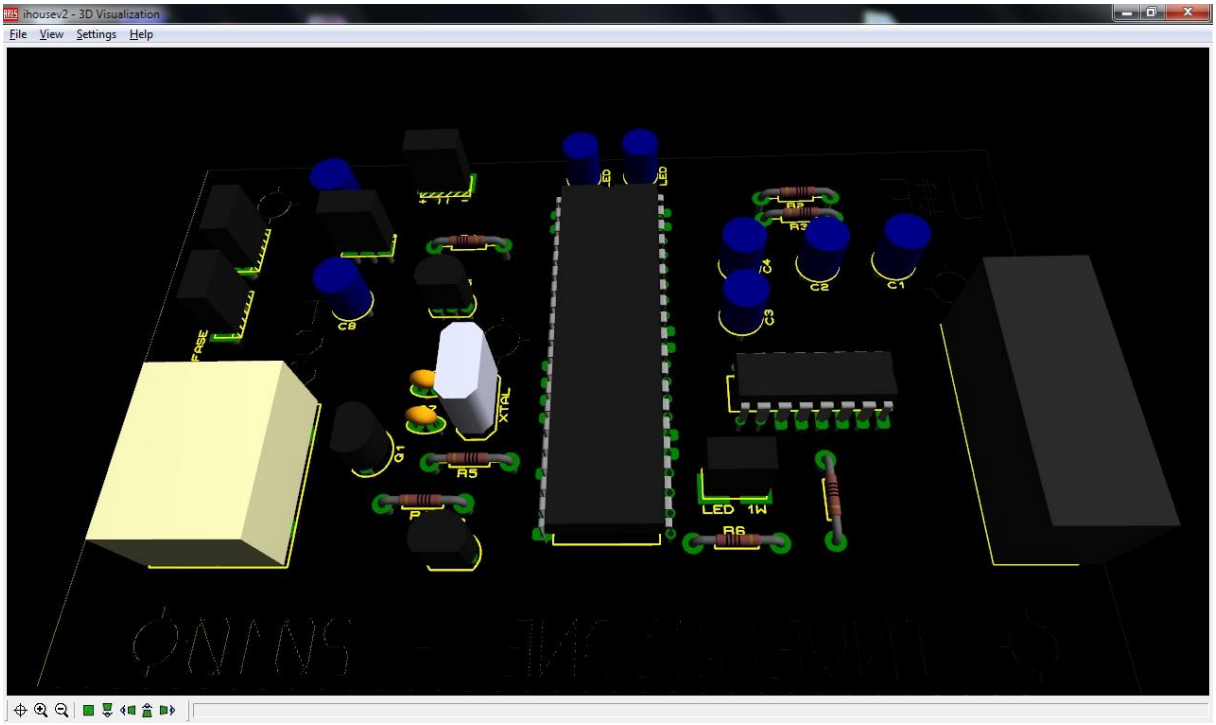


FIGURA 15 - MODELO 3D DA PLACA

4.2.2 Transferência Térmica

Com o desenho pronto, foi feita a impressão do mesmo em papel gougê 150mm em uma impressora a laser, na mais alta qualidade, pois quanto mais tonner o papel receber melhor é o resultado obtido.

Com a impressão feita, colocou-se a mesma sob a face cobreada da placa de fenolite previamente limpa, com palha de aço e álcool isopropílico. O importante é que a placa fique limpa e livre de gordura. O desenho foi afixado à placa com o auxilio de fita adesiva. Com o desenho fixo iniciou-se o processo de termo-transfência.

O processo baseia-se no fato de que ao esquentar a folha de papel o tonner derrete e se fixa ao cobre da placa. Para esquentar a folha de papel foi utilizado um ferro elétrico de passar roupas com o termostato regulado para temperatura máxima, que fica em torno dos 180º C.

Para uma boa transferência, a folha foi aquecida por volta de 4 minutos, mas sempre movimentando o ferro, pois se o mesmo ficar parado por muito tempo, em um local fixo da placa, poderá romper o cobre.

Para finalizar o processo de termo-transfêrencia foi necessário esperar a placa esfriar e retirar o papel. Para facilitar a retirada do papel, mergulhar-se a placa em um recipiente com uma solução de água e sabão e esfrega-se com os dedos.

Este processo mostra um resultado muito satisfatório, pois com ele é possível criar placas com alto nível de detalhe facilmente e com um custo reduzido, entretanto o processo mostra algumas falhas que foram corrigidas com uma caneta para marcar CD’s.

A Figura 16 mostra a placa ao final deste processo.

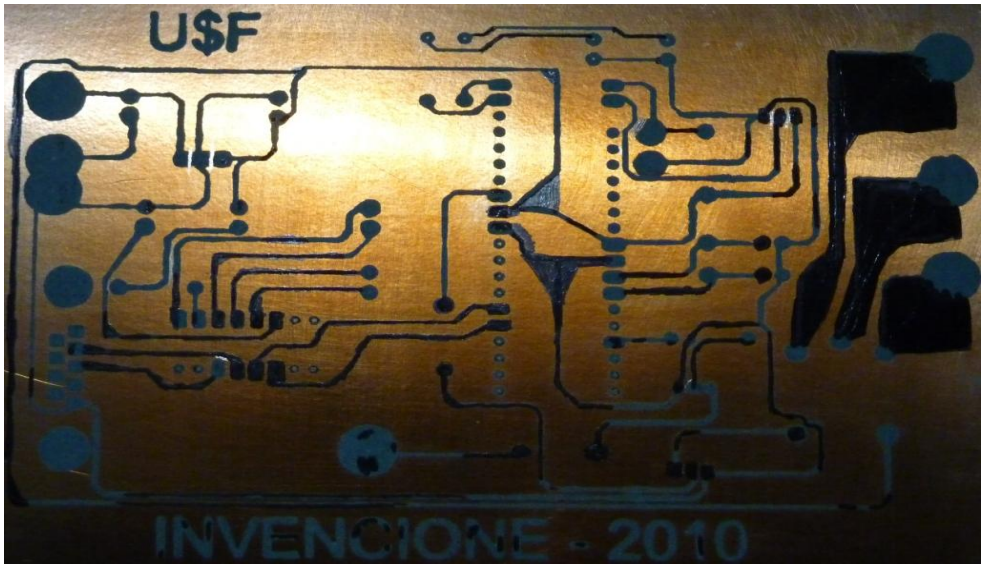


FIGURA 16 - PLACA APÓS O PROCESSO DE TERMO-TRANSFÊRENCIA E RETOQUES

4.2.3 Corrosão da PCB

Com a placa já desenhada, foi feita a corrosão da mesma. Para isso mergulhou-se a placa em percloreto de ferro, que é uma solução que ataca e corrói metais, como o cobre da placa. Todos os locais em que o cobre estava desprotegido, ou seja, sem tinta ou sem tonner, foi corroído. Ao final deste processo foi obtido o mesmo desenho da Figura 14 na placa de fenolite. A partir disto deu-se início ao processo de inserção e soldagem dos componentes finalizando assim a confecção da placa.

A Figura 17 e a Figura 18 mostram as faces inferior e superior respectivamente da placa já pronta, ou seja, com os componentes soldados a ela.

FIGURA 17 - FACE INFERIOR DA PLACA MONTADA

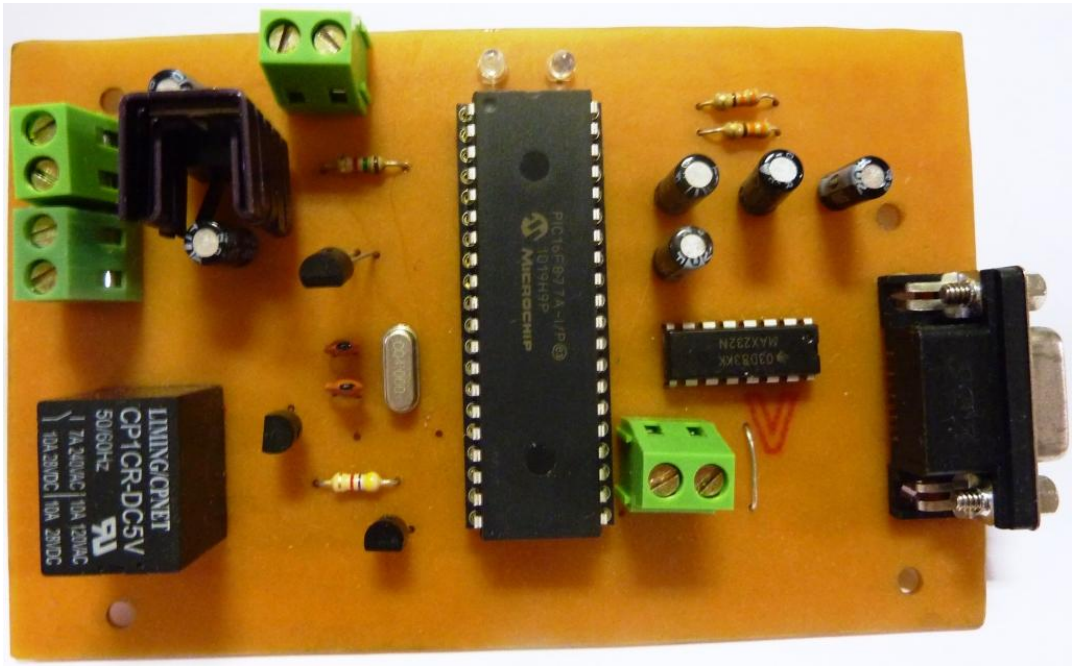


FIGURA 18 - FACE SUPERIOR DA PLACA MONTADA

4.3 Descrição de Software

4.3.1 Fluxograma do software residente no PIC

O *software* que reside na memória do PIC obedece ao fluxograma mostrado na Figura 19. Dentro de um laço infinito o PIC fica monitorando a porta serial. Quando alguma informação chegar ao barramento, esses dados são processados e as ações necessárias são enviadas aos dispositivos, como, por exemplo, acender a lâmpada. A cada ação enviada aos dispositivos uma mensagem é gerada e transmitida pela serial para o *software* de controle.

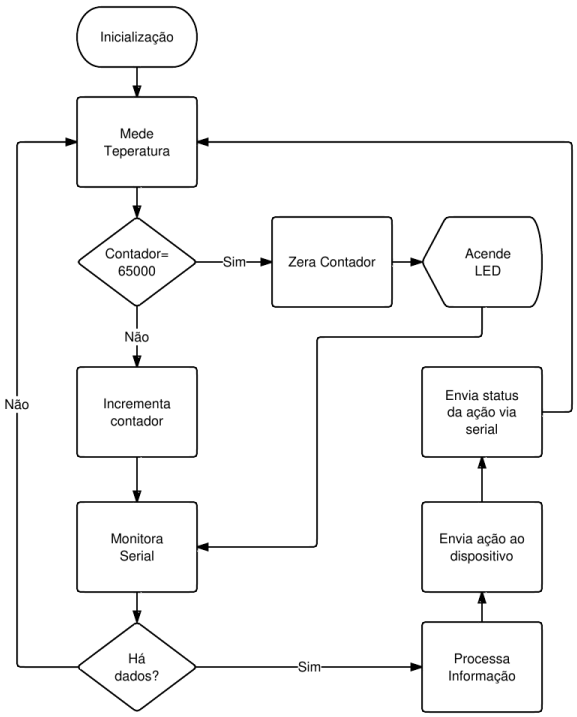


FIGURA 19 - FLUXOGRAMA CÓDIGO RESIDENTE NO PIC

4.3.2 Protocolo de comunicação

O protocolo de comunicação utilizado foi desenvolvido pelo acadêmico Guilherme da Silva Mello e o funcionamento do mesmo será descrito neste capítulo.

Neste projeto, foi necessário a criação de um protocolo de comunicação entre o PIC controlador e a aplicação *iHouse* (*software* supervisor desenvolvido pelo acadêmico Guilherme da Silva Mello). Foram definidas as regras para a conversa dos dois objetos.

Toda mensagem que trafega entre a aplicação e o PIC é composta de uma chave da conversa ou *key*. Essa chave é usada para identificar unicamente uma interação entre esses objetos e deve ser criada pela aplicação. Além disso, toda mensagem enviada pela aplicação ao controlador deve conter o código de iteração que indica o tipo da mesma. A Tabela 13 escreve os códigos de tipos de iteração e seus significados.

TABELA 13 - CÓDIGOS DOS TIPOS DE INTERAÇÕES

Constante(Serial::Interation)	Valor	Significado
DEVICE_DISCOVERY	0	Representa uma mensagem de descoberta de dispositivos na casa.
ACTION_DISCOVERY	1	Representa uma mensagem de descoberta das ações de um dispositivo.
SET_VALUE	2	Representa uma mensagem de efetivo controle sobre o dispositivo, possivelmente alteração do seu estado.
CURRENT_STATUS	3	Representa uma mensagem de questionamento de estado.

As mensagens devem seguir um padrão de formato, devendo ser sempre iniciadas e terminadas pelo caractere “#”. Seguindo o caractere inicial, deve ser concatenada a chave de comunicação e o código de interação para as mensagens enviadas pela aplicação. Toda informação deve ser separada pelo caractere “!”. A Tabela 14 exemplifica uma mensagem genérica enviada pela aplicação ao PIC controlador.

TABELA 14 - FORMATO DE UMA MENSAGEM GENÉRICA ENVIADA PELA APLICAÇÃO AO PIC

#chave!código de interação!informação 1!informação 2...!informação n#	
#	Separador de início e fim de uma interação
Chave	Chave da conversa
Código de interação	Representação do tipo da interação
!	Separador dos dados
Informação n	Dados adicionais da conversa

O formato de uma mensagem genérica de resposta do PIC à aplicação deve conter o caractere “P” logo após o separador inicial de interação. A Tabela 15 exemplifica uma mensagem genérica de resposta do PIC à aplicação.

TABELA 15 - FORMATO DE UMA MENSAGEM GENÉRICA DO PIC À APLICAÇÃO

#Pchave!informação 1!informação 2...!informação n#	
#	Separador de início e fim de uma interação
P	Identifica uma resposta do PIC a aplicação
Chave	Chave da conversa
!	Separador dos dados
Informação n	Dados adicionais da conversa

4.3.3 Descoberta de dispositivos

A descoberta de dispositivos, que ocorre quando se configura a aplicação pela primeira vez ou sempre que um novo dispositivo é adicionado a casa, é feita através de uma mensagem enviada ao PIC controlador composta de um código de descoberta, representado pelo número inteiro “911”. A Tabela 16 exemplifica uma mensagem de descoberta enviada pela aplicação ao PIC.

TABELA 16 - FORMATO DA MENSAGEM DE DESCOBERTA DE DISPOSITIVOS ENVIADA PELA APLICAÇÃO AO PIC

#chave!código de interação!911#	
#	Separador de início e fim de uma interação
Chave	Chave da conversa
Código de interação	Representação do tipo da interação
!	Separador dos dados
911	Código de descoberta de dispositivos

A mensagem de resposta do PIC à aplicação deve conter um número inteiro de identificação da classe do dispositivo, seguido do número inteiro único de identificação do mesmo. A Tabela 17 exemplifica uma resposta a mensagem de descoberta do PIC à aplicação.

TABELA 17 - FORMATO DA RESPOSTA DO PIC À APLICAÇÃO COM A MENSAGEM DE DESCOBERTA DOS DISPOSITIVOS

#Pchave!número da classe do dispositivo n!identificação do dispositivo n#	
#	Separador de início e fim de uma interação
P	Identifica uma resposta do PIC a aplicação
Chave	Chave da conversa
!	Separador dos dados
Número da classe do dispositivo n	Identifica o tipo do dispositivo
Identificação do dispositivo n	Identifica unicamente o dispositivo presente na casa

4.3.4 Descoberta das ações de um dispositivo

Para descobrir as possíveis ações que um dispositivo pode executar, é necessária uma mensagem que contenha o código de identificação de dispositivo e o código inteiro de descoberta de ações "912". A Tabela 18 exemplifica a mensagem enviada pela aplicação ao PIC a fim de descobrir as ações de um dispositivo.

TABELA 18 - FORMATO DA MENSAGEM DE DESCOBERTA DAS AÇÕES DE UM DISPOSITIVO ENVIADA PELA APLICAÇÃO AO PIC

#chave!código de interação!identificação do dispositivo!912#	
#	Separador de início e fim de uma interação
Chave	Chave da conversa
Código de interação	Representação do tipo da interação
!	Separador dos dados
Identificação do dispositivo	Código numérico que identifica unicamente o dispositivo
912	Código de descoberta das ações de um dispositivo

Como resposta, a mensagem enviada pelo PIC deve conter o código de comando da ação, o código de questionamento de estado, e o código de tipo de ação, onde "0" representa uma ação de tipo *TURN_ON_OFF* booleana, e "1" representa uma ação do tipo *RANGE* de faixa de valores. Caso o tipo da ação seja de faixa de valores, o PIC deve ainda enviar os valores mínimo e máximo da faixa. A Tabela 19 exemplifica a resposta do PIC a aplicação.

TABELA 19 - FORMATO DA MENSAGEM DE RESPOSTA DO PIC À APLICAÇÃO DE UMA DESCOBERTA DE AÇÕES DO DISPOSITIVO

#Pchave!código do comando!código de questionamento de estado!tipo da ação[!valor mínimo!valor máximo]#	
#	Separador de início e fim de uma interação
P	Identifica uma resposta do PIC a aplicação
Chave	Chave da conversa
!	Separador dos dados
Código do comando	Código inteiro do comando da ação
Código de questionamento de estado	Código inteiro que indica o valor a ser usado para se obter o estado da ação
Tipo da ação	Inteiro "0" representa ação booleana, "1" faixa de valores
Valor mínimo	Valor mínimo da faixa de valores
Valor máximo	Valor máximo da faixa de valores

4.3.5 Envio de ação

Quando o usuário controla os dispositivos através da aplicação *iHouse*, são enviados ações ao controlador para que os atuadores executem-nas. A mensagem enviada pela aplicação ao PIC é composta do identificador inteiro único do dispositivo, o código do comando da ação e o valor a ser modificado. A Tabela 20 exemplifica a mensagem de ação enviada pela aplicação ao PIC controlador.

TABELA 20 - FORMATO DA MENSAGEM DE AÇÃO ENVIADA PELA APLICAÇÃO AO PIC

#chave!código de interação!identificação do dispositivo!código do comando!valor#	
#	Separador de início e fim de uma interação
Chave	Chave da conversa
Código de interação	Representação do tipo da interação
!	Separador dos dados
Identificação do dispositivo	Código numérico que identifica unicamente o dispositivo
Código do comando	Código numérico que identifica unicamente uma ação
Valor	Inteiro que representa um valor a ser enviado ao atuador

A resposta do PIC deve conter o estado da execução da ação. Caso o estado seja “200”, houve sucesso, caso contrário, algum tipo de erro ocorreu e o código será “500”. A Tabela 21 exemplifica a resposta do PIC à aplicação de uma mensagem de ação.

TABELA 21 - FORMATO DA RESPOSTA DO PIC À APLICAÇÃO DE UMA MENSAGEM DE AÇÃO

#Pchave!código de estado#	
#	Separador de início e fim de uma interação
P	Identifica uma resposta do PIC a aplicação
Chave	Chave da conversa
!	Separador dos dados
Código de estado	Código de representação de sucesso (200) ou erro (500)

4.3.6 *Feedback* / questionamento de estado

Para se obter o *feedback* ou fazer o questionamento do estado de uma ação do dispositivo, é necessário uma mensagem que contenha o identificador do dispositivo, o código de comando da ação e o código de questionamento de estado. A Tabela 22 exemplifica a mensagem enviada pela aplicação ao PIC a fim de se obter o estado de uma ação.

TABELA 22 - FORMATO DA MENSAGEM DE QUESTIONAMENTO DE ESTADO ENVIADA PELA APLICAÇÃO AO PIC

#chave!código de interação!identificação do dispositivo!código do comando!código de questionamento de estado#	
#	Separador de início e fim de uma interação
Chave	Chave da conversa
Código de interação	Representação do tipo da interação
!	Separador dos dados
Identificação do dispositivo	Código numérico que identifica unicamente o dispositivo
Código do comando	Código numérico que identifica unicamente uma ação
Código de questionamento de estado	Código que é interpretado pelo PIC como busca do estado da ação

O PIC deve responder com uma mensagem que contenha um valor numérico do estado, caso a ação seja do tipo *RANGE* de faixa de valores. Caso contrário a mensagem deve conter a *string* "*state_on*" para a resposta booleana "ligado" ou "*state_off*" para resposta booleana "desligado". A Tabela 23 exemplifica a resposta do PIC à aplicação.

TABELA 23 - FORMATO DA RESPOSTA DO PIC À APLICAÇÃO DE UMA MENSAGEM DE QUESTIONAMENTO DE ESTADO

#Pchave!código do estado#	
#	Separador de início e fim de uma interação
P	Identifica uma resposta do PIC a aplicação
Chave	Chave da conversa
!	Separador dos dados
Código do estado	Código de representação numérico para ações do tipo faixa de valores ou *"state_on"* (ligado) e *"state_off"* (desligado) para ações do tipo booleana

4.3.7 Código fonte

O código fonte do *firmware* deste projeto é apresentado no APêNDICE – CÓDIGO Fonte DO PROGRAMA.

5 CONTROLE REMOTO VIA WEB

O projeto aqui descrito visou desenvolver um *hardware* para automação residencial. O sistema supervisor *web*, chamado *iHouse*, foi desenvolvido pelo acadêmico Guilherme da Silva Mello utilizando Ruby *on* Rails.

6 CONCLUSÃO

Mesmo sendo um protótipo, com o desenvolvimento deste projeto nota-se que é possível criar soluções interessantes e compatíveis com o que existe no mercado para automação residencial, baseando o processamento do sistema em microcontroladores PIC, afinal eles são facilmente encontrados no mercado, são baratos, se mostram extremamente versáteis, pois possuem diversos módulos implementados por *hardware* tais como USART, PWM, Conversor A/D, diversos pinos configuráveis para entrada e saída.

O uso do compilador MickroC para o desenvolvimento do *firmware* foi essencial, pois o mesmo além de seguir o padrão ANSI[19], possui uma vasta biblioteca com funções específicas para os módulos encontrados no PIC o que agiliza, e muito, o desenvolvimento. Entretanto funções para trabalhar com alocação dinâmica de memória não estão disponíveis no compilador utilizado, para se contornar o problema de falta de memória do PIC foi adotada outra solução, definir as estruturas de dados estaticamente.

Como extensão do trabalho sugere-se a implementação de mais recursos a placa, como por exemplo, sistemas de segurança, utilizando sensores para se monitorar a intrusão a casa. Seria interessante também substituir a comunicação RS-232 por um padrão mais atual e de preferência sem fio, como por exemplo, o ZigBee ou até mesmo Wi-Fi.

[19] *American National Standards Institute*

7 BIBLIOGRAFIA

[1] GERALDO, Pedro. **A ORIGEM DO HOMEM E A PRÉ-HISTÓRIA**. Disponível em: <www.vestibular1.com.br/revisao/origem_homem_america.doc>. Acesso em: 26 fev. 2010.

[2] SOTERO, Anna Paula. **Dos Transistores aos Computadores**. Disponível em: <http://www.comciencia.br/reportagens/fisica/fisica11.htm>. Acesso em: 26 fev. 2010.

[3] **Os Jetsons**. Disponível em: <http://mundodasmarcas.blogspot.com/2006/09/os-jetsons.html>. Acesso em: 26 fev. 2010.

[4] P&S ENERGIA, **Automação residencial ou Domótica**. Disponível em: <http://www.pesenergia.com.br/domotica_7.html>. Acesso em: 26 fev. 2010.

[5] SCHÜTZER Waldeck, **O que é o PIC?** Disponível em: <http://www.dm.ufscar.br/~waldeck/pic/#pic> Acesso em: 10 abr. 2010.

[6] TREVISAN V.T PEDRO, **Microcontroladores PIC**. Disponível em: http://www2.brazcubas.br/professores1/arquivos/20_franklin/T7037A/Microcontroladores_Pic_-_Apostila.pdf. Acesso em 10 abr 2010.

[7] CNZ Engenharia e Informática Ltda, **Mini Curso: Comunicação Serial – RS232**. Disponível em: http://docs.google.com/viewer?a=v&q=cache:dG3j7WuKMkMJ:www.verlab.dcc.ufmg.br/_media/cursos/introrobotica/2009-2/comunicacao_serial.pdf+comunicação+serial&hl=pt-BR&gl=br&pid=bl&srcid=ADGEESgNChQDQkqoofdd8sEa_A9tTpu8yKO7xQ6GqmTK8EfHpcF0Sdg4s9jtwX3j7_bquzVFpvbwU7NT9KDcNdS2A_Ad19Ou-9GE-fF0h9fPkLhaL_BOgmD2vG7LbymPgQK5ojdMey69&sig=AHIEtbQpg4GE9hRKnWAegBxnWrl9L7xCVA. Acesso em 10 abr 2010.

[8] ZANCO, Wagner da Silva. **Microcontroladores PIC Técnicas de Software e Hardware para Projetos de Circuitos Eletrônicos Com Base no PIC 16F877A**. 1. ed. São Paulo: Érica, 2006. 390 p

[9] SOLBET Ltda, **Programação de Microcontroladores: Considerações básicas sobre o PIC 16F877A.** Disponível em: < http://www.ebah.com.br/consideracoes-basicas-sobre-o-pic16f877a-pdf-a16740.html>. Acesso em: 16 abr. 2010.

[10] SOUZA, David José de, LAVINIA, Nicolás César. **PIC16F877A Conectando o PIC Recursos Avançados**. 4. Ed. São Paulo: Érica, 2007. 380 p.

[11] National Semiconductor Corporation, **Overview Precision Centigrade Temperature Sensor.** Disponível em: < http://www.national.com/mpf/LM/LM35.html#Overview > Acesso em 16.jun.2010.

[12] METALATEX, **COMO FUNCIONAM OS RELÉS.** Disponível em: < http://www.metaltex.com.br/tudosobrereles/tudo1.asp > Acesso em 16.jun.2010.

[13] **Transistores Bipolares**. Disponível em: < http://www.arvm.org/exames/trasistor.htm > Acesso em 16.jun.2010.

MICROCHIP. **PIC16F87XA Data Sheet**. Disponível em: ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf. Acesso em 15 fev 2010.

8 APÊNDICE – Código Fonte do Programa

```
#define TAMANHO_PALAVRA 50

unsigned char caracter_recebido,cpwm,duty_pwm, index;
char palavra_recebida[TAMANHO_PALAVRA];
char chave[3], temp_str[7];
char tipo_interacao, temperatura_casa = 25;
char mensagem[TAMANHO_PALAVRA];
char dispositivo[6];
char acao[6];
char valor_enviado[4];
unsigned char index_anterior;
int tensao, temperatura, valor_atual_pwm = 0;
unsigned int contador_loop = 0;

void envia_mensagem(char *mensagem);
void inicializa_vetor(char *vetor, char tamanho);
void escreve_cabecalho(char *vetor, char *chave);
void atualiza_temperatura();
void escreve_acoes_lampada_led(char *vetor);
void escreve_mensagem_de_estado(char *vetor, int on);


void main() {

 TRISB = 0x00;
 ADCON1 = 0x80;  // Configura entrada analogical Vref
 TRISA  = 0xFF;  // PORTA entrada
 TRISC = 0;              // PORTC saída
 TRISD = 0;
 PORTD.F1 = 1;
 Pwm_Init(1000);
 Usart_Init(9600);
 PORTB.F6 = 0;
 PORTB.F7 = 0;
 Pwm_Start();

 do
 {
   tensao       = Adc_Read(0);
   temperatura   = tensao * 0.48828125;

   if(contador_loop == 65000)
   {
    atualiza_temperatura();
    contador_loop = 0;
   }
   else
```

```
  contador_loop++;

if (Usart_Data_Ready())
{   // If data is received
  caracter_recebido = Usart_Read();         // Read the received data

  if(caracter_recebido == '#')
  {
   inicializa_vetor(palavra_recebida, TAMANHO_PALAVRA);

   index = 1;
   palavra_recebida[0] = caracter_recebido;
   caracter_recebido = 0;

   while(caracter_recebido != '#')
   {
    if (Usart_Data_Ready())
    {
     caracter_recebido = Usart_Read();
     palavra_recebida[index] = caracter_recebido;
     index++;
    }
   }

   inicializa_vetor(chave, 3);
   inicializa_vetor(mensagem, TAMANHO_PALAVRA);

   index_anterior = 1;

  for(index = index_anterior; index < strlen(palavra_recebida); index++)
  {
          if(palavra_recebida[index] != '!')
          {
           chave[index - index_anterior] = palavra_recebida[index];
          }
          else
          {
           index_anterior = index + 1;
           break;
          }
  }

  tipo_interacao = palavra_recebida[index_anterior];
  index_anterior += 2;

  for(index = index_anterior; index < strlen(palavra_recebida); index++)
  {
          if(palavra_recebida[index] != '#')
          {
           mensagem[index - index_anterior] = palavra_recebida[index];
```

```
        }
        else
          break;
}

escreve_cabecalho(palavra_recebida, chave);

// tipo_interação = 2 setar valor || tipo_interação = 3 estado da ação
if(tipo_interacao == '2' || tipo_interacao == '3')
{
  inicializa_vetor(dispositivo, 6);
  index_anterior = 0;

  for(index = index_anterior; index < 6; index++)
  {
        if(mensagem[index] != '!')
        {
          dispositivo[index - index_anterior] = mensagem[index];
        }
        else
        {
          index_anterior = index + 1;
          break;
        }
  }

  inicializa_vetor(acao, 6);
  for(index = index_anterior; index < strlen(mensagem); index++)
  {
        if(mensagem[index] != '!')
        {
          acao[index - index_anterior] = mensagem[index];
        }
        else
        {
          index_anterior = index + 1;
          break;
        }
  }

  inicializa_vetor(valor_enviado, 4);
  for(index = index_anterior; index < strlen(mensagem); index++)
     valor_enviado[index - index_anterior] = mensagem[index];

  if(atoi(dispositivo) == 1)
  {
   if(atoi(acao) == 1)
   {
     if(tipo_interacao == '2')
     {
```

```
      if (atoi(valor_enviado))
        PORTD.F1 = 0;
      else
        PORTD.F1 = 1;

      palavra_recebida[strlen(palavra_recebida)] = '2';
      palavra_recebida[strlen(palavra_recebida)] = '0';
      palavra_recebida[strlen(palavra_recebida)] = '0';
     }
     else // interacao 3
     {
       escreve_mensagem_de_estado(palavra_recebida, PORTD.F1 == 1);
     }

     palavra_recebida[strlen(palavra_recebida)] = '#';

     envia_mensagem(palavra_recebida);
   }
 }
 else if (atoi(dispositivo) == 4)
 {
     if (atoi(acao) == 4)
     {
       tensao       = Adc_Read(0);
       temperatura  = tensao * 0.48828125;

       if (tipo_interacao == '3')
       {
         IntToStr(temperatura, temp_str);

         for(index = 0; index < strlen(temp_str); index++)
         {
            if (temp_str[index] == '0' ||
                temp_str[index] == '1' ||
                temp_str[index] == '2' ||
                temp_str[index] == '3' ||
                temp_str[index] == '4' ||
                temp_str[index] == '5' ||
                temp_str[index] == '6' ||
                temp_str[index] == '7' ||
                temp_str[index] == '8' ||
                temp_str[index] == '9')
              palavra_recebida[strlen(palavra_recebida)] = temp_str[index];
         }
       }
       else
       {
          temperatura_casa = atoi(valor_enviado);
          atualiza_temperatura();
```

```
            palavra_recebida[strlen(palavra_recebida)] = '2';
            palavra_recebida[strlen(palavra_recebida)] = '0';
            palavra_recebida[strlen(palavra_recebida)] = '0';
          }

          palavra_recebida[strlen(palavra_recebida)] = '#';
          envia_mensagem(palavra_recebida);
        }

    }
    else if (atoi(dispositivo) == 5)
    {
         if (atoi(acao) == 5)
        {

          if (tipo_interacao == '3')
          {
            IntToStr(valor_atual_pwm, temp_str);

            for(index = 0; index < strlen(temp_str); index++)
            {
               if (temp_str[index] == '0' ||
                  temp_str[index] == '1' ||
                  temp_str[index] == '2' ||
                  temp_str[index] == '3' ||
                  temp_str[index] == '4' ||
                  temp_str[index] == '5' ||
                  temp_str[index] == '6' ||
                  temp_str[index] == '7' ||
                  temp_str[index] == '8' ||
                  temp_str[index] == '9')
                palavra_recebida[strlen(palavra_recebida)] = temp_str[index];
            }
          }
          else
          {
             valor_atual_pwm = atoi(valor_enviado);
             Pwm_Change_Duty(valor_atual_pwm);

             palavra_recebida[strlen(palavra_recebida)] = '2';
             palavra_recebida[strlen(palavra_recebida)] = '0';
             palavra_recebida[strlen(palavra_recebida)] = '0';
          }

          palavra_recebida[strlen(palavra_recebida)] = '#';
          envia_mensagem(palavra_recebida);
        }
        else if (atoi(acao) == 1)
         {
```

```
            if (tipo_interacao == '3')
            {
               escreve_mensagem_de_estado(palavra_recebida, valor_atual_pwm != 0);
            }
            else
            {
               valor_atual_pwm = (atoi(valor_enviado) == 1) ? 255 : 0;
               Pwm_Change_Duty(valor_atual_pwm);

               palavra_recebida[strlen(palavra_recebida)] = '2';
               palavra_recebida[strlen(palavra_recebida)] = '0';
               palavra_recebida[strlen(palavra_recebida)] = '0';
            }

            palavra_recebida[strlen(palavra_recebida)] = '#';
            envia_mensagem(palavra_recebida);
         }
      }
   }
   // tipo_interação = 0 descoberta de dispositivo
   else if(tipo_interacao == '0')
   {
      //lâmpada
      palavra_recebida[strlen(palavra_recebida)] = '1';
      palavra_recebida[strlen(palavra_recebida)] = '!';
      palavra_recebida[strlen(palavra_recebida)] = '1';

      palavra_recebida[strlen(palavra_recebida)] = '!';

      //temperatura
      palavra_recebida[strlen(palavra_recebida)] = '4';
      palavra_recebida[strlen(palavra_recebida)] = '!';
      palavra_recebida[strlen(palavra_recebida)] = '4';

      palavra_recebida[strlen(palavra_recebida)] = '!';

      //lâmpada LED
      palavra_recebida[strlen(palavra_recebida)] = '1';
      palavra_recebida[strlen(palavra_recebida)] = '!';
      palavra_recebida[strlen(palavra_recebida)] = '5';

      palavra_recebida[strlen(palavra_recebida)] = '#';

      envia_mensagem(palavra_recebida);
   }
   // = 1 descoberta de ações
   else if(tipo_interacao == '1')
   {
      inicializa_vetor(dispositivo, 6);
      index_anterior = 0;
```

```
    for(index = index_anterior; index < 6; index++)
    {
          if(mensagem[index] != '!')
          {
           dispositivo[index - index_anterior] = mensagem[index];
          }
          else
          {
           index_anterior = index + 1;
           break;
          }
    }

    if (atoi(dispositivo) == 1)
    {
       // ação liga/desliga lâmpada
       palavra_recebida[strlen(palavra_recebida)] = '1';
       palavra_recebida[strlen(palavra_recebida)] = '!';
       palavra_recebida[strlen(palavra_recebida)] = '1';
       palavra_recebida[strlen(palavra_recebida)] = '!';
       palavra_recebida[strlen(palavra_recebida)] = '0';

    }
    else if (atoi(dispositivo) == 4)
    {
       // ação mede temperatura
       palavra_recebida[strlen(palavra_recebida)] = '4';
       palavra_recebida[strlen(palavra_recebida)] = '!';
       palavra_recebida[strlen(palavra_recebida)] = '4';
       palavra_recebida[strlen(palavra_recebida)] = '!';
       palavra_recebida[strlen(palavra_recebida)] = '1';
       palavra_recebida[strlen(palavra_recebida)] = '!';
       palavra_recebida[strlen(palavra_recebida)] = '5';
       palavra_recebida[strlen(palavra_recebida)] = '!';
       palavra_recebida[strlen(palavra_recebida)] = '3';
       palavra_recebida[strlen(palavra_recebida)] = '4';
    }
    else if (atoi(dispositivo) == 5)
    {
        escreve_acoes_lampada_led(palavra_recebida);
    }

    palavra_recebida[strlen(palavra_recebida)] = '#';

    envia_mensagem(palavra_recebida);
  }

 }
}
```

```
  }while(1);

}//main

void envia_mensagem(char *mensagem)
{
  Usart_Write(' ');
  Usart_Write(' ');
  Usart_Write(' ');

  for(index = 0; index < strlen(mensagem); index++)
  {
     Usart_Write(mensagem[index]);
  }

  Usart_Write('\n');
}

void inicializa_vetor(char *vetor, char tamanho)
{
    for(index = 0; index < tamanho; index++)
      vetor[index] = '\0';
}

void escreve_cabecalho(char *vetor, char *chave)
{
    inicializa_vetor(palavra_recebida, TAMANHO_PALAVRA);

    vetor[0] = '#';
    vetor[1] = 'P';

    for(index = 0; index < strlen(chave); index++)
      vetor[strlen(vetor)] = chave[index];

    vetor[strlen(vetor)] = '!';
}

void atualiza_temperatura()
{
   if (temperatura_casa > temperatura)
    {
      PORTB.F6 = 1;
      PORTB.F7 = 0;
    }
    else if (temperatura_casa < temperatura)
    {
      PORTB.F6 = 0;
      PORTB.F7 = 1;
    }
    else
```

```
    {
      PORTB.F6 = 0;
      PORTB.F7 = 0;
    }
}

void escreve_acoes_lampada_led(char *vetor)
{
  //ação intensidade da lâmpada LED
  vetor[strlen(vetor)] = '5';
  vetor[strlen(vetor)] = '!';
  vetor[strlen(vetor)] = '5';
  vetor[strlen(vetor)] = '!';
  vetor[strlen(vetor)] = '1';
  vetor[strlen(vetor)] = '!';
  vetor[strlen(vetor)] = '0';
  vetor[strlen(vetor)] = '!';
  vetor[strlen(vetor)] = '2';
  vetor[strlen(vetor)] = '5';
  vetor[strlen(vetor)] = '5';

  vetor[strlen(vetor)] = '!';
  vetor[strlen(vetor)] = '1';
  vetor[strlen(vetor)] = '!';
  vetor[strlen(vetor)] = '1';
  vetor[strlen(vetor)] = '!';
  vetor[strlen(vetor)] = '0';
}

void escreve_mensagem_de_estado(char *vetor, int on)
{
   vetor[strlen(vetor)] = 's';
   vetor[strlen(vetor)] = 't';
   vetor[strlen(vetor)] = 'a';
   vetor[strlen(vetor)] = 't';
   vetor[strlen(vetor)] = 'e';
   vetor[strlen(vetor)] = '_';

   if(on)
   {
     vetor[strlen(vetor)] = 'o';
     vetor[strlen(vetor)] = 'n';
   }
   else
   {
     vetor[strlen(vetor)] = 'o';
     vetor[strlen(vetor)] = 'f';
     vetor[strlen(vetor)] = 'f';
   }
}
```